МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В.Я. Юрчишин

ХМАРНІ ТА ГРІД - ТЕХНОЛОГІЇ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» (освітня програма «Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

Київ
«КПІ ім. Ігоря Сікорського»
2019

Рецензенти: *Тимошин Юрій Афанасійович, канд. техн. наук, доц.*
Зубчукчук Віктор Іванович, канд. техн. наук, доц.
Відповідальний редактор *Легеза Віктор Петрович, д-р. техн. наук, проф.*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол №10 від 20.06.2019 р.) за поданням Вченої ради факультету прикладної математики (протокол № 11 від 18.06.2019 р.)

Електронне мережне навчальне видання

Юрчишин Василь Якович, к. т. н, доц.

ХМАРНІ ТА ГРІД - ТЕХНОЛОГІЇ
КОНСПЕКТ ЛЕКЦІЙ

Хмарні та Грід-технології: конспект лекцій [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)/ В.Я.Юрчишин; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 5,93 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 264 с.

Навчальний посібник розроблено для вивчення студентами теоретичних відомостей та ознайомлення з практичними прийомами роботи по розробці програмних продуктів в області хмарних та грід-технологій.

Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення (освітня програма «Програмне забезпечення комп'ютерних та інформаційно-пошукових систем») освітньо-кваліфікаційного рівня Магістр факультету прикладної математики НТУУ КПІ ім. Ігоря Сікорського.

ЗМІС

ВСТУП

Застосування інформаційних технологій на базі хмарних та Грід-систем сприяють розробці сучасного програмного забезпечення, прикладом якого можуть виступати широкомасштабні системи моніторингу, управління та аналізу з глобально розподіленими джерелами даних. Значення хмарних та грід-технологій є одними з найважливішими в світі, завдяки їм границі країн, відстані та фізичні обмеження перестають мати минулі значення.

В даному навчальному посібнику розглядаються теоретичні та практичні основи Грід- і хмарних технологій та розподілених обчислень, віртуалізації серверних систем, проектування корпоративних обчислювальних систем та застосування кластерних та гетерогенних розподілених обчислювальних систем із використанням мережевих протоколів взаємодії клієнтських та серверних додатків.

Навчальна дисципліна належить до циклу професійно-орієнтовпаних дисциплін навчального плану підготовки студентів спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Програмне забезпечення комп'ютерних та інформаційно-пошукових систем») освітньо-кваліфікаційного рівня Магістр.

Предметом дисципліни є теоретичні та практичні основи програмування в області хмарних та грід-технологій. Мета дисципліни – забезпечити знання теоретичних і практичних основ грід- та хмарних технологій.

Метою конспекту лекцій є отримання необхідного рівня знань і придбання практичних умінь і навичок з використання розподілених обчислень, віртуалізації серверних систем, створення програмних продуктів для проведення складних обчислень в локальних та глобальних мережах.

Курс лекцій розрахований на 36 академічних годин аудиторних занять. Посібник складається з 18 лекцій. В кожній лекції приводяться теоретичні відомості з певної теми, контрольні питання для самоконтролю та список рекомендованої літератури.

Розділ 1. Розподілені комп'ютерні системи

1.1 Розподілені комп'ютерні системи і їх програмування

Єдиного визначення розподіленої комп'ютерної системи (РКС) в даний час не тіснує. Із множини різних визначень можна привести іронічне висловлення Лесли Лампорта [5]: «Розподіленою комп'ютерною системою можна назвати таку систему, в якій відмова комп'ютера, про існування якого ви навіть не підозрювали, може зробити ваш власний комп'ютер непридатним до використання». Це визначення він дав в 1987р в листі колегам по поводу чергового відключення електроенергії в машинному залі.

Ендрю Таненбаум в своїй фундаментальній праці «Распределённые системы. Принципы и парадигмы» [4] запропунував більш строге визначення: «Рорзподілена обчислювальна система (РОС) – це набір з'єднаних каналами зв'язку незалежних комп'ютерів, які з точки зору користувача деякого програмного забезпечення виглядають єдиним цілим». В цьому визначенні фіксуються два суттевих моменти: автономність вузлів РКС і представлення системи користувачем, як єдиної структури. При цьому, основним зв'язуючим елементом РОС являється програмне забезпечення. В кінці минулого століття Ян Форстер визначив задачу розподілених обчислювальних систем як гнучке, безпечне, координоване розподілення ресурсів серед динамічних наборів користувачів, організацій і ресурсів [4]. Він запропонував назвати такі розподілені обчислювальні системи терміном Грід. Розвиваючи ідею Грід, комерційні розробники в 2007-2008р запропонували концепцію хмарних обчислень, в основі якої було надання високомасштабованих віртуальних обчислювальних ресурсів кінцевому користувачу через інтернет в вигляді послуг.

РКС представляє собою програмно-апаратний комплекс, орієнтований на вирішення певних задач. З однієї сторони кожний обчислювальний вузол являється автономним елементом, а з іншої програмна складова РОС має забезпечити користувачам видимість роботи з єдиною обчислювальною системою. В зв'язку з цим виділяють наступні важливі характеристики РОС, а саме можливість роботи з:

- різними типами пристроїв;
- різними постачальниками пристроїв;
- різними апаратними платформами;
- різними операційними системами.

Має бути забезпечена можливість простого розширення і масштабування, постійна доступність ресурсів і прихованість особливостей комунікацій від користувачів.

Для забезпечення роботи гетерогенного обладнання РОС як єдиного

цілого, стек програмного забезпечення зазвичай розбивають на два шари. На верхньому шарі розміщуються розподілені додатки, відповідаючі за вирішення певних прикладних задач засобами РОС. Їх функціональні можливості базуються на нижньому шарі – проміжному програмному забезпеченні *(*ППО*)* . ППО взаємодіє з системним ПО і мережним рівнем для забезпечення прозорості роботи додатків в РОС (рис. 1.1). Щоб РОС могла бути представлена користувачу як єдина система, використовують наступні типи прозорості в РОС:

– прозорий доступ до ресурсів – від користувачів має бути прихована різниця в представленні даних і в способах доступу до ресурсів РОС;

– прозоре місцезнаходження ресурсів – місце фізичного розташування потрібного ресурсу має бути несуттевим для користувача;

– реплікація – приховання від користувача того, що в реальності існує більше однієї копії використовуємих ресурсів;

– паралельний доступ – можливість одночасного використання одного і того ж ресурсу різними користувачами незалежно один від одного. При цьому факт сумісного використання ресурсу має залишатися прихованим від користувача;

– прозорість відмов – відмова (відключення) яких-небудь ресурсів РОС не повинна впливати на работу користувача і його додаток.

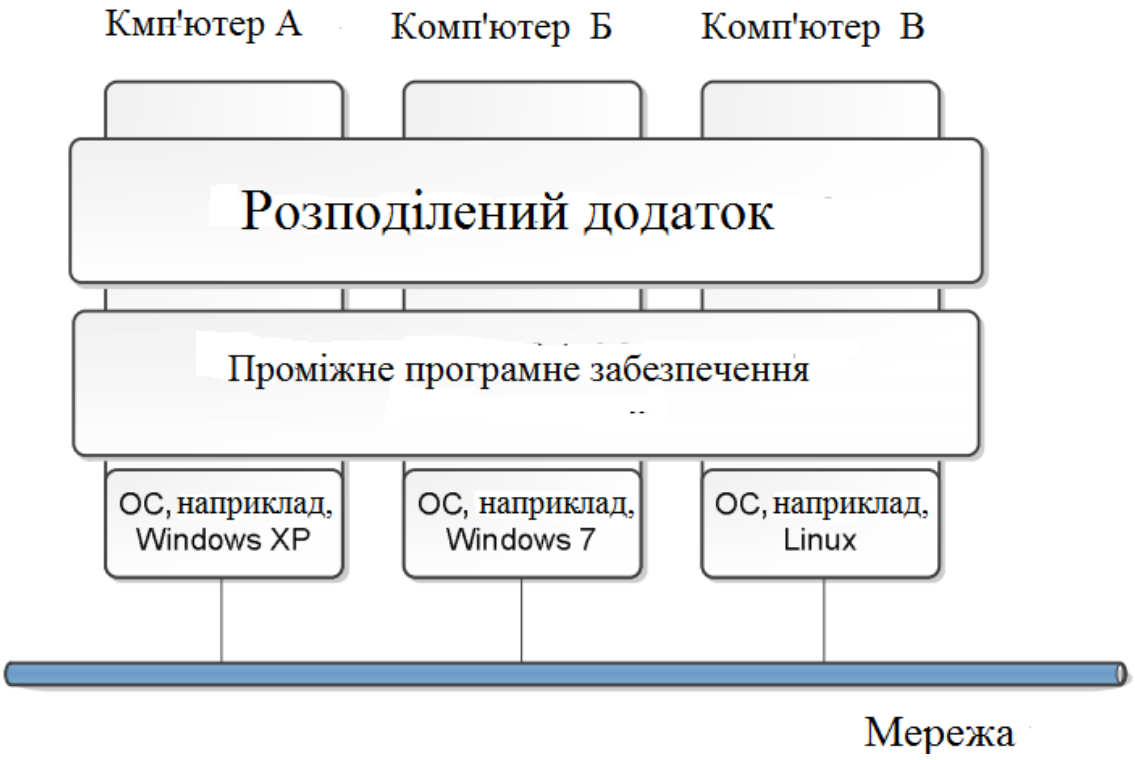


Рис.1.1 Шари програмного забезпечення в РОС

Отже, організація обчислень в розподілених комп'ютерних системах ґрунтується на мережевих технологіях і має певні особливості. Структура розподіленої комп'ютерної системи складається з вузлів, об'єднаних за допомогою комп'ютерної мережі (рис. 1.2) . В якості вузлів можуть використовуватися персональні комп'ютери (ПК), робочі станції, або багатопроцесорні комп'ютерні системи (КС). Кожний вузол РКС працює під керуванням власної операційної системи (КС з локальною пам'яттю використовують лише одну операційну систему, яка керує всіма процесорами). Структура РКС зображена на рис. 1.2.

Рис. 1.2. Структура РКС

Взаємодія вузлів в РКС ґрунтується на посиланні повідомлень з використанням мережевих технологій, що потребує використання спеціальних програмних засобів – сокетів, віддалених методів, в яких задіяні протоколи, IP – адреси та інше [1],[2].

Можливість об'єднати десятки тисяч і навіть сотні тисяч комп'ютерів дозволяє створювати дуже потужні розподілені системи, які легко розширюються, пребудовуються і значно дешевші.

Кластерні, мультикластерні та Грід – це типові розподілені обчислювальні системи. В розподілених системах спільне скоординоване використання ресурсів, різних за своєю структурою, підтримується відповідними технологіями та інфраструктурами.

У кластерах передбачена плавна зміна розмірів і подолання відмов: коли один підвузол у кластері відмовляє, то інший готовий зайняти його місце так, щоб додатки (включаючи і його користувачів) практично цього не помітили.

Ранні кластери були більш-менш закритими системами; це були процесори із сильним зв'язком, підключені до загальної шини оперативної і зовнішньої пам'яті і колективно використовуючі дані на високих швидкостях. Розширення можливостей інформаційного обміну дозволили кластерам покинути

середовище колективної шини й почати існування в значно більшому масштабі. Такий кластер залежить від якості й швидкості виділеного з'єднання для передачі даних. Глобальний кластер використовує відкритий Інтернет для передачі даних. Багатоядерні кластерні системи забезпечують два рівня обробки – на множині ядер і на множині вузлів. Це ускладнює розробку програмного забезпечення і потребує використання великої кількості взаємодіючих процесів у розподіленій програмі.

Сучасні РКС є мультиархітектурними і для них характерні ієрархічна організація і різні пропускні спроможності каналів зв'язку між їхніми ресурсами (вузлами, процесорами і їх ядрами). В архітектурному плані РКС це сукупність взаємодіючих елементарних комп'ютерів (ЕК) з засобами комунікації та зовнішніми пристроями. Кількість ЕК в РКС допускає варіювання від одиниць до сотень тисяч (наприклад, в МВС-15000ВМ їх 1148, в IBM Roadrunner – 122400, в IBM BlueGene – 884736). Ефективність виконання завдання суттєво залежить від досягнення мінімуму накладних витрат на міжмашинний обмін інформацією і дисбаланс завантаження ЕК. Програмне забезпечення для програмування в РКС ґрунтується на засобах, пов'язаних з особливостями передавання даних в комп'ютерних мережах. Такі засоби представлені в вигляді спеціальних бібліотек (WinAPI, PVM, MPI). Мови програмування Java, Ада, C# мають вбудовані засоби програмування для РКС, а поширення потреби в розробленні розподілених додатків привело до необхідності створення проміжного програмного забезпечення (middlewere), яке виконує функції інтерфейсу між розподіленою програмою і розподіленою системою і яке дозволяє автоматизувати процес його розроблення і виконання (CORBA, COM/DCOM). Гама засобів розроблення розподілених програм зараз досить значна і безперевно розширюється, включаючи сокети, віддалені процедури, віддалені об'єкти та ін.

Розподілена програма – це набір розділів, з яких складається програма і які розміщуються та виконуються на різних вузлах РКС. Модель *клієнт – сервер* є класичною для розподілених обчислень. *Сервер* – об'єкт, який надає послуги (або спільні ресурси) іншим об'єктам (обчислювальні сервери, сервери друку, файл-сервери, Web-сервери). Клієнт – це об'єкт, який звертається до серверу за послугами. Розроблення розподіленого додатку пов'язано з побудовою паралельного алгоритму розв'язання завдання, визначенням в ньому серверної і клієнтської частини, розроблення алгоритмів серверу і клієнта та організвції взаємодії сервера і клієнта з використанням необхідних бібліотек, або мовних засобів.

Розподілені алгоритми

Алгоритмами є покрокові визначення потоку обчислень і інформаційного обміну. Сьогодні алгоритмічно обумовленими в мережних розподілених

обчисленнях є три основні архітектурні моделі, які розрізняються по ступені синхронізації і роздільності.

. Модель синхронної передачі повідомлень.

. Модель асинхронної передачі повідомлень.

. Модель із асинхронним спільним використанням пам'яті.

З алгоритмічної позиції системи можуть вважатися асинхронними, якщо немає фіксованої верхньої межі часу для доставки повідомлення, або часу на обробку між кроками. Наприклад, електронний лист може бути доставлено адресатові всього за кілька секунд, але це також може зайняти кілька днів, залежно від характеру мережі. Як такі асинхронні моделі не можуть, по визначенню, забезпечити надійні часові гарантії. З іншого боку, надійність асинхронних моделей може бути підвищена іншими їх властивостями, наприклад, стійкістю повідомлень.

Крім питань синхронності, предметом розподілених алгоритмів також є мережна топологія; визначення форм і границь динамічних мереж з метою ефективної передачі даних.

Найпростіша модель для розуміння та організації інформації і потоку інформації в середовищі мережних розподілених обчислень показана на рис.1.3.

Рис.1.3 Проста модель мережних розподілених обчислень

У цій моделі вихідні дані одного вузла стають входом іншого вузла. Взагалі, інформація може передаватися в обох напрямках.

Проміжним ПЗ позначаються протоколи, фільтри, перетворювачі, брандмауери тощо. Проміжне ПЗ є центральним поняттям мережних розподілених обчислень і його можна розуміти як ПЗ, що з'єднує два і більше розділених застосувань. Як видно, проміжне ПЗ ортогональне (незалежне) стосовно мережних розподілених обчислень, що часто дає можливість використовувати проміжне ПО на одиночному вузлі і забезпечувати зв'язок між локальними додатками.

Передача повідомлень є основою всіх видів передачі даних, до яких відносяться як віддалений виклик процедури RPC (Remote Procedure Call), так і об’єкти, що виконуються. Коли вузол вирішує здійснити доступ, або спільно використовувати дані, відсутні в його пам'яті, але які перебувають у межах фізичної досяжності (тобто за межами високошвидкісної шини, чи

внутрішнього механізму передачі даних), то необхідно передавати повідомлення. Хоча існує багато різних варіантів, ця проста концепція є фундаментальною складовою частиною мережних РО.

Передача повідомлень в мережних РО може приймати багато форм. Певний тип переданих даних повинен підходити для певної операції і це ж справедливо відносно форми їхньої передачі. Реалізація широкомовного розсилання повідомлення, наприклад, багато в чому відрізняється від передачі «один до одного». На природу повідомлень, безперечно, впливають часові співвідношення. Із простої передавальної моделі виникає складний набір реалізацій, що базується на трьох загальних поглядах на повідомлення:

- перетворення даних;
- синхронність;
- маршрутизація.

Це спричиняють різні особливості схем передачі повідомлень, які широко використовуються сьогодні.

Сокети

Сокети - низькорівневий механізм, який застосовується для організації розподілених обчислень (запропоновані при створенні ОС UNIX). Сокет – своєрідний мережевий роз'єм на логічному рівні, який дозволяє передавати і приймати дані між комп'ютерами, з'єднаних мережею. Сокет дозволяє серверу обслуговувати клієнта, чекаючи на його підключення, а клієнту звертатися до сервера. Для створення сокету застосовують порт. Порт – абстрактне поняття, яке є пронумерованим сокетом на конкретному вузлі системи. Сервер «прослуховує» порт доти, поки клієнт не з'єднається з ним. Сервер може обслуговувати кількох клієнтів. При цьому оптимальний серверний додаток має бути реалізованим в вигляді кількох процесів (багатопотоковий сервер), кожен з яких відповідає за з'вязок зі своїм клієнтом. Механізм сокетів реалізовано в Java, Win32.

Віддалені процедури

Механізм виклику віддаленої процедури (механізм RPC – Remote Procedure Call) реалізує високорівневу концепцію взаємодії розділів розподіленої програми. Зовнішньо віддалена процедура нічим не відрізняється від звичайної процедури. Особливість її полягає в тому, що розділ, в якому знаходиться віддалена процедура (сервер) і розділ, де здійснюється виклик віддаленої процедури (клієнт), знаходяться в різних вузлах розподіленої системи, тому виклик та виконання віддаленої процедури пов'язано з трьома діями:

1) передавання вхідних параметрів віддаленої процедури з вузла клієнта на вузол серверу;
2) виконання віддаленої процедури на вузлі серверу (де вона знаходиться);
3) повернення результату виконання віддаленої процедури з вузла серверу на вузол клієнта за допомогою вихідних параметрів.

Таким чином аргументи віддаленої процедури передаються в мережі в вигляді повідомлень між вузлами розподіленої системи. Клієнт ініціює виконання віддаленої процедури створенням і запуском нового процесу на вузлі серверу і чекає його завершення, після чого клієнт продовжує своє виконання (відмінність від механізму рандеву в тому, що останній потребує реалізацію програми серверу в вигляді активного модуля – процесу, а RPC дозволяє використовувати для розміщення віддаленої процедури пасивні модулі, наприклад, пакети).

Взаємодія розділів за допомогою сокетів і порти у взаємодії вузлів за допомогою сокетів показані на рис.1.4 і рис.1.5 відповідно.

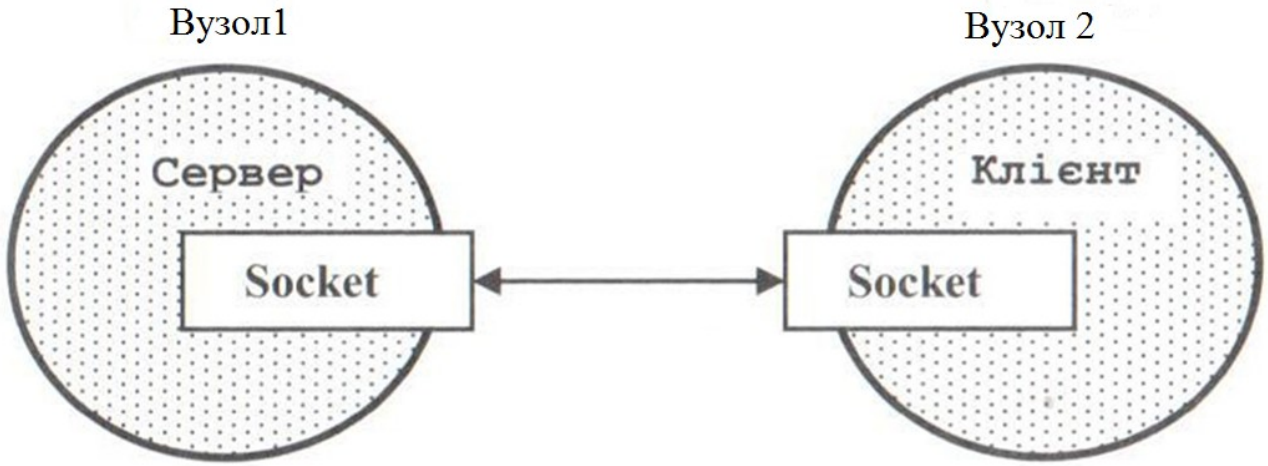


Рис.1.4. Взаємодія розділів за допомогою сокетів

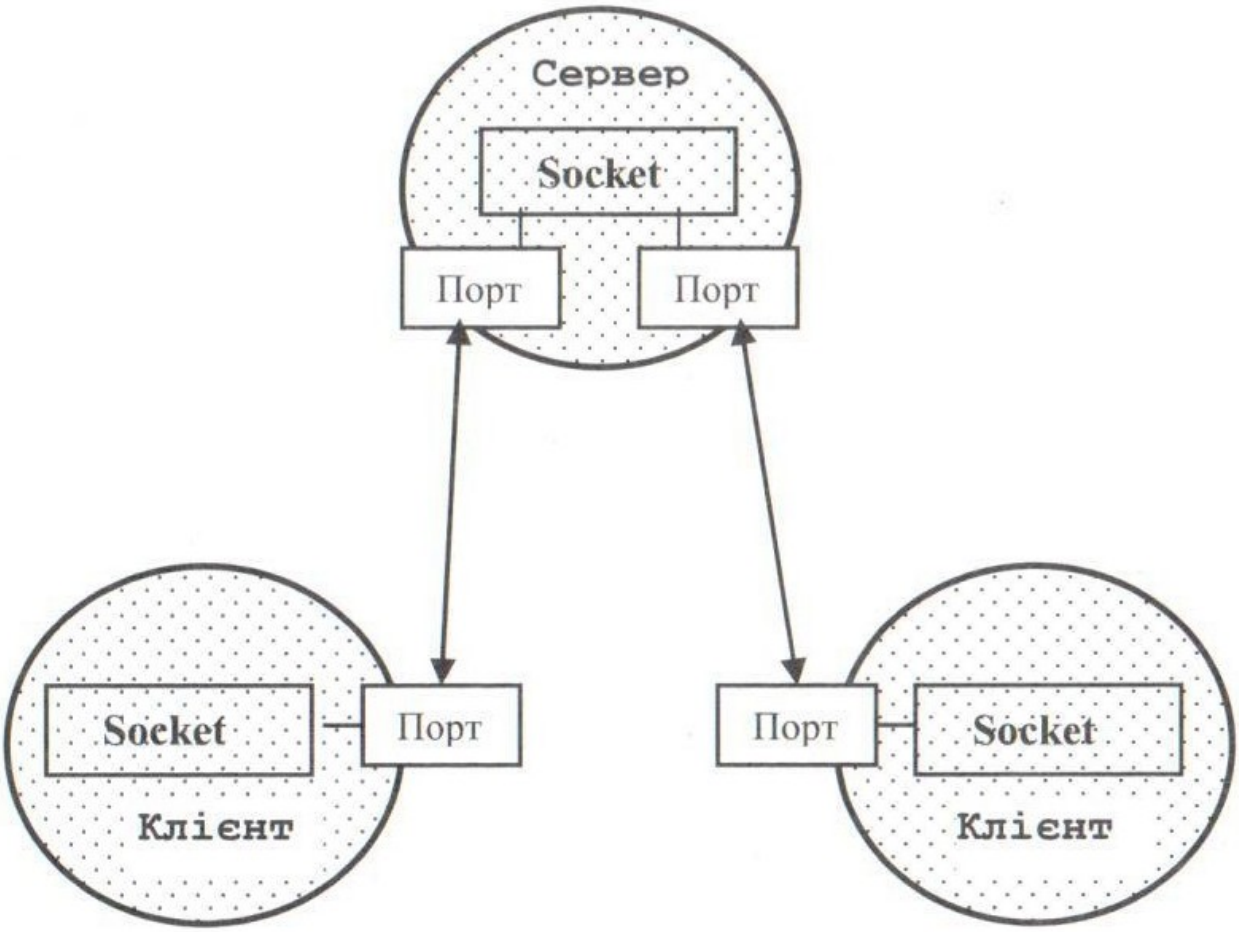


Рис.1.5. Порти у взаємодії вузлів за допомогою сокетів

Взаємодія клієнта і сервера з використанням віддаленої процедури *Sum1 (X,Y: in Вектор ; Z ; out Вектор)* зображена на рис. 1.6

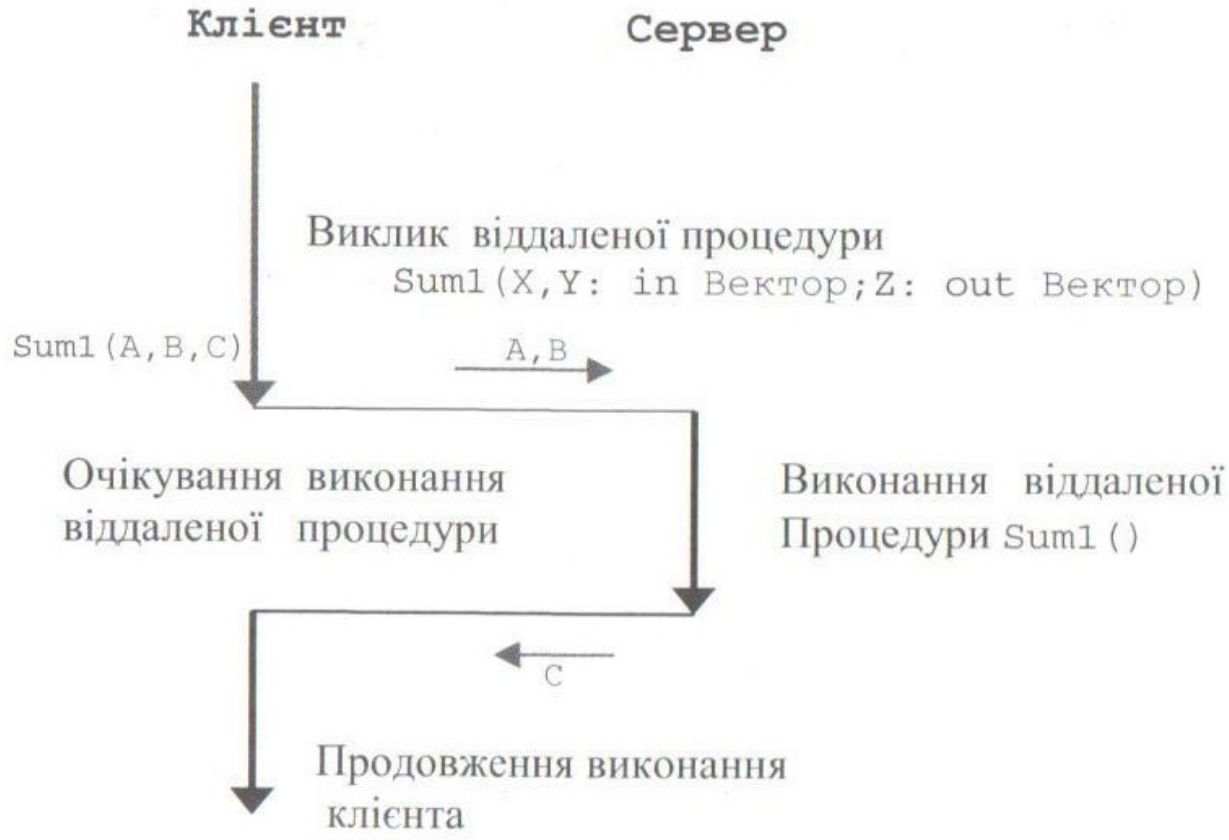


Рис.1.6.Взаємодія клієнта і сервера з використанням віддаленої процедури *Sum1*

На рис.1.7 приведена взаємодія клієнта і сервера з використанням іншої віддаленої процедури *Sum2 (X,Y: in Вектор ; Z).* Виклик віддаленої процедури *Sum2* ( ) асинхронний і зв'язаний з очікуванням клієнтом виконання віддаленої процедури та повернення її результату. При цьому клієнт передає вхідні дані віддаленої процедури *Sum2* ( ) під час її виклику і потім продовжує своє виконання. Результат виконання *Sum2* ( ) можна отримати від сервера пізніше за допомогою виклику іншої віддаленої процедури. Віддалена процедура, яка допускає асиннхронний виклик має бути описана як асинхронна і вона не може мати вихідні параметри для повернення результату.

Нижче приведена асинхронна взаємодія клієнта і сервера за допомогою механізму RPC.

Рис.1.7.Взаємодія клієнта і сервера з використанням віддаленої процедури *Sum2*

Схема взаємодії вузлів розподіленої системи за допомогою механізму RPC показана на рис.1.8.

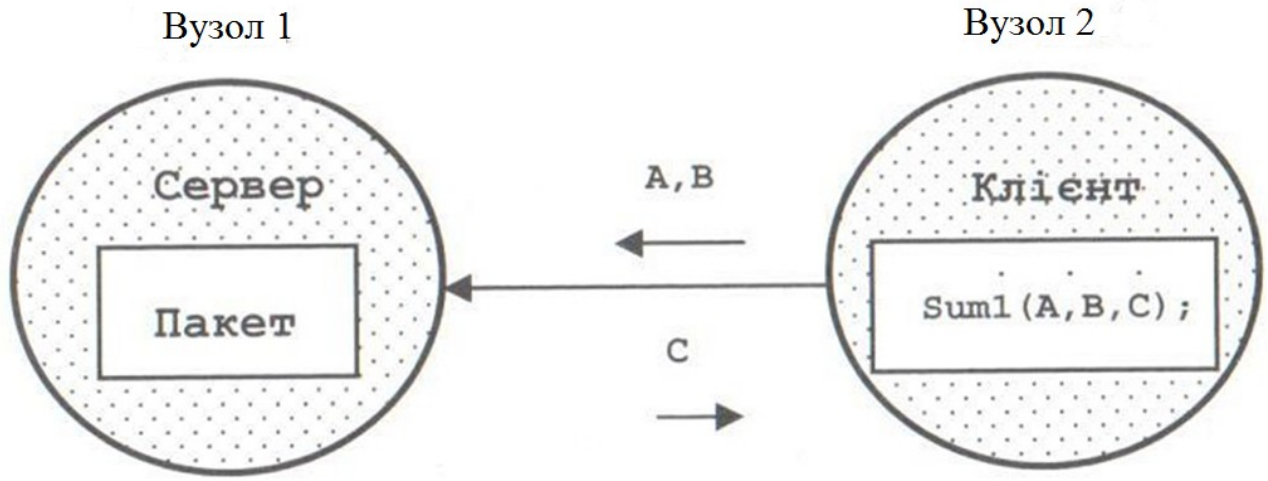


Рис.1.8 взаємодії вузлів розподіленої системи за допомогою механізму RPC

Серверний вузол (Вузел 1) містить пасивний модуль *Пакет,* в якому описана віддалена процедура *Sum1* ( ). Клієнтський вузол (Вузел 2) містить активний модуль – процедуру *клієнт*, в якому здійснюється виклик віддаленої процедури *Sum1* ( ). Механізм RPC реалізовано багатьма мовами, наприклад, Java та Ада. Мова Java забезпечує ресурси для створення розподілених додатків на моделі *клієнт – сервер* в вигляді сокетів і механізму виклику віддалених методів (RMI механізму). Механізм сокетів грунтується на парадигмах, застосованих в комп'ютерних мережах для організації взаємодії комп'ютерів і залежить від використовуваного типу протоколу обміну (TCP, UDP, ICMP та ін.). Механізм

TCP сокетів в мові Java реалізований за допомогою класів *Socket* і *ServerSocket* з пакету *Java.net.* Клас *Socket* застосовують для серверних і клієнтських частин розподіленого додатку. Він дозволяє створити *Socket*-об'єкти, за допомогою яких клієнт *Socket (String Хост, int Порт)* створює сокет, який з'єднує локальний вузел з іншим вузлом на ім'я *Хост* через порт *Порт.* Конструктор може виконати виключення *IOException* або *UnknownHostExctption.* Конструктор *Socket (InetAddress IP_Адрес, int Порт)* створює сокет, який з'єднує локальний вузел з іншим вузлом з адресою *IP_Адрес* через порт *Порт.* Сокет може отримувати пов'язану з ним адресну та портову інформацію з допомогою спеціальних методів. Метод *InetAddress getInetAddress ( )* повертає *InetAddress* об'єкт, пов'язаний з *Socket* об'єктом. Метод *int getPort ( )* повертає локальний порт, з яким з'єднаний *Socket* об'єкт. Після створення *Socket* об'єкту його можна зв'язувати з вхідним або вихідним потоком введення-виведення (*InputStream, OutputStream)* для передавання (приймання) даних з іншого вузла.

Приклад створювання сокета *Сокет1* і потоків введення-виведення *In* і *Os* для запису і читання даних через сокет:

Socket Сокет1 = nev Socket (12 . 12 . 12 . 11 , 45)

InputStream In = Сокет1 . getInputStream ( ) ;

OutputStream Os = Сокет1 . getOutputStream ( ) ;

Після завершення роботи з сокетом його потрібно закрити:

Сокет1.close ( );

Клас *ServerSocket* використовують для створення сокетів на серверній частині розподіленого додатку. Він дозволяє створювати об'єкти (серверні сокети), які застосовуються для встановлення зв'язку з клієнтом. Для цього сервер прослуховує порт, визначений в сокеті і очікує підключення клієнта. Клас має три конструктори для створення серверних сокетів.

Конструктор *ServerSocket (int Порт)* створює серверний сокет на вказаному порті *Порт* з розміром черги за замовчуванням. Конструктор *ServerSocket (int Порт, int Черга)* створює серверний сокет на вказаному порті *Порт* з максимальним розміром черги згідно параметру *Черга.* Конструктор *ServerSocket (int Порт, int Черга, InetAddress Локальна Адреса)* створює серверний сокет на вказаному порті *Порт* з максимальним розміром черги згідно параметру *Черга.* На груповому хост-вузлі параметр *Локальна Адреса* визначає IP – адресу, з якою цей сокет пов'язаний.

Клас *ServerSocket* має ключовий метод *accept ( ),* який дозволяє серверу виконувати з'єднання з клієнтом:

СКС = СС. accept ( ),

де *СКС* та *СС* – звичайний та серверний сокети, створені на серверній частині додатку. Це з'єднання ініціює клієнт при створенні свого сокету. В сервері метод *accept ( )* виконує прослуховування порту, визначеному в серверному сокеті *СС.* Якщо зв'язку з боку клієнта немає, метод *accept ( )* блокує сервер і чекає на підключення клієнта. При підключенні клієнта метод *accept ( )* отримує від клієнта службові дані (IP – адресу клієнта), які записує в звичайний сокет (*Socket* об'єкт) *СКС,* створений на боці серверу. Цей об'єкт буде використаний далі сервером для зв'язку з клієнтом безпосередньо при передаванні основних даних клієнту, тобто на серверному боці створюється два сокета: серверний *СС (ServerSocket об'єкт)* для отримання службової інформації і звичайний *СКС (Socket об'єкт)* для обміну даними.

Взаємодія вузлів за допомогою сокетів мови Java зображена на рис.1.9.

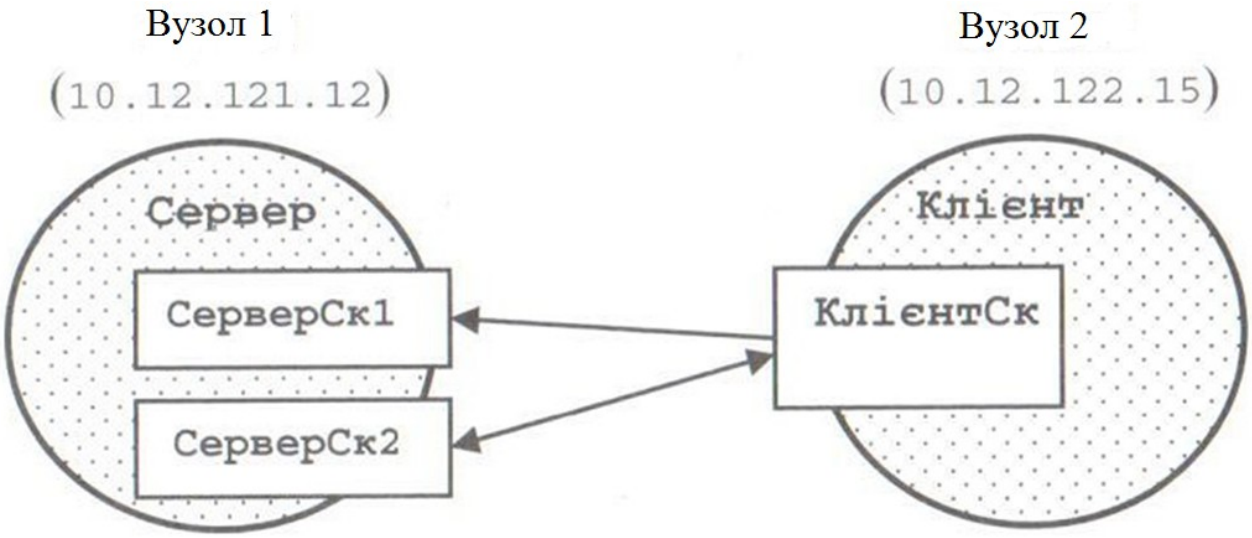


Рис.1.9. Взаємодія вузлів за допомогою сокетів мови Java

Поряд з TPC/IP сокетами Java дозволяє застосовувати для організації розподілених обчислень дейтаграмні сокети, які працюють в протоколах UDP. Дейтаграми – невеликі пакети даних, які транспортуються між вузлами розподіленої системи. Це менш надійний, але швидший механізм сокетної взаємодії. В мові Java існують класи *DatagramPacket* і *DatagramSocret.*

Реалізацію механізму виклику віддалених процедур в Java реалізовано за допомогою RMI (Remote Method Invocation) – виклик віддаленого методу. Він підтримується пакетами *java . rmi* і *java . server*. Реалізація RMI потребує створення чотирьох класів:

1) віддаленого інтерфейсу, який розширює інтерфейс *Remote* з пакету *java . rmi* і в якому визначено віддалені методи;
2) класу, який є ралізацією віддаленого інтерфейсу через розширення класу *UnicastRemoteObject;*
3) класу-серверу з методом *main ( ):*
4) класу–клієнта, який визначає взаємодію з клієнтом (методом *Naming . lookup* об'єкт отримує від служби реєстрації об'єкт сервер та інформацію про сервер і може викликати віддалені методи сервера.

Взаємодія сервера і клієнта в RMI під час виклику віддаленого методу наведена на рис.1.10.

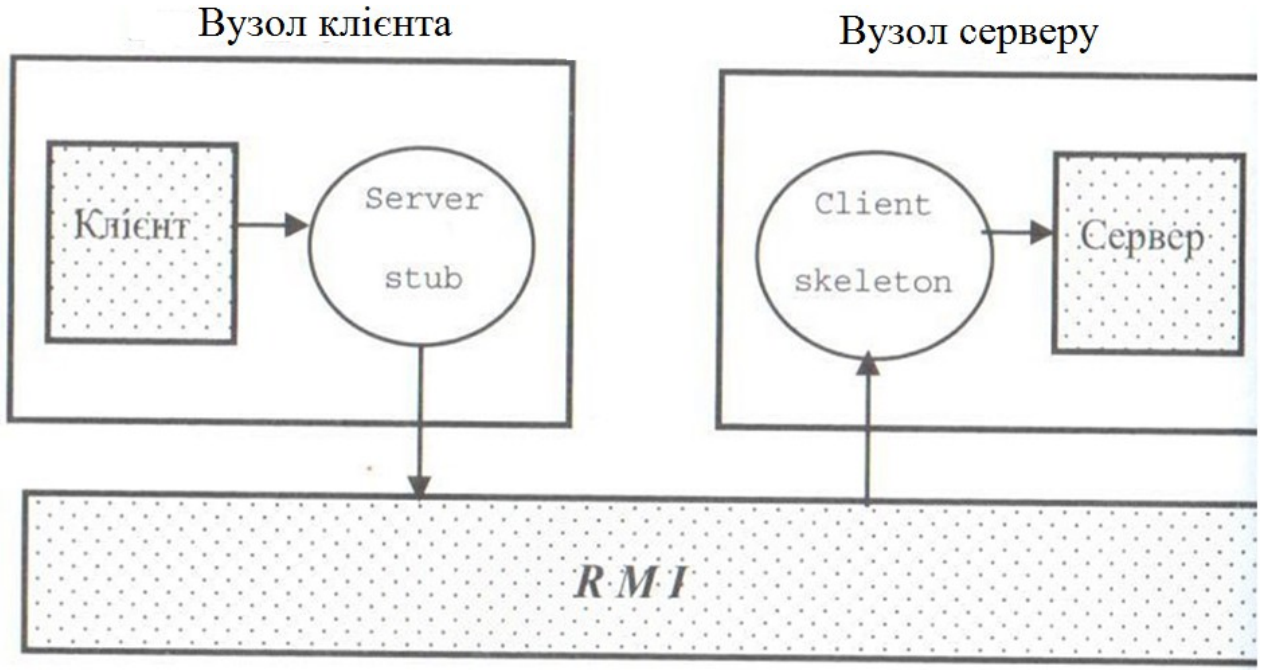


Рис.1.10.Взаємодія сервера і клієнта в RMI під час виклику віддаленого методу

Питання для самоконтролю.

1. В чому полягає головна відмінність РКС від КС з локальною пам'яттю?
2. На що націлені кластерні обчислення ?
3. Привести структуру розподіленої комп'ютерної системи.
4. Дайте визначення механізму виклику віддаленої процедури.
5. В чому полягає особливість віддаленої процедури в РКС?
6. Як клієнт ініціює виконання віддаленої процедури?
7. Що таке синхронній виклик віддаленої процедури?
8. Що таке асинхронній виклик віддаленої процедури?
9. Що таке розподілена програма?

10.Охарактеризуйте модель *клієнт – сервер.*
11. Дайте визначення механізму *сокетів.*

Література основна

1. Жуков І.А., Корочкін О.В. Праралельні та розподілені обчислення. – К.: Корнійчук, 2014, - 284с.
2. Ясько М.М. Паралельні та розподілені обчислення. – Д.;РВВ ДНУ, 2010. - 111 с.

Література допоміжна

3. Малышкин ВЭ Корнеев ВД Параллельное программирование мультикомп'ютеров. – НГТУ, 2006, – 296с.
4. Таненбаум Э., Ван-Стеен М. Распределенные системы. Принципы и парадигмы. Спб.: Питер, 2003. 877 с
5. Lamport L. Distributin. URL: . http://research.microsoft.com/enus/um/people/lamport/pubs/distributed-system.txt

1.2 **Обчислювальний Кластер**

РКС — розподілена інфраструктура (як правило, географічно) , об'єднуюча багато ресурсів різних типів, доступ до яких користувач може отримати з будь-якого місця, незалежно від місця їх розташування. Основою РКС являються обчислювальні ресурси, представлені в вигляді кластерів ЕОМ, суперЕОМ і різних варіантів їх об'єднання. Можна сказати, що кластер – це група обчислювальних вузлів, об'єднаних швидкісною мережею, яка являє собою для користувача єдиний апаратний ресурс, а кластерні обчислення – це особлива технологія високопродуктивних обчислень, яка зародилася разом з розвитком комунікаційних засобів і стала прекрасною альтернативою використанню надпотужних комп'ютерів. Коли з'явилися доступні канали зв'язку з високою пропускною здатністю, з'явилась концепція віртуального супернкомп'ютера, де масштабна задача виконується сумісно в єдиній мережі кластером звичайних комп'ютерів. До розряду кластерних обчислень можна віднести всякі паралельні обчислення, де всі комп'ютери системи використовуються як один уніфікований ресурс. Існує кілька типів кластерів: кластери високої доступності, MPP-кластери, високопродуктивні кластери, кластери розподіленої загрузки. Grid являється різновидністю кластерних систем. Вона має свої особливості, обумовлені масштабом. Обчислювальні вузли Грід розташовані на значній відстані один від одного і доступність того

чи іншого із них не гарантована. Це накладає певні вимоги до керування ресурсами. Структура Грід – це віртуальна організація, влаштована над простором реальних комп'ютерів, мереж, адміністративних зон. Задача віртуальної організації – створити середовище, де частини одного додатку, виконуємого на різних комп'ютерах, будуть взаємодіяти між собою і до системи можна динамічно підключати будь-який новий ресурс. При цьому їх комунікації не повинні залежати від середовища виконання. Незалежно від того, де відбувається обчислення: в рамках одного комп'ютера, в локальній чи глобальній мережі, об'єднуючій багато організацій – це повинно залишатись прозорим для додатку.

Обчислювальний кластер. [2]

Це система, зазвичай, складається з одного серверного вузла і одного або більше клієнтських вузлів, з'єднаних за допомогою Ethernet, або деякої іншої мережі. Обчислювальний кластер, як правило, працює під управлінням однієї з версій ОС Linux – багатокористувацької операційної системи і обчислювальний кластер – це технологія кластерізації комп'ютерів, які працюють під управлінням ОС Linux на різновид паралельного віртуального суперкомп'ютеру. Ключем до поняття кластерних обчислень є ідея єдиного образу системи. У кластерах передбачена плавна зміна розмірів і подолання відмов: коли один підвузол у кластері відмовляє, то інший готовий зайняти його місце так, щоб додатки (включаючи і його користувачів) практично цього не помітили.

Кластер складається з окремих машин (вузлів) і об'єднуючої їх мережі (комутатора**)**. Крім ОС, необхідно встановити та налаштувати мережеві драйвери, компілятори, програмне забезпечення для підтримки паралельного програмування і розподілу обчислювального навантаження.

Вузли кластеру: підходящим вибором в даний момент є системи на базі процесорів Intel (наприклад, існуючий кластер НТУУ «КПІ» - один із найпотужніших в Україні - містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра) . Варто встановити на кожен вузол 2-4Gb оперативної пам'яті. Одну з машин виділити в якості центральної (консоль кластеру), куди встановити великий жорсткий диск, можливо, більш потужний процесор і більше пам'яті, ніж на робочі вузли. Робити консоль кластеру більш потужною машиною має сенс, якщо необхідно мати на цьому комп'ютері крім інтерфейсу командного рядка більш зручне операційне оточення, наприклад віконний менеджер (KDE, Gnome), офісні програми, програми візуалізації даних і т.п.

Будова кластеру

Розгортання кластеру - завдання не надто складне. Для цього підійде будь-який дистрибутив. Який саме з дистрибутивів Linux ставити в якості базової ОС - не має значення. Ubuntu, Mandriva, Alt Linux, Red Hat, SuSE. Вибір залежить тільки від уподобань користувача.

Отже, щоб розвернути кластер, використовуючи дистрибутив загального призначення, слід виконувати наступне:

1. Встановити операційну систему на комп'ютер, який буде виступати в ролі консолі кластеру. Тобто на цьому комп'ютері будуть компілюватися і запускатися паралельні програми. Іншими словами, за цим комп'ютером буде сидіти людина, запускати програми і дивитися, що вийшло.
2. Після інсталяції базової ОС на консолі кластеру необхідно встановити необхідні компілятори (фортран, С) і всі необхідні бібліотеки, desktop environment (GNOME або KDE), текстові редактори і т. д..
3. Встановити з репозитарію або з вихідного пакет MPICH, який являється спеціальною реалізацією MPI для Globus Toolkit.
4. Описати в /etc/hosts майбутні вузли кластеру, в тім рахунку консоль кластеру.
5. Встановити NFS і розшарувати для всіх вузлів кластеру якусь директорію, в якій будуть розміщуватися виконавчі модулі паралельних програм та файли даних, якими ці програми будуть користуватися в процесі своєї роботи.
6. Встановити на консолі кластеру ssh-клієнт (обов'язково) та ssh-сервер (опціоно, якщо буде надаватися доступ до консолі кластеру по мережі).
7. На всіх вузлах кластеру встановити операційну систему, бібліотеки, необхідні для виконання користувацьких паралельних програм, встановити MPICH, NFS-client, ssh-server. Якщо вузли кластеру в цілях економії ресурсів навантажуються в runlevel 3, то ставити GNOME або KDE не треба. Максимум - поставити ряд бібліотек, якщо вони потрібні для користувача.
8. Описати в /etc/hosts всіх вузлів кластеру майбутні вузли вашого кластеру, в тім рахунку і консоль кластеру.
9. На всіх вузлах кластеру необхідно автоматично при завантаженні монтувати розшарений ресурс. Причому, шлях до цього ресурсу має бути однаковий як на консолі кластеру, так і на його вузлах. Наприклад, якщо на консолі кластеру дати доступ каталогу /home/mpiuser/data, то на вузлах кластеру цей ресурс також має бути змонтований в /home/mpiuser/data.
10. На всіх вузлах кластеру забезпечити безпарольний доступу по ssh для консолі кластеру.

Оскільки від мережі залежить ефективність роботи кластеру, то бажано зробити наступне. Необхідно, щоб функціонування мережевої файлової системи NFS не заважало обміну даними, який здійснюють між собою частини паралельної програми, які працюють в різних вузлах. Щоб це здійснити, необхідно фізично розділити мережу на два сегменти. В одному сегменті буде працювати NFS, в іншому - відбуватиметься обмін даними між частинами програми.

Організація мережі обчислювального кластеру

Мережа - це модульна і адаптуюча комутаційна система, яку можна налаштувати відповідно до самих різних вимог. Її модульність полегшує додавання нових компонентів, або переміщення існуючих, а адаптивність спрощує внесення змін і удосконалень.

Мережа кластеру нічим принципово не відрізняється від мережі робочих станцій, тому в найпростішому випадку для побудови кластеру необхідні звичайні мережеві карти та хаби / комутатори, які використовувалися б при

облаштуванні якогось комп'ютерного класу. Однак, у випадку кластеру є одна особливість. Мережа кластеру в першу чергу призначена не для зв'язку машин, а для зв'язку обчислювальних процесів. Тому чим вищою буде пропускна здатність мережі, тим швидше будуть виконуватись паралельні завдання, запущені на кластері, отже робочі характеристики мережі набувають першочергового значення.

Для побудови обчислювальних кластерів використовують найрізноманітніше мережеве обладнання. Але слід пам’ятати, що пропускна здатність мережі, яка зв'язує вузли кластеру, у багатьох випадках виявляється вирішальною для продуктивності кластеру.

Використовуване мережеве обладнання характеризують зазвичай двома параметрами:

Латентність - це середній час між викликом функції передачі даних і самою передачею. Час витрачається на адресацію інформації, спрацювання проміжних мережних пристроїв, інші накладні витрати, що виникають при передачі даних.

Фактично пропускна здатність і латентність не тільки характеризують кластер, але й обмежують клас задач, які можуть ефективно вирішуватися на ньому. Так, якщо завдання вимагає частої передачі даних, кластер, який використовує мережеве обладнання з великою латентністю (наприклад, Gigabit Ethernet), буде велику частину часу витрачати навіть не на передачу даних між процесами, а на встановлення зв'язку, в той час як вузли будуть простоювати, і ми не отримаємо значного збільшення продуктивності. Втім, якщо пересилаються великі обсяги даних, вплив періоду латентності на ефективність кластеру може знижуватися за рахунок того, що сама передача вимагатиме досить великого часу, може бути навіть у рази більшою за період латентності.

Мережеві карти

В якості мережевих адаптерів можна використовувати будь-які доступні карти, які підтримують роботу в стандартах 100BaseTx і GigabitEthernet.

Комутатори

Іншим важливим елементом мережі кластеру є пристрої комутації мережевих каналів. Комутатори та інші елементи мережевої структури використовуються для забезпечення комунікацій між процесорами, для підтримки паралельного програмування і різних функцій керування. Для паралельного програмування організації взаємодії між процесами (Inter Process Communication, IPC) широко використовується комутатор Myrinet-2000 - швидкий, добре масштабований широкосмуговий пристрій.

Паралельна віртуальна машина (PVM)

Основою обчислювального середовища кластеру є паралельна віртуальна машина (PVM). PVM - це пакет програм, який дозволяє використовувати пов'язаний в локальну мережу набір різнорідних комп'ютерів, як один великий паралельний комп'ютер. Таким чином, проблема великих обчислень може бути досить ефективно вирішена за рахунок використання сукупної потужності та пам'яті великої кількості комп'ютерів. Пакет програм PVM легко переноситься на будь-яку платформу. Вихідні тексти, вільно

розповсюджувані netlib, були скомпільовані на комп'ютерах, починаючи від laptop і до CRAY.

У загальному випадку кількість завдань може перевершувати число процесорів, включених в PVM. Крім того, до складу PVM можна включати досить різнорідні обчислювальні машини, несумісні з систем команд і форматів даних. Отже, паралельною віртуальною машиною може стати як окремо взятий ПК, так і локальна мережа, що включає в себе суперкомп'ютери з паралельною архітектурою, універсальні ЕОМ, графічні робочі станції і все ті ж малопотужні ПК. Важливо лише, щоб процеси, які включаються до PVM-обчислювальних засобів були інформацією у використовуваному програмному забезпеченні PVM. Завдяки цьому користувач може вважати, що він спілкується з однією обчислювальною машиною, в якій можливе паралельне виконання багатьох завдань.

PVM дозволяє користувачам використовувати існуючі апаратні засоби, для вирішення більш складних завдань при мінімальній додаткової вартості. Головна мета використання PVM – це підвищення швидкості обчислень за рахунок їх паралельного виконання. Функціонування PVM засноване на механізмах обміну інформацією між завданнями, виконуваними в її середовищі. У цьому відношенні найбільш зручно реалізовувати PVM в рамках багатопроцесорних обчислювальних комплексів, виділивши віртуальній машині кілька процесорів і загальну або індивідуальну (залежно від умов) оперативну пам’ять. Використання PVM доспускається як на багатопроцесорних комп'ютерах (SMP), так і на обчислювальних комплексах, побудованих за допомогою кластерної технології. При використанні PVM, як правило, значно спрощуються проблеми швидкого інформаційного обміну між завданнями, а також проблеми узгодження форматів представлення даних між завданнями, виконуваними на різних процесорах.

Ефективне програмування для PVM починається з того, що алгоритм обчислень слід адаптувати до складу PVM і до її характеристик. Це творче завдання, яке в багатьох випадках повинне вирішуватися програмістом. Крім завдання розпаралелювання обчислень з необхідністю виникає і завдання керування обчислювальним процесом, координації дій і завдань учасників цього процесу. Іноді для керування потрібно створювати спеціальну задачу, яка сама не беручи участь в обчисленнях, забезпечує узгоджену роботу решти завдань.

При паралельних обчисленнях необхідно програмувати спеціальні дії з координації роботи завдань, такі як процеси запуску задач на процесорах кластеру, управління обміном даних між завданнями та ін. Найбільш простий спосіб організації паралельного процесу виглядає наступним чином. Спочатку запускається одне завдання (master), яке в колективі завдань буде відображати функції координатора робіт. Це завдання виробляє деякі підготовчі дії, наприклад, ініціалізація початкових умов, після чого запускає інші завдання (slaves), яким може відповідати або той же виконуваний файл, або різні виконувані файли. Такий варіант організації паралельних обчислень переважно використовується при ускладненні логіки керування обчислювальним процесом, а також коли алгоритми, реалізовані в різних завданнях, істотно

розрізняються, або є великий обсяг операцій (наприклад, введення - виведення), які обслуговують обчислювальний процес в цілому.

Отже резюмуючи розглянуті питання, кластери – це багатопроцесорні ЕОМ, побудовані на елементах серійного виробництва. Це відноситься не тільки до апаратури, але і до програмного забезпечення. Основою операціної системои для кластерів являеться ОС Linux. В свою чергу кластери можна розділити на два помітно різні по спроможності класи:

• кластери типу Beowulf, які будуються на базі звичайної локальнї мережі ПЕОМ. Щоб отримати такий кластер, до мережі ПЕОМ потрібно добавити тільки системне програмне забезпечення, яке безоплатно розповсюджується в інтернеті;

• монолітні кластери, все обладнання яких компактно розміщене в спеціалізованих шкафах. Це дуже швидкі машини, кількість процесорів в яких може досягати сотень та тисяч. Процесори в монолітних кластерах не можуть викороистовуватися в персональному режимі.

Проект Beowulf розпочався в 1994р в центрі NASA: був зібраний 16-процесорний кластер на процесорах Intel 486DX4/100 Мгц. На кожному вузлі було встановлено по 16 Мбайт ОЗП і по 3 мережевих карти для звичайної мережі Ethernet. Були розроблені спеціальні драйвери для роботи в такій конфігурації, які розподіляли трафік між доступними мережевими картами. Результат експлуатації кластеру виявився вдалим і тепер переважна більшість суперкомп'ютерних систем по списку Top500 відносяться до кластерних систем. Обчичлювальний кластер – це мультикомп'ютер, який складається з певної кількості окремих комп'ютерів (вузлів), зв'язаних між собою єдиною комунікаційною системою: кожний вузол має свою локальну оперативну пам'ять, загальної фізичної оперативної пам'яті для вузлів не існує. Вузли кластера – це повноцінні комп'ютери. Якщо вузли кластеру побудовані на мультипроцесорах (мультипроцесорні комп'ютери з загальною пам'яттю), то такий кластер називається SMP-кластером. Якщо всі вузли кластеру мають однакову архітектуру і продуктивність, то такий кластер є однорідним, ящо ні – то неоднорідним. Перші Beowulf-кластери використовували 10-мегабітну мережу Ethernet. Сьогодні використовується мережа Gigabit Ethernet на базі комутаторів. Існують і інші комунікаційні технології. Найпоширинішим є використання однорідних кластерів. Один сервер виконує функції керуючого вузла кластеру. На цьому комп'ютері встановлене програмне забезпечення, яке активує обчислювальні вузли при завантаженні операційної системи і керує запуском програм на кластері. Завдання користувачів виконуються на обчислювальних вузлах, де вони розподіляються таким чином, що на кожний процесор розподіляється не більше одного обчислювального процесу. Користувачі мають термінальний доступ до головної машини кластеру (цей сервер забезпечує зв'язок кластеру із завнішньою мережею через корпоративну локальну мережу, або через Інтернет). Найчастіше для термінального доступу використовується протокол SSH. На цьому ж сервері розміщені домашні каталоги користувачів. У деяких конфігураціях для доступу користувачів і розміщення домашніх каталогів виділяється окремий сервер – так званий сервер доступу. Доступ на вузли

кластеру закритий для користувачів, або можливий, наприклад, для ручного управління компіляцією завдання.

Програми на кластері виконуються в пакетному режимі: користувач не має безпосередньої взаємодії з програмою, прорама не може очікувати введення даних із клавіатури і виводити їх безпосередньо на екран. Більше того, програма користувача може працювати тоді, коли користувач не підключений до кластеру.

Як вже вказувалося обчислювальний кластер, як правило, працює під управлінням однієї з версій ОС Linux – багатокористувацької ОС. Всі вузли кластеру мають доступ до загальної файлової системи, яка розміщується на файл-сервері. Файл може бути створений, наприклад, на головній машині, або на обчислювальному вузлі, а потім прочитаний під тим же ім'ям на іншому вузлі. Запис в один файл одночасно з різних вузлів неможливий, але запис в різні файли припустимий. Крім загальної файлової системи можуть використовуватися локальні диски на обчислювальних вузлах кластеру. Їх використовують програми для зберігання тимчасових файлів. Після закінчення роботи програми ці файли видаляються.

Задіяти обчислювальні потужності кластеру можна різними способами :
- виконувати певну кількість однопроцесорних задач, якщо потрібно провести певну кількість незалежних обчислювальних експериментів з різними вхідними даними, а всі дані розміщуються в оперативній пам'яті, доступній одному процесу;
- використовувати готові пакети програм, в яких реалізовані паралельні обчислення. Для деяких задач доступні безкоштовні паралельні програми, які можна використовувати на кластері;
- здійснювати у власних програмах виклик додатків з стандартних бібліотек для паралельних обчислень;
- створювати власні програми.

Далі розглянемо питання створення навчального (наприклад, кафедрального) кластеру. Створення такої навчальної кафедральної системи може забезпечити практичне освоєння паралельних обчислень в комп'ютерній мережі студентами, наприклад, кафедри ПЗКС ФПМ.

У сучасному світі для освоєння паралельних обчислень широко використовуються кластерні системи, які стали робочим інструментом в наукових дослідженнях та інженерних розрахунках. Їх по праву називають основною технологічною зброєю XXI ст. Однако, справжній класичний кластер це далеко не просте створіння і його побудова під силу тільки потужним структурам. Існуючий кластер НТУУ «КПІ ім.І.Сікорського» є одним із найпотужніших в Україні (містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра) і не взмозі забезпечити потреби повсякденного широкого використання всього існуючого студентського загалу НТУУ. Цей кластер можна розглядати як серйозний інструмент для фахівців з відповідним практичним досвідом, які набили свої «гулі» на простіших інструментах.

Створення навчальної кафедральної системи для освоєння паралельних обчислень в комп'ютерній мережі для потреб окремої кафедри чи факультету для широкого залучення їх студентського загалу є актуальною задачею.

На ФПМ є 12 комп'ютерних класів із загальною кількістю комп'ютерів 119. На кафедрі програмного забезпечення комп'ютерних систем факультету прикладної математики є три комп'ютерних класи, які налічують 28 сучасних комп'ютерів, об'єднаних в локальну мережу, є вихід до Internet. На їх основі і можна побудувати обчислювальну навчальну кафедральну систему для освоєння паралельних обчислень для використання в навчальному процесі. Така обчислювальна система буде повільною із-за затримок в мережі, її ефективність може поступатися реалізації паралельних обчислень навіть в одному хорошому комп'ютері, але для навчальних цілей це не так важливо. Важливішим є можливість здобуття студентами відповідного практичного досвіду без зайвого остраху зробити щось не так.

Практичне створення навчальної кафедральної системи.

Таку навчальну систему можно створити в невеликій організації чи вузівській кафедрі. Цей варіант позволяє навчитися: створювати мережу , інсталювати системне ПО (складна робота) та виконувати паралельні обчислення.

Для того, щоб побудувати паралельну систему на базі комп'ютерної мережі, потрібно:

1. Побудувати мережу комп'ютерів (в нас вона вже є).
2. Інсталювати системне програмне забезпечення.
3. Створити паралельну програму, виконати налаштування системи і програми.
4. Провести обчислення та оцінити їх ефективність.

Нижче представлений процес інсталяції мобільного варіанту з розрахунку, що комп'ютерна мережа з апаратурою та програмним забезпеченням уже створена:

1. На сайті www.globus.org по адресу http://www.globus.org/toolkit/downloads/4.0/ представлений перелік і адреси операційних систем та інших програмних комплексів, які можна використовувати для інсталяції одного із варіантів Грід.
2. За адресою http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html знаходиться опис порядку і всіх деталей інсталяції по варіанту quickstart (швидкий старт).

   Для виконання цього пункту частину програм можна встановити попередньо (пункт 1), а частину – в процесі виконання пункту 2. Але при виконанні пункту 2 всякий раз потрібно переконуватись чи встановлене необхідне програмне забезпечення.

3. Після завершення інсталяції Грід необхідно поверх нього встановити пакет MPICH-G2, який являється спеціальню реалізацією MPI для Globus Toolkit. Таким засобом являється пакет MPICH-G2 (www.globus.org/grid_software/computation/mpich-g2.php) - це спеціальна реалізація MPI для Globus Toolkit. 4. Накінець, потрібно запустити на Грід зі встановленим MPICH-G2 тест для вимірювання продуктивності паралельних систем HPL, призначених для рішення систем лінійних алгебраїчних рівнянь, який є розвитком тесту LINPACK. Інформація по тесту HPL разміщена за адресою http://www.netlib.org/benchmark/hpl/index.html .

MHICH (MPI Chameleon) – вільно розповсюджуєма реалізація MPI. Пакет доступний в вихідних кодах, підтримується в UNIX, Mac, Windows.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення обчислювального кластеру
2. Перерахуйте особливості програмування обчислювального кластеру
3. Що таке координація розрізнених ресурсів?
4. На що націлені Грід- обчислення ?
5. Назвати обчислювальні ресурси Грід
6. Охарактеризуйте Обчислювальний комп'ютерний кластер?
7. Наведіть будову КК
8. Охарактеризуйте мережу кластеру
9. Як можна створити кафедральний кластер?
10. Що таке Латентність?
11. Що таке PVM?
12. Охарактеризуйте типові вузли кластеру
13. Що Ви знаєте про кластер НТУУ «КПІ ім. Ігоря Сікорського»?

Література основна

1.Петренко А.И., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.
2.Пецко В.І., Моца О.В., Грід-системи та технології хмарних обчислень: методичний посібник. – Ужгород, ДНВЗ УНУ, 2016. – 50 с.
3.Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.

Література допоміжна

4 . Воеводин Вл.В., Жмутай С.А. «Вычислительное дело и кластерные системы», -М.: МГУ, 2007. -150с
5. М.Кузьминский. Кластеры на базе ОС Linux. Ж. "Computer World",№5,1998

1.3 **Робота на обчислювальному кластері НТУУ «КПІ ім. Ігоря Сікорського»** [1]

Існуючий кластер НТУУ «КПІ» - один із найпотужніших в Україні - містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра. Багато це, чи мало? На це можу сказати тільки те, що вже в 2009р суперкомп'ютер Jaguar - Cray XT5 включав в себе 224 162 ядра. Навожу це без сарказму, а для інформації, що справжній бум обчислювальної техніки розпочався в 1965р зі створення IBM-360 в США і що в той же час у нас в Україні і конкретно в Києві була створена наша ЕОМ Дніпро-2 аналогічного класу, яка нічим не поступалася американській (64-розрядна машина з 128-розрядною "плаваючою" арифметикою і аналогічною оперативною і зовнішньою пам'яттю). Ми перестали її (Дніпро-2) випускати, після чого кинулися створювати український клон IBM-360. Самі ми до цього би не додумалися. Виконувалися продумані і вдалі поради. І аналогічні дії ми продовжуємо здійснювати до сьогоднішнього дня.

Кластерна обчислювальна система Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» працює під управлінням операційної системи *Linux*. В *Linux* стандартним засобом віддаленого доступу є протокол *Secure Shell* (скорочено *SSH*). Віддалений доступ можливий як через Інтернет, так і з мережі НТУУ «КПІ ім. Ігоря Сікорського». Найбільш популярні програми-клієнти, які дозволяють встановлювати зв'язок за протоколом *SSH* - це утиліти *ssh* в *Linux* та *PuTTY* для *Windows*.

Розглянемо роботу з програмою *PuTTY*, яку можна завантажити з сайту: http://www.putty.org/. Для запуску задач на кластерній системі необхідно мати на ній акаунт. Акаунт створюється адміністратором по заявці. Вам мають бути повідомлені наступні дані:

- логін вашого акаунта (далі *user*);
- пароль вашого акаунта (далі *pass*);
- IP-адресу кластеру або його символьне ім'я (далі *host*);
- порт, на який встановлюється *SSH*-з'єднання (далі *port*).
-

1. Віддалений доступ до командного рядка

1.1 Віконний варіант *PuTTY*

Після інсталяції програми *PuTTY* можна користуватись як віконним варіантом програми, так і консольним. Обидва варіанти представлені виконуваним файлом *putty.exe*. Запуск цього файлу без параметрів відкриває головне вікно програми: *Пуск → Програми → PuTTY → PuTTY*).

Віконна версія програми має перевагу в тому, що вона дозволяє зберігати «сесію» - адресу, порт серверу і тип підключення. Для створення сесії необхідно у головному вікні (рис.3.1) ввести адресу серверу *host* в полі «*Host Name*», порт *port* в полі «*Port*» і обрати тип з'єднання *SSH*. В полі «*Saved Sessions*» необхідно ввести назву сесії (довільне ім'я) і натиснути кнопку «*Save*». Введене символьне ім'я з'явиться в списку нижче і відтепер завжди можна завантажити збережені параметри, обравши зі списку відповідне символьне ім'я і натиснувши кнопку «*Load*». Після завантаження параметрів для встановлення з'єднання з сервером необхідно натиснути кнопку «*Open*». Після цього має відкритись вікно консолі (рис.3.2).

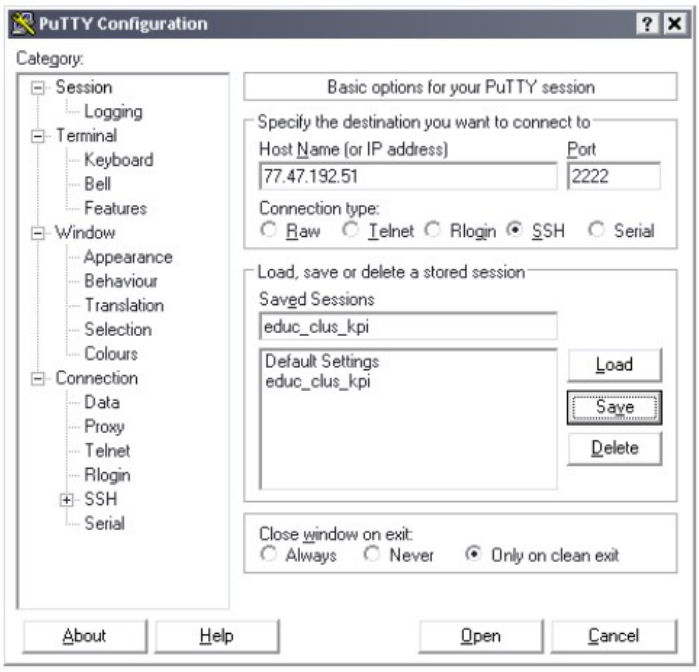


Рис.3. 1. Головне вікно *PuTTY.*

Рис.3. 2. Вікно консолі.

Якщо консоль не з'явилась, а натомість програма повідомила про помилку, то це означає, що або не працює мережа, або не працює сервер, або ви неправильно ввели надані вам дані. В даній консолі необхідно ввести ваш *user*, а потім *pass* (введені символи паролю не відображаються на екрані з метою забезпечення безпеки). Після цього ви отримаєте доступ до командного рядка, а поточним каталогом стане домашній каталог користувача.

1.2 Консольний варіант *PuTTY*

Розглянемо запуск консольного варіанту програми *PuTTY*: вибираємо *Пуск* → *Виконати*, набираємо в полі «*cmd*» і натискаємо клавішу *Enter*. Після цього з'явиться системна консоль (рис.3.3), звідки можна запустити програму.

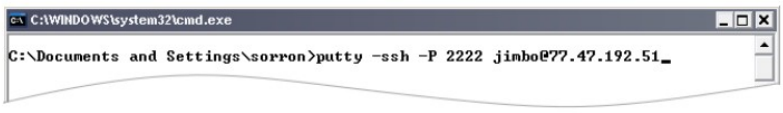


Рис.3.3. Запуск *PuTTY* з командного рядка *Windows*.

Для встановлення з'єднання необхідно ввести командний рядок у наступному форматі:

C:\>putty -ssh -P port user@host

Після цього натискаємо *Enter* і з'являється вікно віддаленої консолі обчислювального вузла, де необхідно ввести пароль. Після вводу коректного паролю відбувається вхід у систему і перехід до домашньої директорії користувача:

login as: user
user@77.47.192.51’s password:
Last login:
user@n001$ pwd
/home/user
user@n001$
Необхідно зауважити, що таким чином відкривається доступ до першого (головного) вузла кластеру. Для того, щоб перейти на інший вузол необхідно дати команду, в якій <*nodenumber*> – номер необхідного вузла.
user@n001$ ssh n<nodenumber>
Приклад переходу на вузол *n002*:
user@n001$ ssh n002
user@n002’s password:
Last login:
user@n002$

2. Робота з файлами на кластері

Для повноцінної роботи з кластером необхідна також можливість завантажувати файли на кластер і з кластера. Для цього разом з програмою *PuTTY* встановлюється утиліта *pscp.exe* (*PuTTY secure copy client*).

Зверніть увагу! Програма *pscp.exe* працює тільки в консолі *Windows*. Необхідно виконати команду: *Пуск* → *Виконати*, набрати в полі «*cmd*» і натиснути клавішу *Enter*. З’явиться системна консоль, звідки можна запускати програму *pscp.exe*.

2.1 Перегляд списку файлів

Для виведення списку файлів в заданій директорії на обчислювальному вузлі призначена команда консолі *Windows*:
C:\>pscp -P 2222 -ls user@host:directory
Після вводу кожної команди відбувається запит паролю користувача.

Приклад виконання команди:
C:\>pscp -P 2222 -ls user@77.47.192.51:/home/user
user@77.47.192.51’s password:
Listing directory /home/user
drwx------ 4 user user 4096 Apr 21 20:14 .
drwxr-xr-x 83 root root 4096 Apr 26 13:22 ..
-rw------- 1 user user 432 Apr 27 06:46 .bash_history
-rw-r--r-- 1 user user 33 Apr 20 11:54 .bash_logout
-rw-r--r-- 1 user user 176 Apr 20 11:54 .bash_profile
-rw-r--r-- 1 user user 124 Apr 20 11:54 .bashrc
drwx------ 2 user user 4096 Apr 21 19:18 .ssh
-rw------- 1 user user 704 Apr 21 20:14 .viminfo
drwxr-xr-x 2 user user 4096 Apr 21 20:14 gmptest
C:\>

2.2 Копіювання файлів на кластер

Для копіювання одного файлу на обчислювальний вузол введіть команду в консолі *Windows*:

C:\>pscp -P port localfile user@host:file
В даній команді *localfile* – ім'я файлу на локальному комп'ютері, *file* – ім'я файлу на обчислювальному вузлі. Приклад виконання команди:
C:\>pscp -P 2222 C:\foo.txt user@77.47.192.51:/home/user/foo.txt
user@77.47.192.51's password:
foo.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\>
Перевіримо наявність файлу на обчислювальному вузлі командою *ls*:
user@n001$ ls
foo.txt gmptest
user@n001$
Також на кластер можна скопіювати каталог і всі його підкаталоги. Для цього слід додати параметр *-r*:
C:\>pscp -P port -r localdirectory user@host:directory
В даній команді *localdirectory* – ім'я директорії на локальному комп'ютері, яка буде скопійована, *directory* – ім'я директорії на сервері, в яку буде скопійовано локальну директорію. Приклад:
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:23 <DIR> innerdir
1 файлов 46 байт
3 папок 1 773 776 896 байт свободно
C:\dir>cd innerdir
C:\dir\innerdir>dir
Содержимое папки C:\dir\innerdir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo2.txt
1 файлов 46 байт
2 папок 1 773 776 896 байт свободно
C:\dir\innerdir>pscp -P 2222 -r C:\dir user@77.47.192.51:/home/user/
user@77.47.192.51's password:
foo1.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
foo2.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir\innerdir>
Перевіримо наявність каталогу на обчислювальному вузлі командою *ls:*
user@n001:~$ ls
dir foo.txt gmptest
user@n001:~/dir$ cd dir
user@n001$ ls
foo1.txt innerdir
user@n001:~/innerdir$ cd innerdir
user@n001:~/innerdir$ ls

foo2.txt
user@n001:~/innerdir$

2.3 Копіювання файлів з кластеру

Для копіювання файлів з обчислювального вузла на локальний комп'ютер призначена команда:

C:\>pscp -P port user@host:file localfile

Приклад роботи команди:

C:\dir>pscp -P 2222 user@77.47.192.51:/home/user/dir/innerdir/foo2.txt
C:\dir\foo3.txt
user@77.47.192.51's password:
foo3.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:51 46 foo3.txt
27.04.2010 07:23 <DIR> innerdir
2 файлов 92 байт
3 папок 1 773 776 896 байт свободно
C:\dir>

3. Компіляція та запуск паралельних програм на кластері

3.1 Система управління чергою задач

3.1.1. Призначення системи управління чергою задач

Під час запуску програм на своєму персональному комп'ютері Ви точно знаєте, скільки програм вже запущено та скільки вони використовують ресурсів комп'ютера. Виходячи з цієї інформації, Ви вирішуєте, коли і скільки програм запускати.

Запуск програм на кластері відрізняється від запуску програм на персональному комп'ютері: Ви не знаєте, скільки та яких ресурсів використовується в даний час на кожному з вузлів кластеру. Тому Ви не можете самостійно обрати, на яких вузлах кластеру запускати свої програми. Для вирішення цієї проблеми використовується *система управління чергою завдань*. Ідея такої системи полягає в наступному: всі користувачі кластеру повідомляють системі управління чергою, які програми вони бажають запустити та скільки обчислювальних ресурсів цим програмам потрібно. Заявки від користувачів кластеру заносяться в чергу. Система управління чергою самостійно запускає програми з черги тоді, коли може задовольнити потреби програми в ресурсах. Таким чином, система управління чергою має повну інформацію про те, які ресурси кластеру зайняті (та якими саме програмами яких користувачів), а які ресурси кластеру вільні. Виходячи зі сказаного вище, можна зробити висновок: для продуктивної роботи всіх користувачів кластеру необхідно (та в інтересах користувачів) запускати обчислювальні програми на кластері тільки через систему управління чергою задач.

Після того, як Ви отримали доступ до командного рядка кластеру за допомогою протоколу *SSH*, Ви можете виконувати команди на вузлі *n001*. На цьому вузлі дозволяється виконувати переміщення та копіювання даних, компіляцію програм, запуск обчислювальних програм за допомогою системи управління чергою. На інших вузлах дозволяється виконувати лише інформаційні команди під час визначення причин проблем роботи програм.
На кластері Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» встановлена реалізація системи управління чергою задач *PBS* (англ. *Portable Batch System*) *Torque*. Нижче розглянуті основні команди для роботи з *Torque*.

3.1.2 Команда *qsub*

Команда *qsub* призначена для додавання програми в чергу виконання. Після успішного виконання програма видає на консоль ідентифікатор задачі в системі управління чергами (не плутати з рангом задачі в *MPI* або ідентифікатором задачі в *OpenMP*). Цей ідентифікатор знадобиться для відстежування стану виконання програми, або для видалення програми з черги.
qsub скрипт_старту_програми
скрипт_старту_програми – це ім'я файлу, в якому вказується деяка інформація для системи управління чергою задач та команди для запуску програми. Цей файл має наступний синтаксис:
#!/bin/bash
#PBS -S /bin/bash
#PBS -N <назва задачі>
#PBS -l <специфікація ресурсів>
#PBS -o <ім'я файлу для виводу stdout>
#PBS -e <ім'я файлу для виводу stderr>
cd <каталог з програмою>
<команда запуску програми>
Призначення рядків цього файлу наступне:
-S /bin/bash – параметр вказує планувальнику, що команди запуску програми слід виконувати за допомогою оболонки /bin/bash (краще не змінюйте цей параметр).
-N <назва задачі> – параметр задає назву задачі. Назва буде відображатись у списку задач (необов'язковий параметр).
-l <специфікація ресурсів> – параметр задає скільки та яких ресурсів необхідно виділити програмі. Різні види ресурсів розділяються комою. Можна задавати наступні види ресурсів:

- *walltime=HH:MM:SS* – необхідний час виконання програми (HH - години, MM - хвилини, SS - секунди). Це реальний час виконання програми, а не машинний. Якщо програма буде виконуватись довше заявленого часу, вона буде примусово завершена.
- *nodes=N:ppn=P* – необхідна кількість вузлів (*N*) та ядер на кожному вузлі (*P*). Всього програмі буде виділено $N \times P$ ядер.
- Хоча в *Torque* можна задавати ресурси оперативної пам'яті, на кластері Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря

Сікорського» оперативна пам’ять розподіляється за принципом: гарантований *1 Гб* на кожне обчислювальне ядро.

-o <ім’я файлу для виводу stdout> – повний шлях до файлу, в який буде записано стандартний потік виводу програми (при звичайному запуску вручну така інформація виводиться на консоль; необов’язковий параметр).

-e <ім’я файлу для виводу stderr> – повний шлях до файлу, в який буде записано стандартний потік помилок програми (при звичайному запуску вручну ця інформація виводиться на консоль; необов’язковий параметр).

Якщо не вказати імена файлів для запису виводу програми, то ці файли створюються у поточному каталозі, а імена файлів починаються з назви задачі.

3.1.3 Команда *qstat*

Команда *qstat* призначена для спостереження за станом виконання задач. Дану команду можна виконувати без параметрів:

```
user@n001$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1000.pbs                  example.sh       user            01:11:12 R batch
1001.pbs                  example42.sh     user            01:15:01 R batch
1002.pbs                  run.sh           user                   0 Q batch
```

Команда *qstat*.

В даному випадку видно, що користувач додав три задачі до черги. В стовпцях вказана наступна інформація:

- *Job id* – ідентифікатори задач;
- *Name* - ім’я скрипту старту програми або назва задачі (якщо задана).
- *Time Use* - кількість використаного процесорного часу.
- *S* - стан виконання задачі: «*Q*» - чекає в черзі, «*R*» - виконується.

Також команду *qstat* можна виконувати з параметром *-a* (виводить відносно більше інформації) чи з параметром *-f* (виводить дуже детальну інформацію).

3.1.4 Команда *qdel*

Команда *qdel* призначена для видалення задач із черги. Видаляти можна як задачу, яка очікує в черзі, так і задачу, яка вже виконується. В останньому випадку програма буде аварійно завершена.

user@n001$ qdel ідентифікатор_задачі

ідентифікатор_задачі – ідентифікатор задачі, який був виданий на консоль командою *qsub*. Якщо ви не пам’ятаєте цього ідентифікатору, дізнайтесь його командою *qstat*.

3.2 Компіляція та запуск програм на *MPI*

Нехай у вас є вихідні коди програми, написаної з використанням *MPI*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

user@n001$ mkdir /home/user/example

2) Завантажте у створений каталог вихідні коди програми.
3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

user@n001$ cd /home/user/example

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, то разом із програмою можуть поставлятись інструкції для компіляції. Приклади програм на мові *C*, які написані з використанням *MPI*, компілюються наступним чином:

```
user@n001$ mpicc -W -Wall -O2 -std=c99 \
ім'я_файлу_1.c ім'я_файлу_2.c ... -o program
```

ім'я_файлу_1.c, *ім'я_файлу_2.c* – імена всіх файлів, які входять до складу програми; *program* – ім'я для виконуваного файлу програми, яке створить компілятор. Якщо програма написана на мові C++, необхідно використовувати компілятор *mpic++*.

5) Створіть в каталозі програми скрипт запуску для системи управління завданнями (чи створіть цей файл на вашому персональному комп'ютері та завантажте його на кластер у каталог програми). Нехай ім'я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=7:ppn=2,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
mpiexec ./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити по 2 ядра на 7 вузлах, також програмі знадобиться якнайбільше 2 години часу для обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*.

Зверніть увагу! Запускаючи *MPI* програми на кластері через систему управління чергою завдвнь, не треба передавати команді *mpiexec* додаткових параметрів *-n* чи *-np*, які задають кількість завдань. Ця інформація буде передана до *mpiexec* системою управління чергою завдань автоматично.

6) Створіть або завантажте на кластер вхідні дані для програми.
7) Додайте програму в чергу виконання командою *qsub:*

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання завдання можна дізнатись за допомогою команди *qstat*.

3.3 Компіляція та запуск програм на *OpenMP*

Нехай у вас є вихідні коди програми, написаної з використанням *OpenMP*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

```
user@n001$ mkdir /home/user/example
```

2) Завантажте у створений каталог вихідні коди програми.
3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

```
user@n001$ cd /home/user/example
```

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, разом з програмою можуть поставлятись інструкції для

компіляції. Приклади програм на мові *C* з даного документу, що написані з використанням *OpenMP*, компілюються наступним чином:

```
user@n001$ gcc -W -Wall -O2 -std=c99 -fopenmp \
ім'я_файлу_1.c ім'я_файлу_2.c ... -o program
```

ім'я_файлу_1.c, *ім'я_файлу_2.c* – імена всіх файлів, що входять до складу програми; *program* – ім'я для виконуваного файлу програми, що створить компілятор. Якщо програма написана на мові *C++*, необхідно використовувати компілятор *g++*.

5) Створіть в каталозі програми скрипт запуску для системи управління задачами (чи створіть цей файл на вашому персональному комп'ютері та завантажте його на кластер у каталог програми). Нехай ім'я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=1:ppn=8,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
export OMP_NUM_THREADS=8
./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити 8 ядер на одному вузлі, програмі також знадобиться якнайбільше 2 години часу для обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*. Значення змінної *OMP_NUM_THREADS* має бути рівне кількості ядер, вказаних в параметрі *ppn=P*.

Зверніть увагу! Запускаючи *OpenMP* програми на кластері через систему управління чергою задач, не слід виділяти програмі більше одного вузла, оскільки програми на *OpenMP* працюють тільки на системах зі спільною пам'яттю.

6) Створіть або завантажте на кластер вхідні дані для програми.
7) Додайте програму в чергу виконання командою *qsub*:

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання завдання можна дізнатись за допомогою команди *qstat*.

3.4 Тестові запуски програм

Тестові (пробні для налагодження) запуски обчислювальних програм дозволяють виконувати завдання на інтерфейсному вузлі для уникнення очікування в черзі. Вимогою до тестових задач є використання не більше 4 ядер та час виконання – не більший 5 хвилин.

3.4.1 Тестовий запуск програм на *MPI*.

Виконайте завантаження програми та її вхідних даних на кластер, а також компіляцію програми. Після цього виконайте команду, де число 4 – це кількість завдань:

```
user@n001$ mpiexec -np 4 ./program
```

Далі розглянемо приклад розробки складної програми за допомогою MPI.[1]

Для вирішення реальної математичної задачі на кластерній системі з використанням *MPI* і створення необхідної програми використаєм відповідний математичний апарат.

Візьмемо двовимірне диференційне рівняння Пуассона у часткових похідних виду:

$$\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}=f(x,y).$$

Розв'язати його в аналітичному вигляді немає можливості. Одним із можливих методів розв'язання є застосування методу кінцевих різниць. Метод кінцевих різниць передбачає дискретизацію диференціальних рівнянь на прямокутних координатних сітках. Для двовимірних задач елементарні комірки таких сіток є прямокутниками, а для тривимірних задач комірки становлять паралелепіпеди.

Кінцево-різничні сітки. Розглянемо одновимірну область Θ, яка являє собою відрізок $[0, s]$. Розіб'ємо цей відрізок точками $x_i = ih$, $i = 0, 1, 2, \dots, n$ на n рівних частин довжиною $h = s/n$ кожна. Множина точок $G = \{x_i = ih | i = 0, 1, 2, \dots, n\}$ називається рівномірною одномірною координатною сіткою, а число h – кроком сітки.

Відрізок $[0, s]$ можна розбити на n частин, вводячи довільні точки $0 < x_1 < x_2 < \dots < x_{i-1} < x_i < x_{i+1} < \dots < x_{n-1} < s$.

Координатна сітка $G = \{x_i | i = 0, 1, 2, \dots, n, x_0 = 0, x_n = s\}$ буде мати крок $h_i = x_i - x_{i-1}$, що залежить від номера i вузла x_i. Якщо $h_i = h_{i+1}$ хоча б для одного номера i, координатна сітка G називається нерівномірною (рис.3.4).

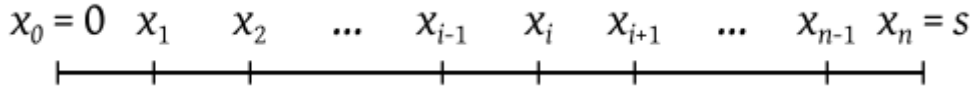


Рис.3.4. Одновимірна сітка.

Аналогічно, двовимірною сіткою називають множину точок $G = \{(x_i = ih_1, y_j = jh_2) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m\}$. Рівномірна двовимірна сітка являє собою множину $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (рис.3.5).

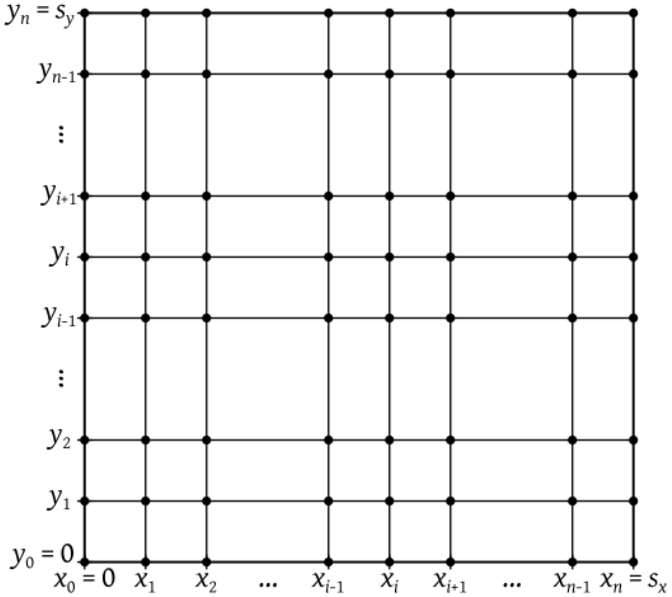


Рис. 3.5. Двовимірна сітка.

Сіткові функції, кінцеві різниці. Введення для області Θ координатної сітки G передбачає що значення всіх змінних та їх похідних розглядаються тільки у вузлах цієї сітки. З метою виконання цієї умови всі змінні задані сітковими функціями, а похідні будь-якого порядку - кінцевими різницями.

Нехай для деякої області Θ задана сітка $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$. Тоді функцію $\Phi = \Phi(x_i, y_j)$, $i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m$ дискретного аргументу (x_i, y_j) називають сітковою функцією, визначеною на сітці G.

Будь-якій неперервній функції $f(x, y)$, заданій в області Θ, можна поставити у відповідність сіткову функцію $\Phi(x_i, y_j)$ (для зручності будемо позначати Φ_{ij}), задану на сітці $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (спроектувати функцію $f(x, y)$ на сітку G), беручи до уваги певне правило співвідношення. Наприклад:

$$\Phi_{ij} = f(x_i, y_j)$$

$$\Phi_{ij} = \frac{1}{(x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}})(y_{i+\frac{1}{2}} - y_{i-\frac{1}{2}})} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{i-\frac{1}{2}}}^{y_{i+\frac{1}{2}}} f(x, y)\, dxdy,$$

де $x_{i\pm\frac{1}{2}}, y_{i\pm\frac{1}{2}}$ – координати серединних точок на відповідних кроках координатної сітки, що визначаються виразами:

$$x_{i+\frac{1}{2}} = \frac{x_{i+1} + x_i}{2}$$

$$x_{i-\frac{1}{2}} = \frac{x_i + x_{i-1}}{2}$$

$$y_{i+\frac{1}{2}} = \frac{y_{i+1} + y_i}{2}$$

$$y_{i-\frac{1}{2}} = \frac{y_i + y_{i-1}}{2}$$

Слід мати на увазі, що одна й та ж сіткова функція, задана на двох різних сітках, які мають спільні вузли, не обов’язково буде мати в цих вузлах рівні значення. За визначенням похідна функції неперервного аргументу x у точці x_0 є ліміт відношення приросту функції до приросту аргументу, коли приріст аргументу наближається до нуля:

$$\frac{\partial f(x)}{\partial x} = \lim_{(x - x_0) \to 0} \frac{f(x) - f(x_0)}{x - x_0}$$

Знехтувавши границею, похідну функції неперервного аргументу можна приблизно замінити (апроксимувати) різницевим виразом, заданим на відповідній сітковій функції $\Phi(x_i, y_j)$. Дана апроксимація може бути виконана декількома способами:

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{\Phi_{i+1, j} - \Phi_{i, j}}{\Delta x_i},$$

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{\Phi_{i, j} - \Phi_{i-1, j}}{\Delta x_i},$$

$$\frac{\partial f(x, y)}{\partial y} \approx \frac{\Phi_{i, j+1} - \Phi_{i, j}}{\Delta y_j},$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{\Phi_{i,j} - \Phi_{i,j-1}}{\Delta y_j},$$

де $\Delta x_i, \Delta y_j$ – кінцеві різниці координат, що визначаються виразами:

$\Delta x_i = x_{i+1} - x_i$,

$\Delta y_i = y_{j+1} - y_j$.

Для кожного з перетворень характерна така похибка апроксимації, яка прямує до нуля коли крок сітки прямує до нуля.

Похідні другого порядку апроксимуються наступним чином:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta x_i} - \frac{\Phi_{i,j} - \Phi_{i-1,j}}{\Delta x_{i-1}}}{\frac{\Delta x_i + \Delta x_{i-1}}{2}}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta y_i} - \frac{\Phi_{i,j} - \Phi_{i,j-1}}{\Delta y_{j-1}}}{\frac{\Delta y_i + \Delta y_{i-1}}{2}}$$

У випадку, коли сітка рівномірна, вирази спрощуються:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{\Delta y^2}.$$

Аналіз задачі

Застосувавши вище приведений метод до нашої задачі, ми можемо визначити структуру обчислювальної моделі та формули, за якими будемо вести розрахунок. Структура обчислювальної моделі зображена на рис.3.6. Сітка дискретизації квадратна та рівномірна. У кожному обчислювальному вузлі буде послідовно розраховуватися матриця точок сітки дискретизації. Реальні вузли будуть працювати паралельно між собою та обмінюватися даними.

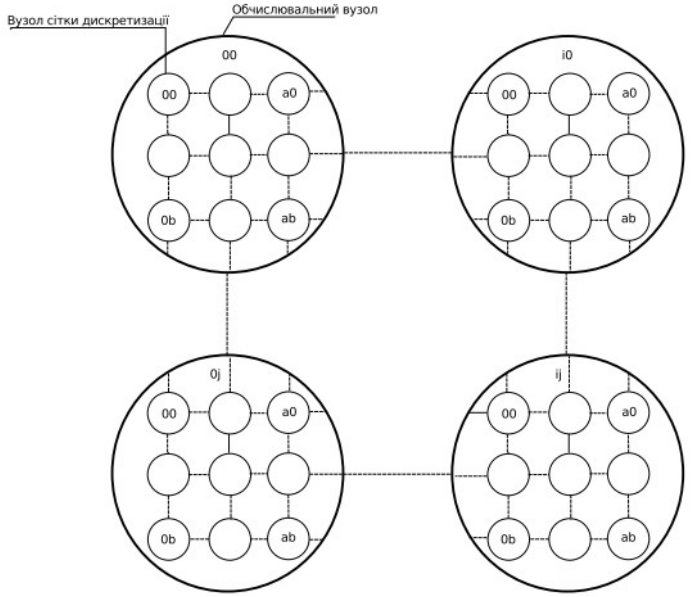


Рис.3. 6. Структура обчислювальної моделі.

Нехай початкові умови функції всередині області $U_0 = 0$, а на границі області $U_0 = (1 + 3\pi) \sin(3\pi(x + y))$.

$$U_{nm}(z)=U_{nm}(z-1)-\omega\frac{h^2}{4}(LU_{nm}-f_{nm}),$$

де:

- n, m – координати вузла сітки дискретизації;
- z – номер ітерації;
- h – крок у просторі, що визначається за формулою $h = 1/N$;
- N – кількість вузлів сітки дискретизації вздовж однієї з координат;
- f_{nm} – значення правої частини диференційного рівняння у вузлі сітки дискретизації;
- LU_{nm} – різницевий оператор, що визначається за формулою:

$$LU_{nm}=U_{nm}-\frac{U_{n+1,m}-2U_{n,m}+U_{n-1,m}}{h^2}-\frac{U_{n,m+1}-2U_{n,m}+U_{n,m-1}}{h^2}.$$

Локальним критерієм завершення алгоритму слугує досягнення заданої точності:

$$\frac{U_{nm}(z)-U_{nm}(z-1)}{U_{nm}(z)}\le\varepsilon,$$

де ε задається до початку розрахунку.

Виконання

Спрощений алгоритм основного циклу програми виглядає наступним чином:

1. Зробити розрахунки для тих вузлів сітки дискретизації, які не потребують даних від сусідніх процесів.
2. Отримати колонку сітки дискретизації від лівого сусіднього процесу, якщо він є.
3. Передати крайню праву колонку сітки дискретизації до правого сусіднього процесу, якщо він є.
4. Отримати колонку сітки дискретизації від правого сусіднього процесу, якщо він є.
5. Передати крайню ліву колонку сітки дискретизації до лівого сусіднього процесу, якщо він є.
6. Отримати рядок сітки дискретизації від верхнього сусіднього процесу, якщо він є.
7. Передати нижній рядок сітки дискретизації до нижнього сусіднього процесу, якщо він є.
8. Отримати рядок сітки дискретизації від нижнього сусіднього процесу, якщо він є.
9. Передати верхній рядок сітки дискретизації до верхнього сусіднього процесу, якщо він є.
10. Завершити розрахунки для інших вузлів, які не були розраховані у пункті 1.
11. Перевірити локальні критерії завершення алгоритмів. Якщо всі вузли сітки у процесі завершили свою роботу- завершити розрахунки у процесі.
12. Якщо всі процеси завершили розрахунки - завершити алгоритм.

Розглянемо деякі проблеми, зв’язані з реалізацією цього алгоритму:

1) В мові *C* матриці зберігаються по рядках (елементи одного рядку розташовані послідовно в пам’яті). Алгоритм розв’язання задачі передбачає

передачу рядків та стовпців матриці. Елементи одного рядку можна передати за допомогою звичайного виклику *MPI_Send*, оскільки вони розташовані послідовно. Елементи одного стовпця таким чином неможливо передати, тому що вони розташовані в пам'яті з кроком, рівним ширині матриці. Для передачі стовпців матриці створимо тип «стовпець матриці» за допомогою функції конструктора типу *MPI_Type_vector* наступним чином:

```
MPI_Type_vector( <висота матриці>,
1, /* довжина одного блоку */
<ширина матриці>,
MPI_DOUBLE,
&column_type);
```

Для передачі стовпця матриці цей тип можна застосувати наступним чином:

```
MPI_Send( <Покажчик на перший елемент стовпця>,
1, /* кількість стовпців */
column_type,
<ранг задачі- отримувача>,
<тег>,
MPI_COMM_WORLD);
```

2) Аналогічна проблема виникає при початковому розподілі координат вузлів сітки дискретизації по процесах. Фактично нам потрібно відправити квадратну під-матрицю. Реалізація змінюється тільки у ширині блоку і їх кількості при конструюванні типу.

```
MPI_Type_vector( <висота блоку, що відправляємо>,
<ширина блоку, що відправляємо>,
<ширина матриці>,
MPI_DOUBLE,
&new_type);
```

3) Попри те, що процес закінчив розрахунок, він усе одно повинен взаємодіяти з сусідніми процесами, щоб ті теж могли завершити розрахунки. Для цього треба реалізувати подвійну перевірку – умови зупинки розрахунку процесу та умови глобального кінця розрахунку. У програмі спочатку виконується перевірка усіх локальних умов закінчення алгоритму. Якщо усі вузли сітки дискретизації, що належать до процесу, закінчили роботу, то встановлюється ознака закінчення обчислень у процесі. Після цього усі ознаки закінчення обчислень у процесах збираються нульовим процесом і над ними виконується операція логічного «*І*» за допомогою *MPI_Reduce*.

```
MPI_Reduce( &node_stop,
&global_stop,
1,
MPI_SHORT,
MPI_LAND,
0,
MPI_COMM_WORLD);
```

Результат цієї операції є глобальною ознакою закінчення алгоритму, оскільки він матиме значення *true* тільки коли усі процеси закінчили обчислення. Після

цього глобальна ознака закінчення обчислення розповсюджується між усіма процесами за допомогою *MPI_Bcast*.

Код програми

Файл *code/lab_diffeq/main.c*

```
#define _GNU_SOURCE
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdbool.h>
/* Опис типів, структур та функцій */
struct my_matrix /* Структура, що описує матриці */
{
int rows; /* Кількість строк у матриці */
int cols; /* Кількість стовпців у матриці */
double *data; /* Покажчик на матрицю, розвернуту у одномірний масив*/
};
MPI_Datatype node_mat_t; /* Тип для розповсюдження частин сітки
дискретизації по матриці процесів*/
MPI_Datatype mat_col; /* Тип для передачі стовпця матриці */
void inline evaluate( int); /*Функція, що містить ітераційні формули */
double func( double X, double Y); /* Повертає значення правої частини
диференційного рівняння у точці X,Y*/
void init_grid(); /* Ініціалізує значення сітки дискретизації у процесі */
void fatal_error( const char *message, int errorcode); /* Функція виведення
помилок вводу-виводу */
struct my_matrix *matrix_alloc( int rows, int cols, double initial); /* Функція для
виділення пам’яті та ініціалізацію матриці*/
void matrix_print( const char *filename, struct my_matrix *mat); /* Функція
для виведення матриці у файл*/
struct my_matrix *read_matrix( const char *filename); /* Функція
для зчитування матриці із файлу*/
/*
**********************************************************************
**** */
/* Опис змінних */
int np; /* Кількість процесів */
int rank; /* Ранг процесу у MPI_COMM_WORLD */
int nodes_width; /* Ширина матриці процесів */
int node_X; /* Координата X у матриці процесів */
int node_Y; /* Координата Y у матриці процесів */
int total_width; /* Ширина матриці вузлів сітки дискретизації*/
int points_per_node; /* Ширина частини матриці вузлів сітки дискретизації,
```

що припадає на кожен процес*/
const char* inpfileX=”inpX.csv”; /* Ім’я файлу з якого вводяться горизонтальні координати сітки дискретизації */
const char* inpfileY=”inpY.csv”; /* Ім’я файлу з якого вводяться вертикальні координати сітки дискретизації */
const char* inpfileInit=”inpInit.csv”; /*Ім’я файлу з якого вводяться початкові та граничні умови для сітки дискретизації*/
const char* outpfile=”outp.csv”; /* Ім’я файлу куди виводиться результат */
my_matrix* allcoords_X; /* Матриця реальних горизонтальних координат вузлів сітки дискретизації */
my_matrix* allcoords_Y; /* Матриця реальних вертикальних координат вузлів сітки дискретизації */
my_matrix* all_grid_init; /*Всі початкові та граничні умови функції*/
my_matrix* total_mat; /* Повна сітка дискретизації, що виводиться у процесі 0*/
my_matrix* grid_init; /*Частина початкових умов для процесу*/
my_matrix* coords_X; /* Частина матриці реальних координат сітки дискретизації, яка зберігається у кожному з процесів*/
my_matrix* coords_Y; /* Частина матриці реальних координат сітки дискретизації, *яка зберігається у кожному з процесів*/*
my_matrix* node_mat; /* *Частина сітки дискретизації, яка зберігається у кожному з процесів*/*
my_matrix* f_xy; /* *Масив значень функції в вузлах сітки дискретизації*/*:
MPI_Status stat; /* *Змінна, у яку повертається статус прийому повідомлень */*
double epsilon=0.01; /* *Точність рішення*/*
double omega=0.4; /* *Коефіцієнт релаксації*/*
int max_iter=1000000; /**Максимальна кількість ітерацій у випадку нев’язки функції */*
double h; /* *Крок сітки дискретизації */*
bool* local_stop; /* *Масив для запам’ятовування локальних умов зупинки */*
int* node_iter;
short node_stop; /* *Ознака завершення обчислень у процесі*/*
short global_stop; /* *Глобальна ознака завершення обчислень*/*
double* left_col; /* *Стовпці, що процес буде приймати з лівого та правого процесів під час ітерацій */*
double* right_col;
double* top_row; /* *Рядки, що процес буде приймати з верхнього та нижнього процесів під час ітерацій */*
double* bot_row;
/*
****
**** */
/ Головна програма */*
int main(**int** argc, **char** *argv[])
{
MPI_Init(&argc, &argv); /* *Ініціалізуємо середовище */*
MPI_Comm_size(MPI_COMM_WORLD, &np); /* *Отримуємо розмір*

```
MPI_COMM_WORLD */
MPI_Comm_rank( MPI_COMM_WORLD, &rank); /* Отримуємо ранг процесу у
MPI_COMM_WORLD*/
nodes_width=sqrt(np); /* Отримуємо ширину матриці процесів*/
node_X=rank%nodes_width; /* Отримуємо горизонтальну координату процесу
у матриці */
node_Y=rank/nodes_width; /* Отримуємо вертикальну координату процесу у
матриці */
if (rank==0)
{
allcoords_X=read_matrix(inpfileX); /* Вводимо координати */
allcoords_Y=read_matrix(inpfileY); /* Вводимо координати */
all_grid_init=read_matrix(inpfileInit); /* Вводимо координати */
total_width=allcoords_X- >rows; /* Отримуємо ширину сітки дискретизації */
}
MPI_Bcast( &total_width, 1, MPI_INT, 0, MPI_COMM_WORLD); /*
Розповсюджуємо ширину
сітки дискретизації */
h=1.0/total_width; /* Визначаємо крок сітки дискретизації */
points_per_node=total_width/nodes_width;
/* Створюємо тип, що відповідає частині матриці, яка буде передаватися */
MPI_Type_vector(points_per_node,points_per_node,total_width, MPI_DOUBLE,
&node_mat_t);
MPI_Type_commit( &node_mat_t); /* Реєструємо тип */
coords_X=matrix_alloc(points_per_node,points_per_node, 0.0);
coords_Y=matrix_alloc(points_per_node,points_per_node, 0.0);
grid_init=matrix_alloc(points_per_node,points_per_node, 0.0);
if(rank==0)
{/* Процес 0 розсилає реальні координати сітки дискретизації*/
for( int i=0;i<nodes_width;i++)
{
for( int j =0;j <nodes_width;j ++)
{
if( ! (i==0&&j ==0))
{
MPI_Send(allcoords_X- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(allcoords_Y- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(all_grid_init- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
}
}
}
for( int i=0;i<points_per_node;i++)
{
```

```
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =allcoords_X- >data[i*total_width+j];
 coords_Y- >data[i*points_per_node+j] =allcoords_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =all_grid_init- >data[i*total_width+j];
}
}
}
else
{/* Інші процеси їх приймають */
my_matrix* temp_X=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_Y=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_init=matrix_alloc(total_width,points_per_node, 0);
MPI_Recv(temp_X- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_Y- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_init- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =temp_X- >data[i*total_width+j];
coords_Y- >data[i*points_per_node+j] =temp_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =temp_init- >data[i*total_width+j];
}
}
}
/* Ініціалізуємо умови локальної зупинки алгоритму*/
local_stop=(bool*)malloc(points_per_node*points_per_node*sizeof(bool));
for( int i=0;i<points_per_node*points_per_node;i++)
{
local_stop[i] =false;
}
for( int i=0;i<points_per_node;i++) /* Граничні точки залишаються
постійними*/
{
if(node_Y==0)
{
local_stop[i] =true;
}
if(node_X==0)
{
local_stop[i*points_per_node] =true;
}
if(node_X==nodes_width- 1)
{
local_stop[(i+1) *points_per_node- 1] =true;
```

```
}
if(node_Y==nodes_width- 1)
{
local_stop[points_per_node*(points_per_node- 1) +i] =true;
}
}
/* Ініціалізуємо і встановлюємо початкові значення внутрішнього діапазону */
node_mat=matrix_alloc(points_per_node,points_per_node, 0);
/* Встановлюємо початкові значення граничних вузлів сітки дискретизації */
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
/* Створимо масив значень функції в локальних вузлах сітки дискретизації*/
f_xy=matrix_alloc(points_per_node,points_per_node, 0.0);
for( int i=0;i<points_per_node*points_per_node;i++)
{
f_xy- >data[i] =func(coords_X- >data[i],coords_Y- >data[i]);
}
/*Визначимо типи для передач під час ітерацій*/
/* Тип, що визначає стовпець матриці*/
MPI_Type_vector(points_per_node, 1,points_per_node, MPI_DOUBLE, &mat_col);
MPI_Type_commit( &mat_col); /* Реєструємо тип */
left_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
right_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
top_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
bot_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
node_iter=( int*)malloc(points_per_node*points_per_node*sizeof( int));
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_iter[i] =0;
}
/* Початок ітерацій */
do
{
/* Обмін між процесами у сітці*/
if(node_X! =0)
{
/* Приймемо стовпець з лівого процесу */
MPI_Recv(left_col, 1,mat_col,rank- 1,rank- 1, MPI_COMM_WORLD, &stat);
}
if(node_X! =nodes_width- 1)
{
/* Відішлемо стовпець до правого процесу */
MPI_Send(node_mat- >data+points_per_node- 1, 1,mat_col
,rank+1,rank, MPI_COMM_WORLD);
```

```
/* Приймемо стовпець з правого процесу */
MPI_Recv(right_col, 1,mat_col,rank+1,rank+1, MPI_COMM_WORLD, &stat);
}
if(node_X! =0)
{
 /* Відішлемо стовпець до лівого процесу */
MPI_Send(node_mat- >data, 1,mat_col,rank- 1,rank, MPI_COMM_WORLD);
}
if(node_Y! =0)
{
/* Приймемо строку з верхнього процесу */
MPI_Recv(top_row,points_per_node, MPI_DOUBLE,rank- nodes_width
,rank- nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =nodes_width- 1)
{
/* Відішлемо строку до нижнього процесу */
MPI_Send(node_mat- >data+(points_per_node- 1)
*points_per_node,points_per_node
, MPI_DOUBLE,rank+nodes_width,rank, MPI_COMM_WORLD);
/* Приймемо строку з нижнього процесу */
MPI_Recv(bot_row,points_per_node, MPI_DOUBLE,rank+nodes_width,
rank+nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =0)
{
/* Відішлемо строку до верхнього процесу */
MPI_Send(node_mat- >data,points_per_node, MPI_DOUBLE,
rank- nodes_width,rank, MPI_COMM_WORLD);
}
/* Виклик функції, що проводить ітераційну обробку*/
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
evaluate(i*points_per_node+j);
}
}
/*Якщо усі вузли сітки дискретизації у процесі завершили
обчислення - встановлюємо ознаку завершення обчислень у процесі*/
node_stop=1;
for( int i=0;i<points_per_node*points_per_node;i++){
if ( ! local_stop[i]){
node_stop=0;
break;
}
```

```
}
global_stop=1;
/*Перевіряємо чи всі процеси закінчили обчислення*/
MPI_Reduce( &node_stop, &global_stop, 1, MPI_SHORT, MPI_BAND, 0,
MPI_COMM_WORLD);
 /*Якщо всі - то посилаємо сигнал про завершення*/
MPI_Bcast( &global_stop, 1, MPI_SHORT, 0, MPI_COMM_WORLD);
} while( ! global_stop);
/* Збираємо результат */
total_mat=matrix_alloc(total_width,total_width, 0.0);
if(rank==0){
for( int i=0;i<nodes_width;i++){
for( int j =0;j <nodes_width;j ++){
if( ! (i==0&&j ==0)){
my_matrix* temp=matrix_alloc(points_per_node,points_per_node, 0.0);
/* Приймаємо частини матриці сітки дискретизації з кожного процесу */
MPI_Recv(temp- >data,points_per_node*points_per_node,
MPI_DOUBLE,i*nodes_width+j,i*nodes_width+j, MPI_COMM_WORLD, &stat);
/* Та розташовуємо їх у повній матриці */
for( int k=0;k<points_per_node;k++){
for( int l=0;l<points_per_node;l++){
total_mat- >data[i*total_width*points_per_node+
j *points_per_node+k*total_width+l]
=temp- >data[k*points_per_node+l];
}
}
}
}
}
for( int i=0;i<points_per_node;i++){
for( int j =0;j <points_per_node;j ++){
total_mat- >data[i*total_width+j] =node_mat- >data[i*points_per_node+j];
}
}
}
else{
MPI_Send(node_mat- >data,points_per_node*points_per_node
, MPI_DOUBLE, 0,rank, MPI_COMM_WORLD);
}
if (rank==0){
matrix_print(outpfile,total_mat);;
}
MPI_Finalize();
}
/*
**********************************************************************
```

```
***** */
/* Допоміжні функції */
double func( double X, double Y)
{
return X*Y;
}
/*Функція, що містить ітераційні формули */
 void inline evaluate( int index)
{
/*Якщо досягнута потрібна точність,
то не потрібно проводити обчислення */
if (local_stop[index])
{
return;
}
/*Якщо вузол сітки знаходиться на границі між обчислювальними
вузлами, то потрібно взяти значення з буфера передачі*/
double left=index%points_per_node==0
?left_col[index] : node_mat- >data[index - 1];
double right=(index + 1) %points_per_node==0
?right_col[index+1- points_per_node] : node_mat- >data[index + 1];
double top=index/points_per_node == 0 ? top_row[index % points_per_node]
: node_mat- >data[index- points_per_node];
double bottom=index/points_per_node==points_per_node- 1
?bot_row[index%points_per_node]
: node_mat- >data[index+points_per_node];
/*Власне обчислення*/
double old_node=node_mat- >data[index];
double LU=old_node- (left+right+top+bottom- 4*old_node) /(h*h);
node_mat- >data[index] =old_node- omega*h*h/4*(LU- f_xy- >data[index]);
/*Якщо досягнута точність або перевищена максимальна кількість ітерацій,
то завершуємо обчислення у цьому вузлі сітки*/
if (fabs(node_mat- >data[index] - old_node) <=
fabs(node_mat- >data[index] *epsilon) | | ++node_iter[index] >max_iter)
{
local_stop[index] =true;
}
}
/* Ініціалізує значення сітки дискретизації у процесі */
void init_grid()
{
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
}
```

```
void fatal_error( const char *message, int errorcode)
{
printf( ”fatal error: code %d, %s\n”, errorcode, message);
fflush(stdout);
MPI_Abort( MPI_COMM_WORLD, errorcode);
}
struct my_matrix *matrix_alloc( int rows, int cols, double initial)
{
struct my_matrix *result = (my_matrix*)malloc( sizeof( struct my_matrix));
result- >rows = rows;
 result- >cols = cols;
result- >data = ( double*)malloc( sizeof( double) * rows * cols);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
result- >data[i * cols + j] = initial;
}
}
return result;
}
void matrix_print( const char *filename, struct my_matrix *mat)
{
FILE *f = fopen(filename, ”w”);
if(f == NULL)
{
fatal_error( ”cant write to file”, 2);
}
for( int i = 0; i < mat- >rows; i++)
{
for( int j = 0; j < mat- >cols; j ++)
{
fprintf(f, ”%lf ”, mat- >data[i * mat- >cols + j]);
}
fprintf(f, ”\n”);
}
fclose(f);
}
struct my_matrix *read_matrix( const char *filename)
{
FILE *mat_file = fopen(filename, ”r”);
if(mat_file == NULL)
{
fatal_error( ”can’t open matrix file”, 1);
}
int rows;
```

```
int cols;
fscanf(mat_file, ”%d %d”, &rows, &cols);
struct my_matrix *result = matrix_alloc(rows, cols, 0.0);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
fscanf(mat_file, ”%lf”, &result- >data[i * cols + j]);
}
}
fclose(mat_file);
return result;
}
```

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення Grid і суперкомп’ютерів
2. Перерахуйте особливості програмування суперкомп’ютерів
3. Що таке координація розрізнених ресурсів?
4. На що націлені GRID- обчислення і суперкомп’ютерні обчислення ?
5. Назвати обчислювальні ресурси суперкомп’ютера
6. Що таке віддалені процедури, сокети?
7. Призначення Middleware
8. Перерахуйте найбільш відомі пакетіи ППЗ для суперкомп’ютерних обчислень
9. Яка модель взаємодії процесів використовується в КС з локальною пам’яттю?
10. Охарактеризуйте бібліотеку MPI
11. Що таке комунікатор в MPI?
12. Охарактеризуйте особливості алгоритмів планування розміщення завдання на розподілених ресурсах.
13. Що являє собою бібліотека OpenMP?
14. Як в MPI задається кількість необхідних процесів?

Література основна

1. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.
2. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.
3. Інструкція по використанню кластеру НТУУ «КПІ ім. Ігоря Сікорського» [Електронний ресурс]. – Режим доступу: http://hpcc.kpi.ua/uk/.

Література допоміжна

4.Г.И.Шпаковский, В.И.Стецюренко, А.Е. Верхотуров, Н.В. Серикова. Применение технологии MPI в Грид. – Минск, БГУ, 2008 , - 137 с.

1.4 **Розподілені бази даних**

В сучасних додатках проблема масштабованості перемістилася переважно в область керування даними і ми досягли того рівня, коли вибір мови програмування, операційної системи і навіть обладнання і порядку експлуатації (дякуючи хостингу віртуальних машин і "хмарам") настільки спростилися і здешевилися, що практично перестали бути проблемою і визначаються особистими уподобаннями і необхідністю. Якщо хочите створювати масштабований додаток, то Ваш вибір перш за все – **база даних**, бо саме від того, яку базу даних виберете, залежить простота збору, зберігання, обробки, аналізу і видалення даних. Визначитися з вибором бази даних часто буває складніше, ніж вибрати жанр для даних із конкретної предметної області.

Ви вчили і добре обізнані з реляційними базами. Їх дуже багато, більше сотні, вони домінуючі і будуть залишатись ними в і майбутній перспективі, але існують і інші, які все більше і більше привертають увагу. Їх теж вже немало і їх поява обумовлена пошуком альтернатив і зокрема: структур даних без схем, нетрадиційних структур даних, простої аплікації, високої доступності, горизонтального масштабування та нових способів формулювання запитів. Всі ці технології отримали збиральну назву *NoSQL*. Так як ці питання досить обширні і не можуть бути осягнуті навіть за допомогою кількох лекцій, тому розглянемо їх в вигляді тез. Детальніше їх освойте самотужки в рамках самостійної роботи студента.

Фрагментально розглянемо сім баз даних: реляційні, ключі і значення, стовпцеві, документо-орієнтовані, графові. Оглянемо бази, покриваючи всі категорії: одну реляційну (Postgres), два сховища ключів і значень (Riak, Redis), стовпцеву (HBase), дві документо-орієнтованих (MongoDB, CouchDB) та одну графову (Neo4J).

Бази призначені для вирішення конкретних практичних задач. Реляційні з'явились, коли гнучкість запитів вважалась важливішою за гнучкість схеми. Стовпцеві добре пристосовані для зберігання великих об'ємів на декількох машинах і зв'язок між даними відходить на інший план. Майже всі задачі можна вирішити за допомогою названих СУБД і питання лише наскільки ефективне використання їх в конкретній предметній області при певних сценаріях використання і наявних ресурсах.

1.***Реляційні СУБД*** .

Реляційні системи управління базами даних (РСУБД) основані на теорії множин, в основі їх реалізації лежать таблиці із рядків і стовпців. Спосіб взаємодії з РСУБД – запити на мові Structured Query Language (SQL).

Дані типізовані, це можуть бути числа, рядки, дати, неструктуризовані двійкові об'єкти.

PostgreSQL – найстаріша і провірена часом СУБД. PostgreSQL – одна із кращих РСУБД с відкритим кодом.

Сильні сторони PostgreSQL Багато років досліджень і промислової експлуатації практично у всіх областях, де використовуються комп'ютери, гнучкі засоби запитів, найвищий рівень непротиріч та довговічності даних. Для Postgres написані та вивірені драйвери большості мов програмування. Можна писати власні мовні разширення, вводити нові типи індексів, створювати власні типи даних і навіть переписувати плани виконання запитів. Якщо для інших СУБД з відкритим початковим кодом існують хитрі лицензійні домовленості, то PostgreSQL – квінтэсенція відткритості. У коду взагалі нема володаря. Всякий може робити з проектом практично все що завгодно, тільки не кладучи при цьому відповідальності на авторів.

Слабкі сторони PostgreSQL

Сегментування – не найсильніша сторона реляційних баз і PostgreSQL в тім рахунку. Якщо вимоги масштабування по горизонталі, а не по вертикалі (декілька паралельних сховищ даних, а не одна суперпотужна машина чи кластер), то краще використати щось інше.

2 **Riak**

Riak –це розподілене сховище ключів і значень, в якому значенням може бути що завгодно – простий текст, документ в форматі JSON чи XML, зображення чи відеокліп. Для доступу до сховища надається простий HTTP-інтерфейс.

Riak зможе зберегти будь-які дані. Riak також може похвалитися відмовоздатністтю. Любий сервер може бути зупинений чи запущений в любий момент, точки загальнї відмови не існує. Кластер продовжує працювати при видаленні, добавлені чи аварійній відмові серверів. Відмова одного вузла – не критична ситуація. Riak не має хорошої підтримки довільних запитів, а сховище ключів і призначень, по самій своїй природі, погано з'вязуються одне з одним (іншими словами, поняття зовнішнього ключа відсутнє). Riak «розмовляє на мові веб» краще, ніж всі інші. Riak –кращий вибір для центрів обробки даних – таких, як Amazon, – які повинні обслуговувати багато запитів з низькою затримкою. В усмовах, коли кожна лишня мілісекунда очікування означае згубити потенціального клієнта, Riak найкраще. Вона проста в налаштуванню та адмініструванні і може рости разом з вимогами.

Riak – перша із розглянутих нами баз даних NoSQL. Це розподілене репліцуємe сховище ключів і значень з доповненими функціями, не маюче точки загальної відмови.

Для працюючого тільки з реляційними базами данх, Riak може здатися дивним створінням. В нім нема ні транзакцій, ні SQL, ні схеми. Є ключі, але з'вязування сегментів зовсім не схоже на з'єднання таблиць. А технологія mapreduce взагалі покажеться чимось малозрозумілим. Однак для вирішення певного класу задач всі ці компроміси виправдані. Здатність Riak масштабуватися шляхом збільшення кількості серверів (а не нарощування потужності одного сервера) та простота використання –кращий приклад спроби вирішити специфічні проблеми масштабування в веб.

Riak опирається на уже існуючу структуру HTTP, даючи максимальну гнучкість для побудови каркасів чи систем з підтримкою веб.

Сильні сторони Riak

При проектуванні широкомасштабної системи замовлень нашталкт Amazon, або коли ставиться на перше місце висока доступність, то Riak, безумовно, заслуовує на увагу. Один із беззаперечних плюсіів Riak – увага до усунення точок загальної відмови з метою забезпечити безперебійну роботу та нарощування чи скрочення кластеру у відповідності зі змінюючимися вимогами. Якщо структура даних не дуже складна, то працювати з Riak буде просто. Але при цьому зберігається можливість маніпуляцій з даними, якщо це необхідно. На даний час підтримуються десяток мов (повний перелік можна знайти на сайті Riak), але, при готовності писати на Erlang, можна разширювати ядро в будь-яку сторону. А якщо потрібна більша швидкодія, ніж здатен забезпечити протокол HTTP, то можете спробовати обмінюватися даними по більш ефективному двійковому транспортному протоколу Protobuf19.

Слабкі сторони Riak

Якщо потрібні прості засоби формулювання запитів, складні структури даних або жорстка схема чи нема потреби в горизонтальному масштабуванню, то Riak, мабуть, не потрібний.

Один з основних недоліків Riak – відсутність простих і надійних засобів формулювання довільних запитів. Технология mapreduce дає фантастичну функциональність, але хотілось би бачити більше вбудованих дій, основаних на URL чи PUT-запитах. Добавлення індексування було вагомим кроком в правильному напрямку, але хотілось би, щоб ці ідеї отимали подальший розвиток. І накінець, при бажанні писати на Erlang, можна зіштовхнутися з деякими обмеженнями JavaScript – неможливості написати функції, визиваємі в точці підключення після фіксації, – і недостатньою швидкодією mapreduce. Однако команда Riak працює і над цими питаннями.

Riak и теорема CAP

(CAP - Consistency, Availability, partition Tolerance (узгодженість, доступність, стійкість до втрати зв’язаності). Слід зазначити , що слово *consistency* в абревіатурах ACID і CAP означає різні речі). Теорема CAP розкриває неприглядну правду про те, як розподілені системи ведуть себе в умовах нестабільної мережі. CAP стверджує, що можна створити розподілену систему, яка буде узгодженою (consistent) (тобто операції записі атомарні і всі наступні операції читання бачуть нові значення), доступною (available) (база даних повертає значення поки працює хоч один сервер) і сітйка до втрати зв’язності (partition tolerant) (система продовжує функціонувати, навіть якщо зв’язок між серверами тимчасово відсутній), але одночасно можна гарантувати тільки дві із

цих трьох властивостей. Іншими словами, можна створити розподілену систему, яка буде узгодженою і стійкою до втрати зв'язаності, систему, яка буде доступною і стійкою до втрати зв'язаності, або систему, яка буде узгодженою і стійкою (але при цьому не буде стійкою до втрати зв'язаності, тобо, по суті, не буде розподіленою). Але не можливо створити розподілену базу даних, яка була би однчасно узгодженою, доступною і стійкою до втрати зв'язаності.

Теорема CAP суттєва при проектуванні розподіленої бази даних, так як Ви повинні вирішити, чем готові пожертвувати.

Яку би базу даних Ви не вибрали, вона зможе гарантувати або доступність, або узгодженість. Стійкість до втрати зв'язаності – чисто архітектурне рішення (від неї залежить, чи буде система взагалі розподіленою). Важливо зрозуміти сенс теореми CAP, щоб реалістично оцінювати можливості. Компроміси, прийняті в різних описаних базах даних, спираються саме на цю теорему.

Riak гарно обходить обмеження , які накладає теорема CAP на всяку разподілену базу даних. І це дивовижно при порівнянні з PostgreSQL, яка здебільшого підтримує тільки строгу погодженість записі. В Riak використовується ідея про те, що умови CAP можна задавати на рівні сегментів чи запитів. Це значний крок в бік надійної і гнучкої СУБД з відкритим початковим кодом. Знайомлячись з іншими базаи даних, згадуйте про Riak – вас постійно буде дивувати його гнучкість. При необхідності зберігати гігантський каталог даних Riak може здатися непоганим вибором.

Якщо необхідна база даних, яка вбудовується в пристрій, чи потрібно оброблювати фінансові транзакції, то Riak не годиться. Якщо стоїть задача забезпечити горизонтальне масштабування чи обслуговувати інтенсивний потік запитів через веб, то до Riak слід придивитися.

2 HBase

Система Apache HBase створена для серйозних робіт. Якщо об'ем даних не вимірюється багатьма гігабайтами, то варто взяти щось простіше. На перший погляд, HBase дуже схожа на реляційну базу даних – настільки, що, не знаючи, з чим маєте справу, можна сплутати одну з іншою. Сама складна частина в вивченні HBase – не технологія; проблема в тім, що багато термінів, використовуваних в HBase, оманливо знайомі. Наприклад, HBase зберігає дані в контейнерах, які називаються *таблицями*. Таблиці же складаються із *комірок*, які знаходяться на перетині *рядків* і *столпців*. Таблиці HBase ведуть себе зовсім не так, як відношення, рядки зовсім не схожі на записі, а склад стовпців може бути змінним (схема його не контролює).

В HBase вбудований ряд функцій, відсутній в інших базах даних, напрклад, версіонування, стиснення, прибирання сміття (для даних з вичерпаним терміном зберігання) і таблиці в памяті. Так як ці можливості мають початкову присутність, значить, прийдеться писати меньше кодів, коли вони знадобляться. Крім того, HBase дає строгі гарантії непротирічності, що спрощує перехід від реляційних баз даних. При обробці трудомістких запитів HBase часто обжене інші СУБД. Цим

пояснюється, чому HBase так часто використовується в великих компаніях для реалізації систем аналізу журналів та пошуку.

HBase – це *стовпчикова* база даних , яка може похвалитися підтримкою непротирічності та горизонтальною масштабованністю. Вона влаштована по зразку BigTable , високопродуктивної закритої бази даних, яка розроблена Google та описана в статті «Bigtable: A Distributed Storage System for Structured Data» (http://research.google.com/archive/bigtable.html) .

HBase – це сімбіоз простоти і складності. Сама модель зберігання даних відносно проста, в схему встроюються лише деякі обмеження. Однако при вивченні системи значно мішає, що термінологія запозичена із реляційних баз даних (наприклад, це відноситься до слів *таблиця та стовчик*). При проектуванні схеми HBase наріжним каменем є спроможність таблиць і стовбців.

Сильні сторони HBase

Серед особливостей HBase уваги заслуговує архітектура горизонтальної масштабованості і вбудовані засоби версіононування та стиснення. Для деяких задач вбудоване версіонування – надзвичайно корисна функція. Наприклад, зберігання історії версій сторінок вікі-сайта необхідно для виробітки політик і обслуговування. HBase позволяє не запобігати ніяких спеціальних мір для реализації історії сторінок – ми отримуємо це безкоштовно.

HBase задумана для масштабування по горизонталі.Якщо об'ем даних вимірюється гігабайтами чи терабайтами, то HBase – те,що потрібно. В HBase є механізми обліку конкретних серверних стоєк; реплікація даних між вузлами, встановленими в одній чи в різних стойках ЦОД, забезпечує швидке відпрацювання відмов без помітного зниження функціональності.

Слабкі сторони HBase

Хотя HBase проектувалась з урахуванням горизонтальної масштабованості, існує нижній поріг кількості вузлів. Для надійної работи необхідно не менше п'яти вузлів. Поскільки система призначена для обработки великих массивів даних, адмініструвати її досить важко. Для вирішення невеликих задач HBase навряд чи годиться, а документації для неспеціалістів практично не існує, що ускладнює її вивчення. Крім того, HBase майже ніколи не розгортається в ізоляції. Вона частина екосистеми масштабіруємих компонентів, до числа яких належить Hadoop (незалежна реаізація каркасу MapReduce, вперше запропонованого Google), розподілена файлова система Hadoop (HDFS). Один із найважливіших вкладів Google в інформатику – популяризація алгоритмичного каркасу mapreduce для параллельного виконання завдань на декількох вузлах. Він описаний в основоположній статті Google і став корисним інструментом для виконання запитів в цілому класі збереження даних, стійких до втрати зв'язаності (http://research.google.com/archive/mapreduce.html).

Zookeeper (служба баз головного серверу, координуюча роботу вузлів). У цієї

системи є як плюси, так і мінуси; вона забезпечує високу стабільність, але покладає на адміністратора груз відповідальності за обслуговування.

Слід відмітити, що HBase не дає ніяких засобів сортування чи індексування, крім ключів рядків. Рядки зберігаються відтсортованими по ключах, але ні по якому іншому полю: по імені і значенню стовпця сортування не проводиться. Тому для пошуку рядка по любому критерію, кром її ключа, прийдеться сканувати всю таблицю, або будувати і супроводжувати власний індекс. Відсутнє також поняття типу даних. Значення всіх полів в HBase трактуются як неінтерпретуємі масиви байтів. Нема ніякого розрізнення між цілим числом, рядком та датою. Для HBase все це просто байти, а їх інтерпретація покладається на додаток. .

HBase і теорема CAP

В термінології CAP HBase, безумовно, відноситься до класу CP. HBase дає строгі гарантії погоджувальності. Якщо один клієнт успішно записав якесь значення, то всі інші клієнти побачуть його при наступному запиті. Деякі бази даних, наприклад Riak, дозволяють змінити параметры CAP на рівні окремої операції. Але тільки не HBase. При не дуже серйозній втраті зв'язаності, наприклад, при виході з ладу одного вузла, HBase залишиться доступною – відповідальність за обслуговування запитів буде передана іншим вузлам. Але в патологічній ситуації, наприклад, коли працездатність зберіг лише один вузол, у HBase не буде іншого вибору, як відмовити в обслуговуванні запиту.

Обговорення властивостей CAP дещо ускладниться, якщо взяти до уваги міжкластерну реплікацію. В типовій багатокластерній конфігурації кластери можуть бути територіально разнесені. В такому випадку для любого заданого сімейства стовпців запись проводиться тільки на один кластер, а інші просто надають доступ до реплікованих даних. Така система являється погодженою в кінечному рахунку , поскільки репліковані кластери повертають останнє значення, про яке знають, а воно може і не співпадати з останнім значенням, записаним на головному кластері. Термінологія може здатися обманливо знайомою, а установка і налаштування – заняття не для слабких. З іншого боку, деякі можливості HBase, наприклад, версіонування і стискання, просто унікальні і можуть зробити HBase дуже приваблевим вибором для рішення деяких задач. Крім того, не можна не відмітити можливість горизонтального масштабування на вузли зі стандартним обладнанням. Взагалі, HBase серйозний інструмент, який вимагає відповідного поводження.

4. MongoDB

Перша публічна версія MongoDB була випущена в 2009 році. Система задумувалась як масштабіруєма база даних – назва Mongo походить від слова «humongous», отриманим об'єднанням «huge» (гігантський) і «monstrous», а в якості основних проектних цілей були поставлені висока працездатність і простота доступу до данх. Це документна база даних, яка забезпечує не тільки збереження, але і опитування вкладених даних, пред'являя довільні запити. Схема бази даних не нав'язується (в

цім MongoDB схожа на Riak, але відрізняється від Postgres), тому один документ може мати поля чи типи, відсутні у всіх інших документах колекції. Mongo – сховище JSON-документів (хотя, строго кажучи, дані зберігаються в двійковому варіанті JSON, який називається BSON). Документ Mongo можно уподобити рядку реляційної таблиці без схеми, в якій допускається довільна глибина вкладеності значень.

Сильні сторони Mongo

Головна перевага Mongo – здатність оброблювати гігантські масиви даних (і величезній потік запитів) за рахунок реплікації та горизонтального масштабування. Але вона також пропонує гнучку модель даних, так як не потрібно визначати схему, а дані можуть бути вкладеними (для досягнення аналогічного эффекту в РСУБД прийшлось би використовувати з'єднання). Накінець, при проектуванні MongoDB закладувалась простота використання. Існує схожість між командами Mongo та деякими концепціями баз даних на основі SQL. Це не випадково, і саме по цій причині Mongo має багато прибічників із кола бувших користувачів объектно-реляційних моделей (ORM).

Слабкі сторони Mongo

Те, що Mongo заохочує денормалізацію схеми (хоча самі схеми і відсутні), деяким може здатися неприйнятним. Є розробки, в яких жорсткі обмеження, які накладаються реляційною СУБД з її холодною логікою, вселяють впевненість. Можливість вставки в будь-яку колекцію значення довільного типу викликає побоювання. Єдина опечатка може стати причиною довготривалих мук, якщо Ви не здогадаєтеся шукати причину в іменах полів і колекцій. Гнучкість Mongo - не така вже велика перевага, якщо модель даних вже досить зріла і давно зафіксована. Оскільки Mongo орієнтована на великі набори даних, то використати її має сенс у великих кластерах, для проектування і обслуговування яких потрібно деякі зусилля.Навідміну від Riak, де додавання нових вузлів робиться прозоро і майже непомітно в процесі експлуатації, налаштування кластера Mongo має на увазі попереднє планування. Mongo - відмінний вибір для тих, хто вже звично використовує для зберігання даних реляційну базу даних і звертається до неї за допомогою якої-небудь системи об'єктно-реляційного відображення. ЇЇ часто рекомендують працюючим на платформах Rails, Django і взагалі усім, хто користується патерном модель-представлення-контролер (MVC), оскільки вони все одно реалізують перевірку даних і управління полями за допомогою моделей на рівні додатка і оскільки з міграцією схеми можна буде розпрощатися (здебільшого). Для додавання нових полів в документ досить просто додати поле в модель даних, і Mongo спокійно сприйме нові поняття. В порівнянні з реляційними базами даних Mongo дає набагато природніше рішення багатьох типових завдань, в яких набір даних визначається додатком.

5.CouchDB

CouchDB - типова документо-орієнтована база даних на основі JSON і REST. Вже перша версія, випущена в 2005 році, була спроектована з розрахунку на мережу веб і усі незліченні помилки, відмови і "глюки", які її супроводять. Тому по стабільності з CouchDB не може порівнятися майже жодна з існуючих баз даних. Якщо інші системи здатні пережити випадкові пропажі мережі, то CouchDB прекрасно себе почуває навіть тоді, коли поява мережі - рідкісна подія. Як і MongoDB, CouchDB зберігає документи - JSON-об'єкти, що складаються з пар ключ-значення, причому значеннями можуть бути дані різних типів, у рахунку і інші об'єкти з необмеженою вкладеністю. Проте довільні запити не підтримуються; основний спосіб пошуку документів - це індексовані представления, що породжуються інкрементною процедурою mapreduce. Ця база даних може масштабуватися вгору і вниз, а тому легко адаптується до завдань будь-якого розміру і складності. Вона розрахована не лише на велетенські кластери з дорогих серверів, але і на інші сценарії розгортання : від центру обробки даних до смартфону. Запустити CouchDB можна на телефоні Android, на персональному MacBook або в ЦОДі. CouchDB - документо-орієнтована база даних, в якій форматом зберігання і взаємодії із зовнішнім світом є JSON. Як і в Riak, усі звернення до CouchDB робляться за допомогою REST- інтерфейсу. Реплікація може бути як одно-, так і двосторонньою, за запитом або безперервною. CouchDB забезпечує високу гнучкість при ухваленні рішень про структуру, захист і розподіл даних.

Сильні сторони CouchDB

CouchDB - надійний і стабільний представник сімейства баз даних категорії NoSQL. Припускаючи, що мережі принципово ненадійні, а апаратні збої неминучі, CouchDB пропонує півністю децентралізований підхід до організації сховищ даних. Досить мала, щоб уміщатися в смартфоні, і досить велика для підтримки корпоративних рішень, CouchDB може бути розгорнута на самих різних платформах. CouchDB - в рівній мірі база даних і API. (Changes API – інтерфейс, позволяючий відслідковувати зміни в базі даних і миттєво отримувати нові значення). Крім канонічного проекту Apache CouchDB існують альтернативні реалізації і постачальники служб CouchDB, побудованих на базі гібридних серверних рішень, причому їх кількість постійно росте. Оскільки CouchDB з'явилася "з веб і для веб", то вона досить просто вбудовується як окремий шар у веб- технології - подібно балансувальникам навантаження і розподіленим кешам, - але при цьому зберігає унікальність своїх API.

Слабкі сторони CouchDB

Зрозуміло, CouchDB годиться не для будь-якого завдання. Основані на технології mapreduce представлення, при усій своїй новизні, не можуть забезпечити такі ж гнучкі засоби вибірки даних, як реляційні СУБД. Насправді, у виробничій системі взагалі не рекомендується прибігати до довільних запитів. Крім того, прийнята в CouchDB стратегія реплікації не завжди являється відповідним вибором. У CouchDB в основу реплікації покладений принцип "все або нічого", тобто на усіх реплікуємих серверах зберігаються одні і ті ж дані. Не існує механізму сегментації, що дозволяє розподілити дані по різних

серверах в ЦОД. Головна причина додавання нових вузлів в CouchDB - не стільки розподілити дані, скільки збільшити продуктивність операцій зчитування і запису. Прагнення CouchDB забезпечити надійність в умовах неоднозначності робить її відповідною системою для протистояння жорсткій реальності дикого світу Інтернету. Спираючись на стандартні веб-технології, зокрема HTTP/REST і JSON, CouchDB відмінно вбудовується у будь-яке рішення, де такі технології активно використовуються, тобто - з урахуванням сучасних тенденцій - скрізь. Але і у відгородженому від зовнішнього світу садку ЦОД CouchDB теж знайде застосування, якщо Ви готові вирішувати виникаючі конфлікти, або не проти альтернативної реалізації типу BigCouch, але не чекаєте знайти готовий механізм сегментації. У CouchDB є немало інших особливостей, що роблять її унікальною базою даних, що заслуговує на увагу. Не можливо розглянути їх всі, але хоч би перерахуємо деякі: простота резервного копіювання, двійкові вкладення в документи і CouchApps - система для розробки і розгортання веб-додатку прямо через CouchDB без додаткового ПЗ проміжного шару.

6. Neo4J

Neo4j - новий тип сховищ даних NoSQL, що отримав назву графова база даних. Як випливає з назви, дані в ній зберігаються у вигляді графу (у тому сенсі, в якому він розглядається в математиці). Відмітною особливістю таких баз даних є можливість малювати їх структуру на дошці у вигляді прямокутних блоків і ліній, що сполучають їх. Все, що можна зображувати у такому вигляді, можна зберегти в Neo4j. У Neo4j упор робиться швидше на зв'язку між значеннями, ніж на загальні характеристики значень (як в колекціях документів або в рядках таблиці). Таким чином, зовсім різнорідні дані можна зберігати просто і природньо. Розмір Neo4j невеликий - настільки, що її можна впровадити практично у будь-яке застосування. З іншого боку, в Neo4j можна зберігати десятки мільярдів вузлів і стільки ж ребер. А враховуючи підтримку реплікації на багато серверів, ця СУБД здатна впоратися із завданням будь-якого розміру. Neo4j - краща реалізація графської бази даних (досить рідкісний клас) з відкритим початковим кодом. У графових базах даних упор робиться на зв'язках між даними, а не на спільності значень. Моделювати дані у вигляді графів просто. Треба лише створити вузли і зв'язки між ними і за бажання асоціювати з ними пари ключ-значення. Запити зводяться до опису обходу графа, вирушаючи з початкового вузла.

Сильні сторони Neo4j

Neo4j - один з кращих зразків графових баз даних з відкритим початковим кодом. Такі бази прекрасно підходять для зберігання неструктурованих даних - іноді навіть краще, ніж документні сховища. Мало того що в Neo4j немає ні типів, ні схеми, вона ще і не накладає ніяких обмежень на взаємозв'язки між даними. Це "звалище" в хорошому значенні слова. Поточна версія Neo4j підтримує 34,4 мільярда вузлів і стільки ж зв'язків - більш ніж достатньо для більшості завдань (у одному графі Neo4j можна було б зберігати більше 42 вузлів для кожного з 800 мільйонів користувачів Facebook). У дистрибутив

Neo4j входять інструменти для прискорення пошуку (Lucene) і прості у використанні (хоча, можливо, не цілком звичні) мовні розширення, зокрема Gremlin і REST- интерфейс. Але Neo4j не лише проста у використанні, але і працює швидко. На відміну від операцій з'єднання в реляційних базах даних, або операцій mapreduce в інших базах, обхід графа вимагає постійного часу. Пов'язані дані знаходяться всього в одному кроці - не треба робити масивне з'єднання з подальшою фільтрацією результатів, як у більшості розглянутих вище СУБД. Яким би великим не був граф, перехід з вузла А у вузол В вимагає усього одного кроку, якщо між цими вузлами є зв'язок. Нарешті, у видання Enterprise включені засоби для створення високодоступних сайтів з великою кількістю запитів на читання - шляхом створення HA-кластеру Neo4j.

Слабкі сторони Neo4

Neo4j не позбавлена недоліків. Вибрана термінологія (вузол замість вершини і зв'язок замість ребра) тільки утрудняє взаєморозуміння. HA-кластер прекрасно справляється з реплікацією, але реплікувати дозволено тільки граф цілком. Неможливо сегментувати граф, що накладає обмеження на його розмір. Нарешті, якщо ви шукаєте продукт з відкритим початковим кодом і ліцензією, відповідною для корпоративного використання (наприклад, MIT), то Neo4j, мабуть, не для вас. Видання Community поставляється за ліцензією GPL, але якщо вам потрібні засоби, що входять тільки у видання Enterprise (високодоступний кластер і резервне копіювання), для побудови виробничої системи, то, ймовірно, за ліцензію доведеться заплатити.

Neo4j і теорема CAP

Для тих, хто збирається будувати розподілену систему, стратегія Neo4j має бути зрозуміла з фрази "високодоступний кластер". Neo4j має властивості доступності і стійкості до втрати зв'язаності. Кожен підпорядкований сервер повертає тільки ті дані, які має в розпорядженні в даний момент, і вони впродовж деякого періоду часу можуть відрізнятися від тих, що зберігаються в головному вузлі. Затримку оновлення можна скоротити, зменшивши інтервал між опитуваннями на підпорядкованому сервері, але технічно система все одно забезпечує тільки узгодженість. Тому HA-кластер Neo4j рекомендується використати тільки в додатках, орієнтованих здебільшого для читання. Простота Neo4j може бентежити, якщо Ви не звикли моделювати дані у вигляді графів. Незважаючи на потужний API з відкритим початковим кодом і роки промислової експлуатації, у цієї СУБД все ще відносно мало користувачів. Пояснюється це тільки недостатньою інформованністю, тому що графові бази даних природньо відбивають процес інтерпретації даних людиною. Ми представляємо сім'ї у вигляді дерев, друзів - у вигляді графів; мало хто розглядає зв'язки між людьми як самовідносні типи даних. Для деяких класів завдань, наприклад, соціальних мереж, вибір Neo4j напрошується сам собою. Але і в не таких очевидних випадках має сенс придивитися до цієї СУБД уважніше. .

7. Redis

Redis (REmote DIctionary Service - віддалена служба словників) - просте в роботі сховище ключів і значень з розвиненим набором команд. Перша версія системи вийшла в 2009 році. І в швидкодії у неї практично немає суперників. Читання робиться швидко, а запис ще швидший - на деяких еталонних тестах продемонстрировано до 100 000 операцій SET в секунду. Творець Redis Сальваторе Санфилиппо (Salvatore Sanfilippo) називає свій проект "Сервером структур даних", щоб підкреслити закладені в нього можливості роботи із складними типами даних і інші особливості. Вивченням цього надшвидкого "не просто сховища ключів і значень" ми і завершимо огляд сучасного ландшафту баз даних. Redis подтримує складні структури даних, хоча і не до такої міри, як документо-орієнтовані бази даних. Вона підтримує запити, що повертають множини, але не на такому рівні детальності, як реляційні СУБД, і без підтримки типів. І, зрозуміло, вона швидка, хоча для досягнення цієї мети приносить довговічність даних в жертву швидкодії. Окрім серверу структур даних, Redis є ще і блокуючою чергою (чи стеком), а також системою публікації-підписки. У ній можна настроювати політики терміну зберігання, рівні довговічності і параметри реплікації. Усе це робить Redis швидше комплектом інструментальних засобів, в який входять корисні алгоритми роботи із структурами даних і процеси, ніж базою даних якогось певного жанру. Великий список клієнтських бібліотек для Redis дозволяє використати її у багатьох мовах програмування. Робота з нею не лише проста, але і приносить задоволення.

Резюме

Redis - компактне сховище ключів і значень (чи структур даних), споживаюче мало ресурсів. Воно схоже на багатофункціональні інструменти. Як і вони, Redis придатна для вирішення самих різних, часто несподіваних завдань. До усього іншого, сховище Redis працює швидко, просте в експлуатації і забезпечує таку довговічність даних, яку Ви налаштуєте. Redis рідко використовується як автономна база даних, частіше вона входить до складу багатосторонньої екосистеми, де застосовується для трансформації даних, кешування запитів, або управління чергами повідомлень (завдяки вбудованим блокуючим командам). .

Сильні сторони Redis

Очевидна цінність Redis - швидкодія, чим можуть похвалитися багато подібних сховищ ключів і значень. Але, на відміну від більшості таких сховищ, Redis ще уміє зберігати складені значення - списки, хеши і множини - і застосовувати до них спеціальні операції. Мало того, в Redis ще можна тонко настроювати довговічність даних, віддаючи перевагу або надійності зберігання, або швидкості. Вбудована реплікація типу головний-підлеглий - ще один зручний спосіб підвищити довговічність, не приносячи швидкодію в жертву повільного дозапису у файл при кожній операції До того ж, реплікація підвищує продуктивність систем, в яких основне навантаження доводиться на операції зчитування.

Слабкі сторони Redis

Своєю швидкодією Redis зобов'язана, передусім, тому, що зберігає усі дані в пам'яті. Хтось назве це обманом, тому що база даних, яка взагалі не звертається до диску, природньо, працюватиме швидко. Будь-якому сховищу в оперативній пам'яті властива проблема довговічності даних; якщо зупинити базу, не скинувши дані на диск, то дані будуть втрачені. Навіть якщо встановити синхронний режим дозапису у файл при кожній операції, все одно при відтворенні файлу відновлення даних з простроченим терміном зберігання не цілком надійно - це характерно для будь-якого механізму відтворення подій, прив'язаних до часу. Втім, ця проблема швидше теоретична, ніж практична. Redis також не підтримує набори даних, розмір яких перевищує об'єм доступної оперативної пам'яті (підтримка віртуальної пам'яті з Redis буде виключена). Розроблявся кластер Redis, який дозволить вийти за межі обмеження на об'єм ОЗУ в одному комп'ютері, але назараз я не маю достовірної інформації про результат. Доки користувачі, що бажають організувати кластер, повинні самі написати відповідний клієнт (як драйвер на Ruby).

Redis не відчуває нестачі в командах - всього їх більше 120. Назви більшості команд пояснюють їх призначення, треба лише звикнути до думки, що деякі надмірні букви опущені (як, наприклад, в команді INCRBY). Redis вже став невід'ємною частиною багатьох систем. На його основі створені декілька проектів з відкритим початковим кодом - від Resque (написана на Ruby служба асинхронних черг завдань) до SocketStream (система управління сеансами для Node.js).

Висновки

За способом зберігання бази даних можна приблизно розбити на п'ять жанрів: реляційні, сховища ключів і значень, стовпцеві, документні і графські. Чим вони відрізняються один від одного і для яких завдань підходять або не підходять - коли їх має сенс **використати**, а коли треба триматися **чимдалі**:

Реляційні бази даних

Це класичний і найпоширеніший варіант. Реляційні системи управління базами даних (РСУБД) грунтовані на теоріях множин, дані в них зберігаються у вигляді двовимірних таблиць, які складаються з рядків і стовпців. Реляційні бази строго контролюють типи даних і зазвичай допускають зберігання чисел, рядків, дат і двійкових об'єктів (BLOB'ов), що не інтерпретуються, хоча, PostgreSQL підтримує такі розширення, як масив і куб.

Хороші для:

Завдяки структурованій природі даних реляційні бази має сенс використати, коли структура даних заздалегідь відома, а способи їх можливого використання - ні. Іншими словами, Ви готові авансом заплатити за складність організації, сподіваючись, що в майбутньому це окупиться гнучкістю запитів. Багато завдань бізнесу зручно моделюються саме так: замовлення, постачання, облік складських запасів, кошики закупівель і інші. Заздалегідь невідомо, як згодом

потрібно буде витягати дані (наприклад, скільки замовлень ми обробили в цьому місяці?), але дані за своєю природою регулярні і контролювати цю регулярність корисно.

***Не дуже добрі для**:*

Якщо структура даних змінна або характеризується глибокою вкладеністю, то реляційна СУБД - не кращий помічник. У ній необхідно заздалегідь визначити схему, що дуже важко зробити, якщо усі записи структурно відрізняються один від одного. Подумайте, як би Ви стали розробляти базу даних для опису усіх живих істот в природі. Спроба створити повний список ознак ("має волосся", "кількість ніг", "відкладає яйця" і т. д.) приречена на провал. У такому разі потрібна база даних, яка не накладає таких строгих обмежень на дані, які зберігаються.

Сховища ключів і значень

Сховище ключів і значень - проста з усіх розглянутих моделей. У нім прості ключі відображаються на, можливо, складніші значення, як у велетенській хеш-таблиці. Завдяки відносній простоті бази даних цього жанру забезпечують максимальну гнучкість реалізації. Пошук в хеш-таблиці здійснюється швидко, тому у разі Redis швидкодія була поставлена основною задачею. Крім того, пошук в хеш-таблицях легко розподіляється на декілька машин, і в Riak цей факт використаний для побудови простих в управлінні кластерів. Зрозуміло, простота може виявитися мінусом, якщо до моделювання даних пред'являються складні вимоги.

Хороші для:

Оскільки необхідність підтримувати індекси відсутня або невелика, то при проектуванні сховищ ключів і значень часто закладається горизонтальна масштабованість, дуже висока швидкодія, або те і інше разом. Особливо хороші вони для задач, де між даними немає великої кількості зв'язків. Наприклад, у веб-застосуванні цьому критерію відповідають сеанси користувачів; дії одного користувача в сеансі практично ніяк не пов'язані з діями інших користувачів.

Не дуже добрі для:

Через відсутність індексів і засобів сканування КЗ-сховища мало чим допоможуть, якщо вимагається пред'являти запити, відмінні від простих операцій CRUD (створення, читання, оновлення, видалення).

Стовпцеві базы даних

Стовпцеві бази даних (також бази даних з сімействами стовпців) мають багато загальних рис з КЗ-сховищами і РСУБД. Як і в КЗ-сховищах, значення просяться по ключу. Як і в реляційних базах, значення групуються в нуль або більше за стовпці, хоча в кожному рядку може бути заповнене довільне число стовпців. Але на відміну від того і іншого, в стовпцевих базах дані зберігаються

по стовпцях, а не по рядках. Додавання нових стовпців обходиться дешево, версіонування реалізується тривіально, і на зберігання відсутніх значень місце не витрачається. Розглянута нами база даних HBase - класичний приклад цього жанру.

Хороші для:

Столвцеві бази даних традиційно розроблялися з метою забезпечити горизонтальну масштабованість. Тому вони особливо хороші для завдань, пов'язаних з "великими даними", коли база розміщується в кластері з десятків, сотень, а то і тисяч вузлів. Крім того, в них зазвичай вбудовується підтримка стискування і версіонування. Канонічний приклад завдання, орієнтованого на зберігання даних по стовпцях, - індексування веб-сторінок. Сторінки у веб містять багато тексту (і тут дуже корисне стискування), в якійсь мірі взаємозв'язані і змінюються з часом (тут на допомогу приходить версіонування).

Не дуже добрі для:

У різних стовпцевих баз даних різні можливості і, відповідно , різні недоліки. Але усім їм властива одна загальна риса: краще проектувати схему з урахуванням майбутніх запитів. Це означає, що треба апріорі уявляти собі, як використовуватимуться дані, а не тільки, що вони містять. Якщо способи використання даних заздалегідь невідомі - наприклад, потрібні довільні звіти, - то стовпцева база даних - не оптимальний варіант.

Документні бази данных

У документній базі даних об'єкт, що зберігається, може містити довільний набір полів і навіть допускає вкладеність будь-якого рівня. Зазвичай такі об'єкти представляються в нотації JavaScript Object Notation (JSON), прийнятій як в MongoDB, так і в CouchDB, хоча ця вимога аж ніяк не є концептуально значимою. Оскільки документи не пов'язані один з одним так тісно, як в реляційних базах даних, їх порівняно просто сегментувати і реплікувати на декілька серверів, тому часто зустрічаються розподілені реалізації. Система MongoHQ ставить метою підвищити доступність за рахунок створення ЦОД, здатних підтримувати гігантскі набори даних масштабу веб. З іншого боку, CouchDB орієнтована на простоту і довговічність даних, а доступність досягається за рахунок реплікації типу головний-головний на вузли з високим рівнем автономності. Ці проекти значною мірою перекриваються.

Хороші для:

Документні бази даних хороші для завдань, де предметна область характеризується високою мірою мінливості. Якщо зарані невідомо, як виглядають дані, то документна база - якраз те, що треба. Крім того, документи по самій своїй природі часто добре лягають на об'єктно-орієнтовані моделі програмування. А, значить, зменшується розузгодження інтерфейсів при переході від моделі бази даних до об'єктної моделі додатку.

Не дуже добрі для:

Якщо ви звикли до складних запитів із з'єднаннями в реляційній базі даних з добре нормалізованою схемою, то в документній базі цих можливостей бракуватиме. Документ зазвичай містить усю або велику частину інформації про об'єкт. Якщо в реляційній базі природнє прагнення нормалізувати дані з метою усунення дублювання, яке може привести до розузгодження, то в документних базах денормалізоване зберігання – це норма.

Графові бази даних

Графові бази даних - новий, швидкорозвиваємий клас систем, в яких упор робиться швидше на встановлення довільних зв'язків між даними, ніж на фактичні значення. Прикладом може служити проект з відкритим початковим кодом Neo4j, який все частіше використовується у багатьох соціальних мережах. На відміну від інших стилів баз даних, де колекції схожих об'єктів групуються в загальних контейнерах, в графових базах дані зберігаються у вільнішій формі; обробка запиту зводиться до дотримання по ребрах, що сполучають дві вершини, інакше кажучи, до обходу вузлів. У міру використання у все більшому числі проектів графові бази даних починають застосовуватися не лише для побудови простих соціальних застосувань, але і для більш специфічних цілей, наприклад: системи рекомендації, списки управління доступом і географічні дані.

Хороші для:

Графовіі бази даних неначе спеціально придумані для додатків, які характеризуються наявністю мережі даних. Типовий приклад - соціальна мережа, де вузли представляють користувачів, між якими існують різні зв'язки. Моделювання таких даних за допомогою інших засобів частенько є складною проблемою, але графові бази даних справляються з цим завданням на ура. До того ж, вони прекрасно лягають на об'єктно-орієнтовану систему. Все, що можна змоделювати на дошці, можна змоделювати і за допомогою графа.

Не дуже добрі для:

Із-за наявності великї кількості зв'язків між вузлами графові бази даних як правило погано пристосовані до розміщення на декількох машинах. Швидкий обхід графа був би неможливий, якби довелося відправляти по мережі запити іншим вузлам, тому графові бази даних масштабуються погано. Швидше за все, графова база буде лише однією з частин вашої системи, в якій зберігаються тільки зв'язки, тоді як основні дані знаходяться у іншому місці. Визначитися з вибором бази даних нерідко виявляється складніше, ніж просто вирішити який жанр краще всього підходить для даних з конкретної предметної області. Очевидно, що соціальний граф найпростіше зберігати в графовій базі даних, але якщо йдеться про Facebook, то даних надто багато - в такій базі вони просто не помістяться. Швидше, ви зупинитеся на якій-небудь системі, пристосованій до "великих даних", наприклад, HBase або Riak. Таким чином, ви будете вимушені вибрати стовпцеву базу або сховище ключів і значень. Чи узяти інший приклад: реляційна база даних начебто ідеально підходить для реалізації банківських

транзакцій, але ж і Neo4j підтримує ACID- транзакції, так що вибір є. Ці приклади показують, що при виборі бази даних - або декількох баз даних, - оптимальних для предметної області, слід брати до уваги не лише жанр. Взагалі кажучи, у міру зростання об'єму даних, гранична місткість деяких баз стає обмеженням. Стопцеві бази даних часто розподіляються по декількох ЦОД і підтримують "великі дані" максимального розміру, тоді як місткість графових баз найменша з усіх. Взагалі, це правило не універсальне. Так, Riak - великомасштабне сховище ключів і значень, призначене для сегментації на сотні і тисячі вузлів, тоді як Redis проектувалася для роботи в одному вузлі, але з можливістю реплікації типу головний-підлеглий і сегментації, реалізованої на рівні клієнта. При виборі бази даних треба враховувати і такі характеристики, як довговічність, доступність, узгодженість, масштабованість і безпека. Ви повинні вирішити, чи потрібна можливість пред'являти довільні запити, або буде досить технології mapreduce. Ви віддаєте перевагу HTTP/REST-інтерфейсу, або потребуєте драйверу для спеціалізованого двійкового протоколу? Навіть такі, на перший погляд, другорядні питання, як існування програми масового завантаження даних, можуть виявитися важливими в конкретному випадку.

Питання для самоконтролю

1. Охарактеризуйте роль і значення баз даних для Grid і хмарних обчислень
2. Перерахуйте особливості використання баз даних в розподілених обчисленях
3. Що таке розподілена база даних?
4. З чого складається система керування розподіленими базами даних ?
5. Що таке гомогенні та гетерогенні розподілені СКБД?
6. Що таке фрагментація, розподіл, реплікація?
7. Охарактеризуйте централізоване розміщення, фрагментоване розміщення
8. Що таке розміщення з повною реплікацією?
9. Що таке розміщення з вибірковою реплікацією?
10. Що таке розподілена обробка?
11. Що таке паралельна СКБД?
12. Охарактеризуйте переваги, недоліки і властивості СКБД.
13. Що лежить в основі реалізації реляційних СУБД ?
14. СУБД PostgreSQL – переваги і недоліки
15. Riak – розподілене сховище ключів і значень – переваги і недоліки
16. HBase – стовпчикова база даних – переваги і недоліки
17. MongoDB – переваги і недоліки
18. CouchDB - документо-орієнтована БД на основі JSON і REST– переваги і недоліки
19. Neo4j - тип сховищ даних NoSQL, отримав назву графова база- переваги і недоліки
20. Redis (віддалена служба словників) - сховище ключів і значень з розвиненим набором команд – переваги і недоліки.

Література

1. Эрик Редмонд, Джим. Р. Уилсон Введение в современные базы данных и идеологию NoSQL – М.: ДМК Пресс, 2013. – 384с.
2. http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks
3. http://allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
4. http://www.erlang.org/
5. http://ruby-lang.org
6. http://rubygems.org
7. http://rubygems.org/gems/riak-client
8. http://research.google.com/archive/mapreduce.html
9. http://wiki.basho.com/Replication.html
10. http://code.google.com/p/leveldb/
11. http://www.mongodb.org/downloads
12. http://rdoc.info/github/couchrest/couchrest/master/
13. http://download.freebase.com/datadumps/latest/browse/film/performance.tsv
14. http://neo4j.org/download/
15. http://redis.io

Розділ 2. Грід-системи та технології

2.1. Введення в інформаційні технології та Грід

Аналізуючи історію розвитку інформаційних технологій та сучасні тенденції можна зробити висновок, що еволюційний виток IT, який почався разом з епохою великих обчислювальних машин (ЕОМ) понад п'ятдесят років тому, замкнувся – разом з хмарами ми повернулися до централізації ресурсів, але на цей раз не на рівні мейнфреймів, а на новому технологічному рівні.

Виступаючи на конференції, присвяченій проблемам сучасних процесорів, професор Массачусетського технологічного інституту Ананд Агарвал сказав: "Процесор – це транзистор сучасності. Новий рівень відрізняється тим, що тут також збираються мейнфрейми, але віртуальні, і не з окремих транзисторів, як півстоліття тому, а з цілих процесорів, або цілком з комп'ютерів. На зорі IT численні компанії та організації створювали власні комп'ютери з дискретних компонентів, монтуючи їх на саморобних друкованих платах - кожна організація робила свою машину, і ні про яку стандартизацію чи уніфікацію мови не могло бути. Зараз ситуація повторюється – точно так саме: з серверів, комп'ютерів, різноманітного мережевого обладнання збираються зовнішні та приватні хмари. Одночасно спостерігається та ж сама технологічна роз'єднаність і відсутність уніфікації: Microsoft, Google, IBM, Aptana, Heroku, Rackspace, Ning, Salesforce будують глобальні мейнфрейми, а хтось під власні

потреби створює приватні хмари. Залишається припустити, що попереду винахід інтегральної схеми і мікропроцесора".

Трохи історії.

Перша інтегральна мікросхема була сворена в вересні 1958р.

Через 6 років в 1965р один із засновників компанії ***Інтел*** Гордон Мур підмітив тенденцію, згідно якої кількість транзисторів на кристалі подвоювалась приблизно через рік, на основі чого був сформульований так званий закон Мура. В 1975р Мур підкоректував свій закон, по якому кількість транзисторів на чіпі інтегральної мікросхеми подвоюється через 2 роки – потім ***Інтел*** скорегував цей термін на 18 місяців.

Закон імперичний. Зараз він перестав виконуватись. Чому?

Перші мікросхеми створювались на технологічних нормах в декілька мікрометрів, микропроцесори Пентіум створювалися на технологічних нормах 0,6 мкм, а зараз технологічні норми досягли 10 нанометрів. Технологічні норми – це мінімальна ширина токопровідної доріжки в планарній інтегральній мікросхемі, яка технологічно освоєна в серійному виробництві. Досягнуті норми такі значні, що вже назріла криза в розвитку комп'ютерів із-за атомарної природи твердого тіла. Подолання цієї кризи бачать в квантових обчисленнях, на що вже зараз татять величезні сили і ресурси.

Інформатизація сьогодні вступила в четвертий етап свого розвитку. Перший був пов'язаний з появою великих комп'ютерів, другий - зі створенням персональних комп'ютерів, третій - з появою Інтернету. Четвертий етап інформатизації включає ряд нових технологій.

Інтернет, WWW - World Wide Web, Grid і хмарні обчислення - пов'язані між собою, але різні технології:

• ***Інтернет*** - це глобальна мережа, що об'єднує безліч комп'ютерів і локальних (порівняно невеликих) мереж і дозволяє їм взаємодіяти один з одним.

• ***Веб*** (паутина) - це спосіб доступу до інформації, яка знаходиться на віддаленому, але включеному в Інтернет комп'ютері.

• ***Web-служби (Web Services)*** - це віддалені сервісні об'єкти, які реалізують за запитом користувача деяку функціональність.

Історично першими з'явилися великі комп'ютери, які в західній термінології називають ***"мейнфрейми"***, у них відсутні обмеження на обсяг обладнання, тому їх називають ще великими ЕОМ. Це відбулося в 1964 році, коли компанія IBM створила перший комп'ютер сімейства IBM-360, які були багато років присутні на ринку ЕОМ. Слід зауважити, що в той час в Україні була створена Інститутом Кібернетики ЕОМ "Дніпро-2" - повний функціональний аналог IBM-360: 64-розрядна ЕОМ з 128-розрядною "плаваючою" арифметикою, оперативною пам'яттю до 65536 байт і повним набором периферійних пристроїв (пристрій паралельного друку АЦПУ-128, пристрій послідовного друку, пристрої вводу і виводу інформації на перфострічку, пристрої вводу та виводу на перфокарту, зовнішня пам'ять на магнітній стрічці в складі 8-ми накопичувачів, екранний пульт: алфавітно-цифровий та графічний та ін.). Ця ЕОМ ("Дніпро-2") серійно вироблялась в Києві до 1971р. Незважаючи на це була прийнята постанова керівництва країни про припинення виробництва вітчизняної розробки і створення радянського

повного клону IBM-360. Головний конструктор «Дніпра-2» ак.Глушков В.М. не зміг переконати керівництво тодішнього СРСР, що це змусить нас поступитися предовими позиціями і перейти в розряд відстачих. Так вітчизняна IT-індустрія перетворилася з передової в «вічнодоганяючу». За короткий час після постанови про створення вітчизняного клону IBM-360 досить швидко було розроблене сімейство-аналог**:** ЕС ЕОМ та ЕОМ АСВТ (ЕОМ агрегатної системи засобів обчислювальної техніки). В 1969р в Києві була створена, а в 1970 році в Україні почалося серійне виробництво повного аналога IBM-360 ЕОМ АСВТ М-3000.

СуперЕОМ - це продовження великих ЕОМ в бік збільшення числа АЛУ, процесорів, цілих машин, об'єднаних в одну систему для виконання паралельних обчислень в рамках однієї задачі. Такі ЕОМ включають сотні і тисячі процесорів, розташованих локально, так що ефективність визначається простим законом Амдала.

Можливості централізованого обчислення обмежені, оскільки такі системи за своєю природою обмежені в обладнанні, тому для великих задач більший інтерес представляють розподілені системи, сумарна потужність яких за визначенням перевершує потужність будь-якої централізованої системи. Коротко розглянемо засоби для організації таких разподілених систем.

Інтернет

Інтернет є єдиним інформаційним простором, в якому можна побудувати різні конструкції - сайти, хости, сервери і т.д. Історія Інтернету почалася з 1958 року, коли в відповідь на запуск першого супутника, США створили організацію під назвою ARPA (Advanced Research Projects Agency). До 68-го року в ARPA і в інших організаціях велася робота по з'єднанню комп'ютерів у мережі. В 67-му році пройшов симпозіум трьох незалежних розробників комп'ютерних мереж: ARPA, NPL, RAND. В 1968-м році була побудована перша мережа, заснована на сучасних принципах Інтернету.
Вона складалася з 4-х комп’ютерів. Слід зазначити, що тоді ж (в 1968р) на заводі обчислювальних машин в м.Києві (вул. Велика Окружна 4) була створена автоматизована система управління АСУ підприємства в складі 4-х ЕОМ «Дніпро 2», які були розміщені в різних приміщеннях заводу і об’єднані швидкісною локальною мережею.

Впродовж наступних десяти років в мережі ARPANET зв'язували багато організацій та університетів. До 1978-го року були розроблені всі базові протоколи (тобто мови спілкування між комп'ютерами), які і зараз використовуються в Інтернеті. Для того, щоб комп'ютери різних моделей могли спілкуватися в Інтернеті, потрібно було розробляти єдині протоколи. Крім того, потрібно було встановити спосіб розмежування одного комп'ютера в мережі від іншого, а також - спосіб визначення їх місця розташування.

Цій цілі служать IP адреси (адреси Інтернет-протоколу). IP адреси - це чотири числа від 0 до 255. Зазвичай їх розділяють крапками. Наприклад: 123.23.110.1. Однак такий спосіб добре підходить тільки для спілкування самих комп'ютерів, але дуже незручний для людини. Для зручності ж людини служать домени. Домен - це більш простий і зрозумілий спосіб ідентифікувати

комп'ютери в Інтернеті. Наприклад, комп'ютер з IP-адресою 209.155.82.19 має доменне ім'я www.cdrom.com.

Домени мають кілька рівнів. Наприклад, в доменном імені www.cdrom.com, "com" - домен першого рівня, "cdrom" - другого рівня, "www" - третього рівня. Тот, хто володіє доменом необмеженого рівня, може створювати скільки-небудь бажаних доменів нижчих рівнів. Домени першого рівня не підлягають продажі. Вони визначаються організацією по розвитку Інтернету.

Загальноприйнято, що за протоколом http спілкуються по порту 80. І ваш браузер, і програма, яка обслуговує порт 80, вміють говорити на протоколі http. HTTP означає Hyper Text Transfer Protocol, або Протокол передачі гіпертексту. Гіпертекст - це текст, в якому є посилання на інші гіпертексти або місця в цьому тексті. При відправці пошти, Ваша поштова програма використовує протокол 'SMTP', який використовує порт 25, а при отриманні - протокол POP3 на порту 110. Є ще величезна кількість різних протоколів і портів. Їх для простоти можна називати сервісами, а комп'ютери, які очікують, що до них прийдуть запити на використовуваних протоколах, - серверами.

WWW

WWW - є спроба організувати всю інформацію в Internet, а також будь-яку локальну інформацію за вашим вибором як набір гіпертекстових документів. Ви переміщується по мережі, переходячи від одного документа до іншого по посиланнях. Всі ці документи написані на спеціально розробленій для цього мові, що називається HyperText Markup Language. World Wide Web, Всесвітня павутина, WWW, Web, Інтернет, - це все назви одного й того ж сервісу, який був придуманий в 1991 році і використовує протокол HTTP для передачі гіпертекстових документів та інших файлів від Веб- сервера до клієнтів. Головна відмінність WWW від інших інструментів для роботи з Internet полягає в тому, що WWW дозволяє працювати практично з усіма доступними зараз на комп'ютері видами документів: це можуть бути текстові файли, ілюстрації, звукові і відеоролики і т.д. Принцип роботи WWW наступний. Користувач запускає у себе програму, яка розуміє протокол HTTP і спеціальну мову, на якій створюється вміст WWW. Ця програма називається «програмою перегляду HTML-сторінки», або по-англійськи - browser (браузер). При цьому HTML (Hyper Text Markup Language) - це «мова розмітки гіпертексту», або мова, за допомогою якої створюється гіпертекст. Далі користувач набирає адресу www сервера. Браузер звертається до сервера з проханням віддати документ, розташований за цією адресою. Сервер віддає документ. Браузер отримує документ, обробляє його і, якщо в ньому є картинки, також просить сервер віддати йому їх, як і інші матеріали документу. Цей документ прийнято називати сторінкою, або WEB (Веб) -сторінкою, або HTML- сторінкою. Після цього браузер обробляє всі прийняті дані і показує готову сторінку на Вашому екрані. Деякі елементи сторінки (тест, картинки, кнопки) можуть бути посиланнями. Якщо Ви натиснете на них, то Ваш браузер пошле запит серверу, щоб попросити у нього документ.

Таким чином, Ви можете пересуватися від документа до документа, від сервера до сервера, що перетворює весь Інтернет в одну гігантську Мережу, яка пов'язує документи і сервери один з одним нитками гіперпосилань.

WWW - система в цілому складається з наступних компонент:

• Мова гіпертекстової розмітки HTML
• Протокол передачі гіпертексту HTTP
• Специфікацій на типи даних в Internet (Internet Media Types)
• Системи WWW-адресації (URL, URN, URI etc.).

Мова HTML, як уже згадувалося раніше, дуже проста. З чисто з практичної точки зору HTML є розмітка, зроблена звичайними англійськими словами усередині документа. HTML був розроблений для того, щоб виділити в документах логічну структуру. Абревіатура URL розшифровується як Uniform Resource Locator, що можна перекласти, як "єдиний покажчик на ресурс". Практично, це адреса документа.

Web service

Спочатку визначимо, що таке Web-служба. Web-сервіс - це серверний об'єкт, який реалізує певний елемент функціональності, з яким можуть взаємодіяти віддалені програми по протоколу HTTP за допомогою повідомлень на мові XML.

Технологія Web Services призначена для створення розподілених додатків, що функціонують в гетерогенному (неоднорідною) середовищі Інтернет , компоненти яких взаємодіють на базі стандартних Web - протоколів. Архітектура мережі Web Services і взаємодія між клієнтами і службами представлені на рис.2.1.1. Архітектура Web-служб передбачає слабку зв'язність між компонентами мережі (loose coupling). Слабка зв'язність означає, зокрема, що компонентам системи зовсім не обов'язково знати, як влаштовані взаємодіючі з ними підсистеми, а для взаємодії немає необхідності у визначенні нових форматів даних та створенні спеціального програмного забезпечення. Принцип слабкої зв'язності далеко не новий. Вважається, що саме слабка зв'язність дозволила web-технологіям в рекордно короткі терміни стати надзвичайно популярними. Як вже було сказано, Web Services базуються на застосуванні відкритих стандартах і протоколах, які затверджуються консорціумом IT-спільноти, ключовими з яких є наступні:

1. SOAP (Simple Object Access Protocol) - протокол доступу до простих об'єктів, тобто механізм для передачі інформації між об'єктами на базі протоколу HTTP і деяких інших Інтернет-протоколів;
2. WSDL (Web Services Description Language) - мова опису Web-сервісів;
3. UDDI (Universal Description, Discovery and Integration) - універсальне опис, - спрощено кажучи, протокол пошуку ресурсів в Інтернеті.

Основою для реалізації всіх цих протоколів є мова XML (EXtensible Markup Language).

З урахуванням згаданих стандартів алгоритм опису деякого завдання в системі, представленої на рис.2.1.1, можна описати таким чином. Клієнт спочатку звертається на UDDI до реєстру ресурсів (лінія 1), щоб за заданим

ним признаком необхідного ресурсу отримати відомості про наявність потрібного ресурсу і місце його розташування (лінія 2). Такий запит ми зазвичай робимо в будь-якій пошуковій системі, наприклад, Google. Отримана з реєстру відповідь містить один або кілька адрес ресурсів. Клієнт по одному з отриманих адрес звертається до ресурсу (лінія 3), щоб отримати WSDL на інтерфейс ресурсу (лінія 4). Інтерфейсом називається видима користувачу частина ресурсу (взагалі, інтерфейс - це набір правил і домовленостей, за якими функціонують обчислювальні засоби). В інтерфейсі вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт SOAP передає ресурсу завдання (лінія 5) і через деякий час отримує результат. У загальному випадку завдання може виконуватися не одним, а безліччю ресурсів.

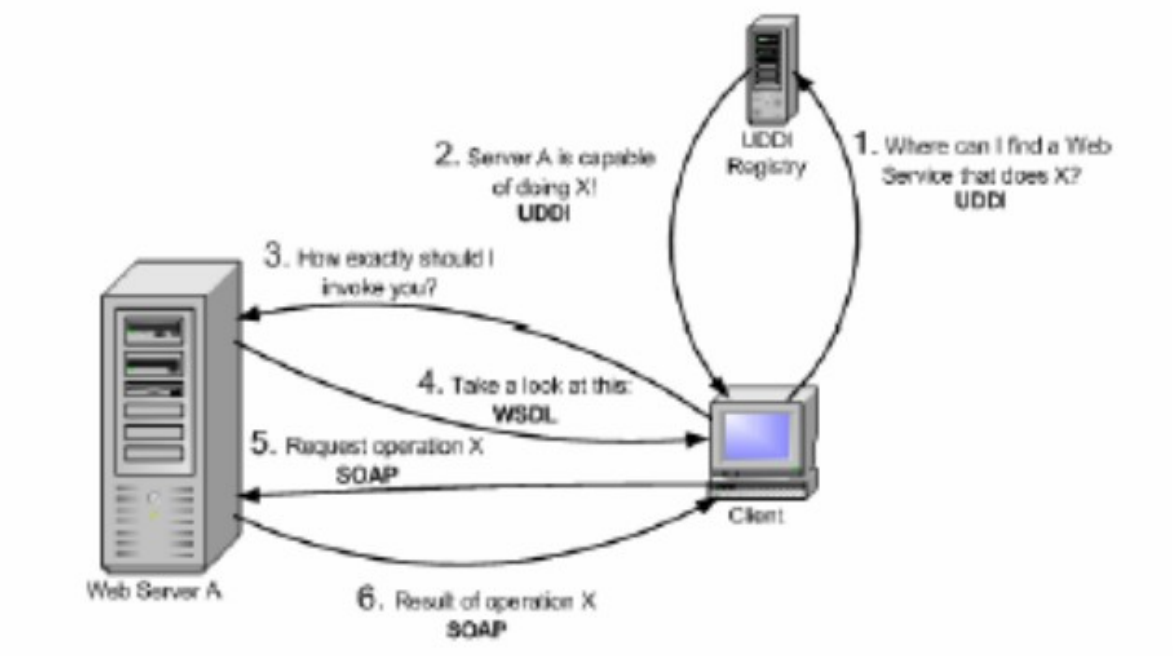


Рис. 2.1.1 Архітектура Web Services

Мова XML

XML (EXtensible Markup Language) - це в перекладі "розширювана мова розмітки". HTML і XML створювалися з різними цілями:
• HTML створювався для демонстрації даних і фокусується на тому, як дані виглядають.
• XML створювався для опису даних і фокусується на тому, чим є дані. Ще потрібно написати якесь програмне забезпечення, щоб відправити цю записку, отримати її або показати її на екрані. На відміну від HTML, XML-теги ідентифікують дані (вказує тип даних), а не спосіб їх відображення. Якщо HTML-тег вказує, наприклад, "відобразити ці дані жирним шрифтом" (<b> ... </ b>), XML-тег діє як ім'я поля у вашій програмі. Він ставить мітку на частину даних, які ідентифікує (наприклад: <message> ... </ message>). Ніякої інформації про структуру, тільки теги візуального відображення, мінімум логічної розмітки. Теги заголовків попередньо описуються, можна описати і форматування абзаців. Але що ще краще, різним записам можна задати унікальні стильові ідентифікатори, якими в подальшому можна маніпулювати.

Наприклад, зміна атрибутів виведення конкретного стилю призведе до відповідних змін у всіх документах сайту.

Грід

Під англійським терміном GRID (решітка) розуміється сукупність просторово розподілених обчислювальних вузлів, пов'язаних деякою мережею для обміну даними. Надалі замість GRID будемо використовувати слово Грід. Web Service дозволяє клієнту виконати на обладнанні власника ресурсу деяку функцію зі списку, складеного власником цього обладнання. Грід принципово відрізняється від Web Services.

Грід – це метод використання глобальних процесорних потужностей і систем зберігання інформації (дискові системи великої місткості) без їх фізичного переміщення в просторі. Природно, Грід включає як ресурси і все напрацьоване в WebServices. . Більш того, протоколи WebServices (WSDL, SOAP, UDDI), засоби адресації в розширеному варіанті є основними протоколами Грід.

Грід характеризується наступним чином:
• Географія ресурсів глобальна
• Ресурси неоднорідні, відрізняються по апаратурі, операційним системам, протоколам функціонування і зв'язку
• Склад ресурсів змінний, в загальному випадку непередбачуваний
• Елементи середовища можуть бути ворожими, тобто можуть пошкодити або захопити закриті дані.

Існують наступні два основні різновиди Грід:

• Комунальний Грід. Такий Грід дозволяє надавати на запит клієнта відповідний ресурс.
• Метакомпьютер. В цьому випадку об'єднуються ресурси ряду корпорацій, можливо - весь Інтернет для вирішення загальнолюдських завдань. Структура Грід показана на рис.2.1.2. Тут клієнт спочатку звертається до реєстру ресурсів MDS, щоб отримати по заданих ним ознакам відомості про наявність потрібного ресурсу і місце його розташування. Клієнт по одній з отриманих адрес звертається до ресурсу, щоб отримати інтерфейс ресурсу. В інтерфейсі вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт передає ресурсу завдання і через деякий час отримує результат.

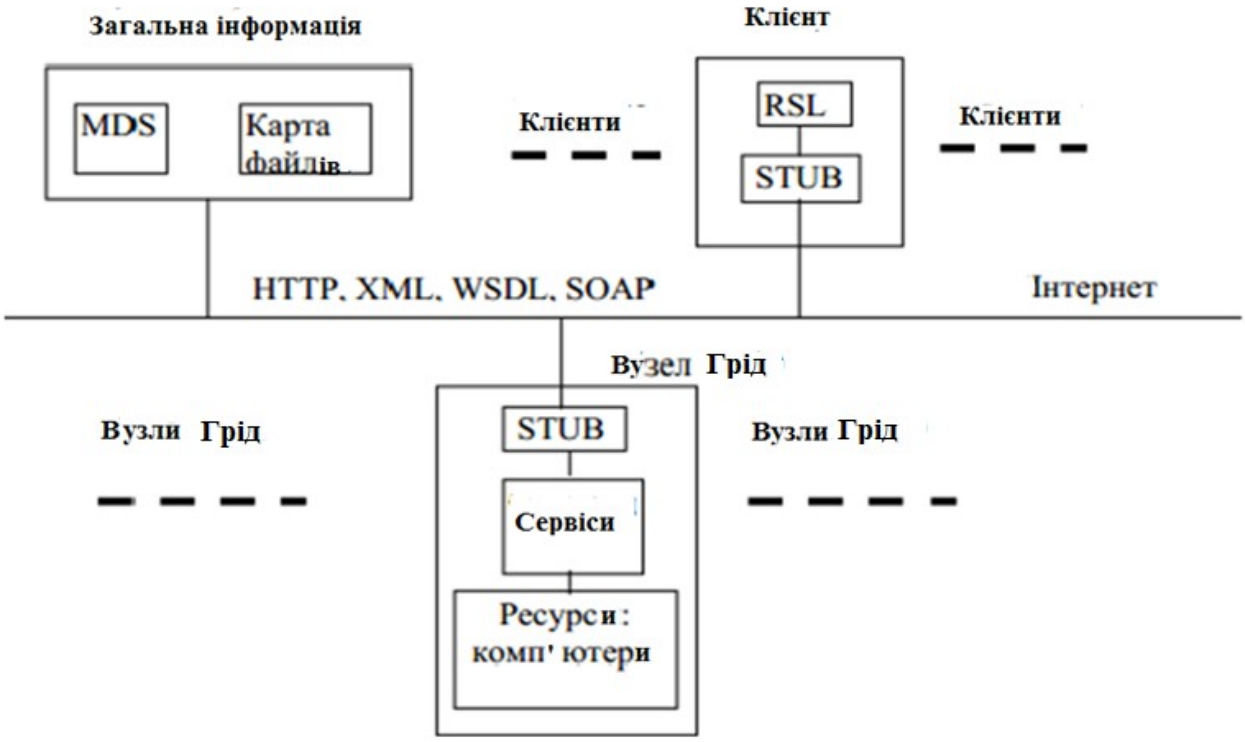


Рис. 2.1.2 Структура і функціонування Грід

Тут RSL (Resource Specification Language) - мова для опису завдання, яке потрібно виконати: програма, дані, де все це розміщено, куди послати результат, вимоги до ресурсів та ін.).

Отже, **Ґрід** — це географічно розподілена інфраструктура, яка об'єднує множину різних типів обчислювальних ресурсів , доступ до яких користувач може отримати з будь-якої точки, незалежно від місця їх розміщення. Ґрід є формою розподілених обчислень, в якому багато комп'ютерів об'єднані в один потужний віртуальний комп'ютер, і які працюють разом для виконання трудомістких завдань.

Ґрід-обчислення з'єднує комп'ютери з багатьох адміністративних доменів для досягнення певних цілей, Одніє з основних стратегій ґрід-обчислень є використання проміжного програмного забезпечення (ППЗ), яке може адаптуватися під завдання, що розв'язується в одному віртуальному домені, для того щоб розподілити програму серед декількох комп'ютерів, іноді навіть серед тисяч. Основним проміжним ПЗ є Globus Toolkit, gLite, UNICORE, EMI (Europen middleware initiative).

Координація додатків на Грід- системах може бути складним завданням, особливо коли координують потоки інформації через розподілені обчислювальні ресурси.

Грід-обчислення покладається на цілі комп'ютери (з повною комплектацією), під'єднані до комп'ютерної мережі (приватної або публічної) звичайним мережевим інтерфейсом, в той час як звичайний суперкомп'ютер містить безліч процесорів, підключених до локальної високошвидкісної шини. Також є певні відмінності у програмуванні та устаткуванні. Писати програми, які працюють у середовищі суперкомп'ютера, що може мати унікальну операційну систему, може бути дорого і складно. Якщо проблема може бути

адекватно розпаралелена, тонкий шар ґрід-інфраструктури може дозволити звичайній програмі запуститись на декількох машинах.

Термін «грід-обчислення» з'явився на початку 1990-х років, як метафора, що демонструє можливість простого доступу до обчислювальних ресурсів як і до електричної мережі (англ. Power grid), у збірнику під редакцією Яна Фостера і Карла Кессельмана "The Grid: Blueprint for a new computing infrastructure ".

Використання вільного часу процесорів і добровільного комп'ютингу стало популярним в кінці 1990-х років після запуску проектів добровільних обчислень GIMPS в 1996 році, distributed.net в 1997 році і SETI @ home в 1999 році. Ці перші проекти добровільного комп'ютингу використовували потужності приєднаних до мережі комп'ютерів звичайних користувачів для вирішення дослідницьких завдань, що вимагають великих обчислювальних потужностей.

Ідеї Грід-системи (включаючи ідеї з областей розподілених обчислень, об'єктно-орієнтованого програмування, використання комп'ютерних кластерів, веб-сервісів та ін.) були зібрані і об'єднані Іеном Фостером, Карлом Кессельманом і Стівом Тікі, яких часто називають батьками Грід-технології. Вони почали створення набору інструментів для Грід-комп'ютингу Globus Toolkit, який включає не тільки інструменти менеджменту обчислень, але й інструменти управління ресурсами зберігання даних, забезпечення безпеки доступу до даних і до самого Грід, моніторингу використання і пересування даних, а також інструментарій для розробки додаткових Грід-сервісів. В даний час цей набір інструментарію є де-факто стандартом для побудови інфраструктури на базі технології Грід, хоча на ринку існує безліч інших інструментаріїв для Грід-систем.

Хмарні обчислення (*Cloud Computing*) — це модель забезпечення повсюдного та зручного доступу на вимогу через мережу до спільного пулу обчислювальних ресурсів, що підлягають налаштуванню (наприклад, до комунікаційних мереж, серверів, засобів збереження даних, прикладних програм та сервісів), і які можуть бути оперативно надані та звільнені з мінімальними управлінськими затратами та зверненнями до провайдера.

При використанні хмарних обчислень програмне забезпечення надається користувачеві як Інтернет-сервіс. Користувач має доступ до власних даних, але не може керувати і не повинен піклуватися про інфраструктуру, операційну систему і програмне забезпечення, з яким він працює. «Хмарою» метафорично називають Інтернет, який приховує всі технічні деталі. Згідно з документом IEEE, опублікованим у 2008 році - «Хмарні обчислення - це парадигма, в рамках якої інформація постійно зберігається на серверах у мережі Інтернет і тимчасово кешується на клієнтській стороні, наприклад, на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах тощо».

Хмарні сервіси, які дозволяють перенести обчислювальні ресурси та дані на віддалені Інтернет-сервери, в останні роки стали одним з основних трендів розвитку IT-технологій.

Концепція хмарних обчислень з'явилася ще в 1960 році, коли американський учений, фахівець з теорії ЕОМ Джон Маккарті (John McCarthy)

висловив припущення, що коли-небудь комп'ютерні обчислення стануть надаватися подібно комунальним послугам (public utility). Розповсюдження мереж з високою потужністю, низька вартість комп'ютерів і пристроїв зберігання даних, а також широке впровадження віртуалізації, сервіс-орієнтованої архітектури привели до величезного зростання хмарних обчислень. Кінцеві користувачі можуть не перейматися роботою обладнання технологічної інфраструктури «в хмарі», яка їх підтримує. Аналогією обчислювальних «хмар» зі звичного життя можуть служити електростанції. Хоча домовласник може купити електрогенератор і піклуватися про його справність самостійно, більшість людей вважає за краще отримувати енергію від централізованих постачальників.

Майже всі сучасні характеристики хмарних обчислень, порівняння їх з електроенергетикою та використання приватних, публічних та громадських моделей були представлені Дугласом Паркхілом (Douglas Parkhill) в книзі «The Challenge of the Computer Utility», в 1966 році. Згідно інших джерел, хмарні обчислення беруть початок з 1950-х років, коли вчений Херб Грош (Herb Grosch) стверджував, що весь світ буде працювати на терміналах, якими керують близько 15 великих центрів обробки даних.

Сам термін «хмара» походить з телефонії, тому що телекомунікаційні компанії, які до 1990-х років пропонували в основному виділені схеми передачі «точка-точка», почали пропонувати віртуальні приватні мережі (VPN), з порівняною якістю обслуговування, але при набагато менших витратах. Премикаючи трафік для оптимального використання каналів, вони мали змогу ефективніше використовувати мережу. Символ хмари був використаний для позначення розмежування між користувачем і постачальником.

Ключову роль в розвитку хмарних обчислень зіграв Amazon, модернізувавши свої центри обробки даних, які, як і більшість комп'ютерних мереж в один момент часу використовують лише 10 % своєї потужності, заради забезпечення надійності при стрибку навантаження. Дізнавшись, що нова хмарна архітектура забезпечує значне внутрішнє підвищення ефективності, Amazon почав нові дослідження в галузі розвитку продуктів для забезпечення хмарних обчислень для зовнішніх клієнтів, і запустив Amazon Web Service (AWS) на основі розподілених обчислень в 2006 році.

На початку 2008 року Eucalyptus став першою API-сумісною платформою з відкритим кодом для розгортання приватної хмари. На початку 2008 року OpenNebula став першим проектом з відкритим кодом для розгортання приватних і гібридних хмар.

У 1993 році термін «хмара» був вперше використаний в комерційних цілях для опису великих мереж, що задіюють технологію високошвидкісної одночасної передачі трафіку всіх видів (дані, голос і відео) в мережах з комутованими каналами. У них з'являлося проміжне віртуальне з'єднання між відправником і отримувачем, що спрощує процес передачі інформації.

На початку XXI століття термін «хмарні обчислення» став вживатися стосовно створеному тоді напрямку SaaS (Software as a Service - «програмне забезпечення як послуга»). Першопрохідцем в цьому відношенні став інтернет-магазин Amazon, який викрутився зі складної ситуації в період кризи доткомів

шляхом переведення своїх дата-центрів на рішення Open Source. 90% серверів компанії стали працювати на базі операційної системи Red Hat Linux (разом з додатком веб-сервера Stronghold, одного з варіантів збірки Apache), а залізо замінили на недорогі моделі серверів на основі чіпсетів від Intel і HR. У 2002 році були створені веб-сервіси Amazon. Вони представляли собою те, що через п'ять років стало називатися «хмарою», - набір сервісів, розташованих на віддалених серверах, до яких користувач може отримати доступ через веб-браузер з будь-якого місця, де є Інтернет.

Провайдери хмарних рішень дозволяють орендувати через інтернет обчислювальні потужності та дисковий простір. Переваги такого підходу - доступність (користувач платить лише за ті ресурси, які йому потрібні) і можливість гнучкого масштабування. Клієнти позбавляються від необхідності створювати і підтримувати власну обчислювальну інфраструктуру.

Хмари можуть бути публічними або приватними. Сервіси публічних хмар можуть використовуватися будь-ким. Основна відмінність приватних хмар від публічних - це надання сервісу з хмари в закритій від загального доступу інфраструктурі обмеженому числу користувачів. Існує ще одне визначення "віртуальна приватна хмара", про яке йде мова, коли провайдер використовує публічну хмарну інфраструктуру для організації приватнї хмари. За такої організаційної структури, частина даних клієнта зберігаються і обробляються за рахунок ресурсів власної інфраструктури, а частина за рахунок ресурсів зовнішнього провайдера. Як приклад віртуальної приватної хмари можна привести сервіс компанії Amazon під назвою Amazon Virtual Private Cloud (Amazon VPC).

Переваги хмарних обчислень:

* Недорогі комп'ютери для користувачів. Користувачам не потрібно купувати дорогі комп'ютери, з великим об'ємом пам'яті і дисків, щоб використовувати програми через веб-інтерфейс. Також немає необхідності в CD і DVD приводах, так як вся інформація і програми залишаються в "хмарі". Користувачі можуть перейти з звичайних комп'ютерів і ноутбуків на більш компактні і зручні нетбуки.

* Збільшена продуктивність комп'ютерів користувачів. Оскільки велика частина програм і служб запускаються віддалено в мережі Інтернет, комп'ютери користувачів з меншим числом програм швидше запускаються і працюють. Одним із прикладів є антивірусне рішення Panda Cloud Antivirus, яке дозволяє сканувати дані на віруси віддалено на потужних серверах і тим самим в 2 рази знижує навантаження на комп'ютер користувача.

* Зменшення витрат і збільшення ефективності IT інфраструктури. Звичайні сервери середньої компанії завантажені на 10-15%. В одні періоди часу є потреба в додаткових обчислювальних ресурсах, в інших ці дорогі ресурси простоюють. Використовуючи необхідну кількість обчислювальних ресурсів в "хмарі" (наприклад, Amazon EC2) в будь-який момент часу, компанії скорочують витрати на обладнання і його обслуговування до 50%. При цьому істотно зростає гнучкість виробництва в постійно мінливій економічній обстановці.

Якщо досить велика фірма стурбована тим, що цінна інформація буде зберігатися і оброблятися на стороні, для такої фірми можна побудувати свою власну "хмару".

* Менше проблем з обслуговуванням. Так як фізичних серверів з впровадженням Cloud Computing стає менше, їх стає легше і швидше обслуговувати. Що стосується програмного забезпечення, то останне встановлюється, налаштовується і оновлюється в "хмарі".

* Постійне оновлення програм. У будь-який час, коли користувач запускає віддалену програму, він може бути впевнений, що ця програма має останню версію - без необхідності щось заново встановлювати або платити за оновлення.

* Збільшення доступних обчислювальних потужностей. У порівнянні з персональним комп'ютером обчислювальна потужність, доступна користувачу "хмарних" комп'ютерів, практично обмежена лише розміром "хмари", тобто загальною кількістю залучених серверів. Користувачі можуть запускати більш складні завдання, з великою кількістю необхідної пам'яті, місця для зберігання даних, тоді, коли це необхідно. Іншими словами, користувачі можуть при бажанні легко і дешево попрацювати з суперкомп'ютером без будь-яких фактичних придбань.

* Необмежений обсяг збережених даних. У порівнянні з доступним місцем для зберігання інформації на персональних компьютерах обсяг сховища в "хмарі" може гнучко і автоматично підлаштовуватися під потреби користувача. При зберіганні інформації в "хмарі" користувачі можуть забути про обмеження, що накладаються звичайними дисками, - "хмарні" розміри обчислюються мільярдами гігабайт доступного місця.

* Сумісність з більшістю операційних систем. У Cloud Computing операційні системи не грають ніякої ролі. Користувачі Unix можуть обмінюватися документами з користувачами Microsoft Windows і навпаки без будь-яких проблем. Доступ до програм та віртуальних комп'ютерів відбувається за допомогою веб-браузера, або іншими засобами доступу, що встановлюються на будь-який персональний комп'ютер з будь-якою операційною системою.

* Покращена сумісність форматів документів. Хорошим прикладом сумісності є офісний пакет Google Docs, що дозволяє спільну роботу над документами, презентаціями і таблицями, маючи під рукою будь-який комп'ютер з веб-браузером.

* Простота спільної роботи групи користувачів. При роботі з документами в "хмарі" немає необхідності пересилати один одному їх версії або послідовно редагувати їх. Тепер користувачі можуть бути впевненими, що перед ними остання версія документа і будь-яка зміна, внесена одним користувачем, миттєво відбивається в іншого. Можна уявити собі, як 100 чоловік одночасно редагують макет книги - спільна робота в реальному часі.

* Повсюдний доступ до документів. Якщо документи зберігаються в "хмарі", вони можуть бути доступні користувачам в будь-який час і в будь-якому місці. Більше немає такого поняття як забуті файли: якщо є Інтернет - вони завжди поруч.

* Доступність з різних пристроїв. Користувачі Cloud Computing мають набагато

більш широкий вибір пристроїв доступу до документів і програм. Тепер можна вибирати між звичайним персональним комп'ютером, ноутбуком, Інтернет-планшетом, смартфоном або нетбуком.

* Постійне з'єднання з мережею Інтернет. Cloud Computing завжди вимагає з'єднання з мережею Інтернет. Або майже завжди. Деякі "хмарні" програми завантажуються на локальний комп'ютер і використовуються в той час, коли Інтернет недоступний. В інших випадках, якщо немає доступу в Інтернет - немає роботи, програм, документів. Це напевно найсильніший аргумент проти Cloud Computing. Але з огляду на розвиток сучасного світу, Інтернет буде доступний завжди і скрізь, як, наприклад, електрика і вода.

* Програми можуть працювати повільніше, ніж на локальному комп'ютері. Деякі програми, в яких потрібна передача значної кількості інформації, працюватимуть на локальному комп'ютері швидше не тільки через обмеження швидкості доступу в Інтернет, а й із-за завантаженості віддалених серверів і проблем на шляху між користувачем і "хмарою".

* Безпека даних може бути під загрозою. Тут ключовим є слово "може". Все залежить від того, хто надає "хмарні" послуги. Якщо цей хтось надійно шифрує Ваші дані, постійно робить їх резервні копії, вже не один рік працює на ринку подібних послуг і має хорошу репутацію, то загрози безпеки даних може ніколи не статися. Як сказав відомий фахівець з криптографії та комп'ютерної безпеки Брюс Шнайер, все питання в довірі.

Якщо Ваші дані в "хмарі" втрачені, вони втрачені назавжди. Це факт. Але втратити дані в "хмарі" набагато складніше, ніж на локальному комп'ютері.

Незважаючи на те, що кількість плюсів перевершує мінуси, в кожній конкретній ситуації вони мають велику важливість, або, навпаки, не мають ніякого значення. Кожен вибирає сам.

Сьогодні хмарні обчислення, що дозволяють компаніям передати на аутсорсинг їх процеси обробки даних комерційним провайдерам таких послуг, стали популярним, вже досить великим і швидкозростаючим ринком. Але природа хмарних обчислень змушує клієнтів задуматися про захищеність даних і безпеку такої послуги. Якщо дані, або їх обробка не знаходяться під безпосереднім контролем компанії - власника інформації, то у неї є привід для занепокоєння.

Недоліки "хмарних" рішень зводяться, в основному, до проблеми довіри постачальнику сервісу, від якого залежить як безперебійна робота, так і збереження важливих даних користувача. Крім того хмарні обчислення висувають високі вимоги до якості каналів зв'язку, які гарантують повсюдний якісний доступ в інтернет.

Поки для клієнтів немає швидкого і зрозумілого способу переконатися, що хмарні обчислення надійно захищені.

Однак незважаючи на недоліки, плюси від впровадження даної технології зрозумілі всім.

Обчислювальна хмара може бути розгорнута як: приватна, публічна, громадська або гібридна, персональна

Приватна хмара (*private cloud*) - це хмарна інфраструктура, яка призначена для використання виключно однією організацією, що включає декілька

користувачів (наприклад, підрозділів). Приватна хмара може перебувати у власності, керуванні та експлуатації як самої організації, так і третьої сторони (чи деякої їх комбінації). Така хмара може фізично знаходитись як в, так і поза юрисдикцією власника.

Публічна хмара (*public cloud*) - це хмарна інфраструктура, яка призначена для вільного використання широким загалом. Публічна хмара може перебувати у власності, керуванні та експлуатації комерційних, академічних (освітніх та наукових) або державних організацій (чи будь-якої їх комбінації). Публічна хмара перебуває в юрисдикції постачальника хмарних послуг.

Громадська хмара (*community cloud*) - це хмарна інфраструктура, яка призначена для використання конкретною спільнотою споживачів із організацій, що мають спільні цілі (наприклад, місію, вимоги щодо безпеки, політику та відповідність різноманітним вимогам). Громадська хмара може перебувати у спільній власності, керуванні та експлуатації однієї чи більше організацій зі спільноти або третьої сторони (чи деякої їх комбінації). Така хмара може фізично знаходитись як в, так і поза юрисдикцією власника.

Гібридна хмара (*hybrid cloud*) - це хмарна інфраструктура, яка складається з двох або більше різних хмарних інфраструктур (приватних, громадських або публічних), які залишаються унікальними сутностями, але з'єднані між собою стандартизованими або приватними технологіями, що уможливлюють переносимість даних та прикладних програм (наприклад, використання ресурсів публічної хмари для балансування навантаження між хмарами).

Персональна хмара (*personal cloud*) - це приватна колекція цифрового контенту та додаткових сервісів, які доступні з будь-якого пристрою і призначена для використання окремою особою (власником) та особами, яким надано доступ. Це місце, де користувач має можливість зберігати, синхронізувати, транслювати в потік та розповсюджувати приватний контент на сумісні платформи, екрани, з одного місцеположення в інше.

Єдиного і однозначного визначення хмарних обчислень не існує, але основним є те що концепція Cloud Computing – це такий підхід до розміщення, надання і споживання прикладних програм і комп'ютерних ресурсів, при якому прикладне програмне забезпечення і ресурси стають доступні через Інтернет у вигляді сервісів, які використовуються на різних платформах і засобах. Оплата таких сервісів здійснюється за їх фактичним використанням.

Незважаючи на те, що ця концепція не нова, сучасні технології роблять її значно більш ефективною завдяки розподілу фізичних систем за допомогою серверної віртуалізації. Cloud Computing надає користувачам доступ до потужних обчислювальних та накопичувальних ресурсів без необхідності знання того, де ці ресурси розташовані і як вони налаштовані.

Приведене визначення хмарних обчислень перетинається з багатьма існуючими технологіями, такими як Грід, Утилітарні обчислення, Сервісні обчислення та розподілені обчислення загалом. Але хмарні обчислення не тільки перетинаються з Грід -обчисленнями, вони насправді походять від Грід - обчислень та будуються на них, як на їх основі та інфраструктурній підтримці. Розвиток є результатом переміщення фокусу від інфраструктури, яка надає

ресурси зберігання та обчислення (як у випадку Грід), до тієї, яка заснована на економії та націлена надавати більш абстрактні ресурси та сервіси (як у випадку хмар). Хмарні обчислення не є новою парадигмою обчислюваної інфраструктури; навпаки, це бізнес- модель, в якій обчислювальні ресурси, такі як обчислення та накопичення, упаковані як сервіси, що схожі на фізичні комунальні послуги, такі як електрика та комутована телефонна мережа загального користування.

Можна вважати, що Грід та хмарні обчислення доповнюють одне одного. Грід -системи забезпечують високе завантаження обчислювальних ресурсів за допомогою розподілу одного складного завдання на декілька обчислювальних вузлів, а хмарні обчислення йдуть шляхом виконання декількох завдань на одному сервері у вигляді віртуальних машин.

Грід - і хмарні обчислення націлені на різні типи обчислень. Спочатку, Грід - обчислення були орієнтовані на рішення наукових завдань за допомогою суперкомп'ютерних систем. Нині Грід застосовується для науково-дослідницьких завдань, вирішення яких вимагає об'єднання декількох суперкомп'ютерних платформ. З іншого боку, хмарні обчислення орієнтовані не на вирішення окремих завдань, а на перманентне надання певних сервісів кінцевим користувачам. Вони забезпечують динамічний розподіл фізичних ресурсів для задоволення змінного завантаження таких сервісів.

Також потрібно відмітити різну взаємодію з постачальниками ресурсів. Грід -обчислення грунтуються на понятті віртуальних організацій, що включають декілька різних окремих організацій з чіткими правилами взаємодії між ними і чіткими політиками надання програмно-апаратних ресурсів, в той час як хмарні обчислення забезпечують можливість будь-якій компанії використовувати хмарні сервіси для вирішення власних завдань, оплачуючи тільки ті ресурси, які потрібні. Грід -платформи надають базу для розгортання обчислювальної інфраструктури, в той час як хмарні обчислення надають інтегрований підхід на усіх рівнях для надання інформаційних ресурсів: інфраструктура як послуга, платформа як послуга, програмне забезпечення як послуга.

Питання для самоконтролю

12. Привести визначення Grid computing і Cloud computing
13. На що націлені Грід - і хмарні обчислення ?
14. Яке програмне забезпечення використовується в Грід і хмарних обчис.?
15. Хто може бути учасником Grid computing?
16. Що таке віртуальна організація?
17. Взаємодію з постачальниками Грід та Cloud ресурсів
18. Основні задачі Грід -платформ
19. Основні задачі Cloud -платформ
20. Переваги хмарних обчислень
21. Недоліки хмарних обчислень
22. Що таке *private cloud*, *private cloud, community cloud, hybrid cloud, personal cloud ?*

Література основна

1. Петренко А.І., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.
2. Тимошин Ю.А., Огляд сервісів: лекція. – НТУУ «КПІ», ФІОТ, 2014. – 18с.

Література допоміжна

3. Foster I., Kesselman C. The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.
4 I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. - International J. Supercomputer Applications, 15(3), 2001, http://www.globus.org/research/papers/anatomy.pdf
5 . Foster, C. Kesselman, S. Tuecke, J.M. Nick. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. - Morgan Kaufmann Publishers, 2002.
6 . Foster, H. Kishimoto, A. Savva, D. Berry et al. The Open Grid Services Architecture. - Global Grid Forum, 2005
7 M. Z. Zgurovsky. Development of Educational and Research Segment of Information Society in Ukraine. - //Proc. WSIS.-Tunis.- 2004.-P.103-107.
8 Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
9 GRID Computing in Research and Education, April 2005. – IBM Redbook, − 1 45 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
10 GRID Services Programming and Application Enablement, May 2004. – IBM Redbook, − 273 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
11. Anthony Velte, Toby Velte, Robert Elsenpeter: Cloud Computing: a practical approach, 2010
12. “The evolution of Cloud Computing”. Retrieved 22 April 2015.

2.2 Архітектура Грід

Open GRID Services Architecture (OGSA) направлена на стандартизацію адресації (для сумісності) за допомогою визначення основи структури додатку Грід. По суті стандарт OGSA визначає сервіси Грід, їх можливості і те, на яких технологіях вони засновані. Проте OGSA не розрізняє особливостей технічної сторони специфікації; метою є визначення – що є системою Грід. OGSA називають архітектурою, оскільки вона направлена на побудову і установку інтерфейсів, з яких можуть бути побудовані

системи, засновані на відкритих стандартах WSDL.

Архітектура Грід визначає системні компоненти, цілі і функції цих компонент і відображає способи взаємодії компонент один з одним. Архітектура Грід є архітектурою взаємодіючих протоколів, сервісів і інтерфейсів, що визначають базові механізми, за допомогою яких користувачі встановлюють з'єднання з Грід -системою, спільно використовують обчислювальні ресурси для вирішення різного роду завдань.

Архітектура протоколів Грід розділена на рівні (рис.2.2.1), компоненти кожного з них можуть використовувати можливості компонент будь-якого з розташованих нижче рівнів. В цілому ця архітектура задає вимоги для основних компонент технології (протоколів, сервісів, прикладних інтерфейсів і засобів розробки ПО), не надаючи строгий набір специфікацій, залишаючи можливість їх розвитку в рамках прийнятої концепції.

Рис.2.2.1 Рівні архітектури протоколів Грід

Інфраструктура Грід заснована на наданні ресурсів в загальне користування, з одного боку, і на використанні публічно доступних ресурсів, з іншого. У цьому плані ключове поняття інфраструктури Грід – віртуальна організація, в якій кооперуються як споживачі, так і власники ресурсів. Мотиви кооперації можуть бути різними. У існуючих Грід - системах віртуальна організація є об'єднанням (колаборацію) фахівців з деякої прикладної області, які об'єднуються для досягнення загальної мети.

Грід - система є середовищем колективного комп’ютингу, в якій кожен ресурс має власника, а доступ до ресурсів відкритий в режимі, що розділяється за часом і по простору, безлічі користувачів, які входять до віртуальної організації. Віртуальна організація може утворюватися динамічно і мати обмежений час існування.

Ефективний розподіл ресурсів і їх координація є основними завданнями системи Грід і для їх вирішення використовується планувальник (брокер ресурсів). Користуючись інформацією про стан Грід - системи, планувальник визначає найбільш відповідні ресурси для кожного конкретного завдання і резервує їх для її виконання. Під час виконання завдання планувальник може запитати додаткові ресурси, або звільнити надмірні. Після завершення

завдання всі відведені для нього обчислювальні ресурси звільняються, а ресурси пам'яті можуть бути використані для зберігання результатів роботи.

Важливою властивістю систем Грід є те, що користувачеві не потрібно знати про фізичне розташування ресурсів, відведених його завданню. Вся робота по управлінню, перерозподілу і оптимізації використання ресурсів лягає на планувальник і виконується непомітно для користувача. Для користувача створюється ілюзія роботи в єдиному інформаційному просторі, що володіє величезними обчислювальними потужностями і об'ємом пам'яті.

Грід є найбільш складним інформаційним середовищем, яке коли-небудь створювалося людиною. Для системи такої складності дуже важлива проблема забезпечення надійного функціонування і відновлення при збоях. Людина не здатна устежити за станом тисяч різних ресурсів, що входять в Грід - систему, і з цієї причини завдання контролю над помилками покладається на систему моніторингу, яка стежить за станом окремих ресурсів. Дані про стан заносяться в інформаційні ресурси, звідки вони можуть бути прочитані планувальником і іншими сервісами, що дозволяє мати достовірну інформацію, яка постійно оновлюється, про стан ресурсів.

У Грід - системах використовується складна система виявлення і класифікації помилок. Якщо помилка відбулася з вини завдання, то завдання буде зупинено, а відповідна діагностика направлена її власникові (користувачеві). Якщо причиною збою послужив ресурс, то планувальник проведе перерозподіл ресурсів для даного завдання і перезапустить його.

Збої ресурсів є не єдиною причиною відмов в Грід - системах. Через величезну кількість завдань і постійно змінної складної конфігурації системи важливо своєчасно визначати переобтяжені і вільні ресурси, проводячи перерозподіл навантаження між ними. Переобтяжений мережевий ресурс може стати причиною відмови інших ресурсів. Планувальник, використовуючи систему моніторингу, постійно стежить за станом ресурсів і автоматично приймає необхідні заходи для запобігання перевантаженням і простою ресурсів.

У розподіленому середовищі, яким є Грід - система, життєво важливою властивістю є відсутність так званої єдиної точки збоїв. Це означає, що відмова будь-якого ресурсу не повинна приводити до збою в роботі всієї системи. Саме тому планувальник, система моніторингу і інші сервіси Грід - системи розподілені і продубльовані. Не дивлячись на всю складність, архітектура Грід розроблялася з метою забезпечити максимальну якість сервісу для користувачів. У Грід - системах використовуються сучасні технології передачі даних, забезпечення безпеки і відмовостійкості.

Базовий рівень

Базовий рівень (Fabric Layer) архітектури Грід описує служби, що безпосередньо працюють з ресурсами. Ресурс є одним з основних понять архітектури Грід.

Можна виділити декілька основних типів ресурсів (рис.2.2.2):

· обчислювальні ресурси;

· ресурси зберігання даних;

· інформаційні ресурси, каталоги;
· мережні ресурси.

Рис. 2.2.2 Ресурси Грід

Обчислювальні ресурси надають користувачеві Грід -системи (точніше кажучи, завданню користувача) процесорні потужності. Обчислювальними ресурсами можуть бути як кластери, так і окремі робочі станції. При всій різноманітності архітектури будь-яка обчислювальна система може розглядатися як потенційний обчислювальний ресурс Грід -системи.

Необхідною умовою для цього є наявність спеціального програмного забезпечення, так званого ПЗ проміжного рівня (middleware), що реалізує стандартний зовнішній інтерфейс з ресурсом і дозволяє зробити ресурс доступним для Грід - системи. Основною характеристикою обчислювального ресурсу є продуктивність.

Ресурси пам'яті є простором для зберігання даних. Для доступу до ресурсів пам'яті також використовується програмне забезпечення проміжного рівня, що реалізує уніфікований інтерфейс управління і передачі даних. Основною характеристикою ресурсу пам'яті є його об'єм.

Інформаційні ресурси і каталоги є особливим видом ресурсів пам'яті. Вони служать для зберігання і надання метаданих і інформації про інші ресурси Грід - системи. Інформаційні ресурси дозволяють структуровано зберігати величезний об'єм інформації про поточний стан Грід - системи і ефективно виконувати завдання пошуку.

Мережевий ресурс є сполучною ланкою між розподіленими ресурсами Грід - системи. Основною характеристикою мережевого ресурсу є швидкість передачі даних. Географічно розподілені системи на основі даної технології здатні об'єднувати тисячі ресурсів різного типу, незалежно від їх географічного положення.

Рівень зв'язку

Рівень зв'язку (Connectivity Layer) визначає комунікаційні протоколи і протоколи аутентифікації.

Комунікаційні протоколи забезпечують обмін даними між компонентами базового рівня.

Протоколи аутентифікації, ґрунтуючись на комунікаційних протоколах, надають криптографічні механізми для ідентифікації і перевірки

достовірності користувачів і ресурсів. Протоколи рівня зв'язку повинні забезпечувати надійний транспорт імаршрутизацію повідомлень, а також привласнення імен об'єктам мережі. Недивлячись на існуючі альтернативи, зараз протоколи рівня зв'язку в Грід-системах припускають використання тільки стека протоколів TCP/IP, зокрема: на мережевому рівні – IP і ICMP, транспортному рівні – TCP, UDP, на прикладному рівні – HTTP, FTP, DNS, RSVP. Враховуючи бурхливий розвиток мережевих технологій, в майбутньому рівень зв'язку, можливо, залежатиме і від інших протоколів.

Для забезпечення надійного транспорту повідомлень в Грід-системі повинні використовуватися рішення, які передбачають гнучкий підхід до безпеки комунікацій. В даний час ці рішення ґрунтуються як на існуючих стандартах безпеки, спочатку розроблених для Інтернет (SSL, TLS), так і на нових розробках.

Ресурсний рівень

Ресурсний рівень (Resource Layer) побудований над протоколами комунікації і аутентифікації рівня зв'язку архітектури Грід. Ресурсний рівень реалізує протоколи, які забезпечують виконання наступних функцій:
- узгодження політик безпеки використання ресурсу;
- процедура ініціації ресурсу;
- моніторинг стану ресурсу;
- контроль над ресурсом;
- облік використання ресурсу.

Протоколи цього рівня спираються на функції базового рівня для доступу і контролю над локальними ресурсами. На ресурсному рівні протоколи взаємодіють з ресурсами, використовуючи уніфікований інтерфейс і не розрізняючи архітектурні особливості конкретного ресурсу.

Розрізняють два основні класи протоколів ресурсного рівня:
· **інформаційні протоколи**, які отримують інформацію про структуру і стан ресурсу, наприклад, про його конфігурацію, поточне завантаження, політику використання;
· **протоколи управління**, які використовуються для узгодження доступу до ресурсів, які розділяються, визначаючи вимоги і допустимі дії по відношенню до ресурсу (наприклад, підтримка резервування, можливість створення процесів, доступ до даних). Протоколи управління повинні перевіряти відповідність запрошуваних дій політиці розділення ресурсу, включаючи облік і можливу оплату. Вони можуть підтримувати функції моніторингу статусу і управління операціями.

Список вимог до функціональності протоколів ресурсного рівня близький до списку для базового рівня архітектури Грід. Додалася лише вимога єдиної семантики для різних операцій з підтримкою системи оповіщення про помилки.

Колективний рівень

Колективний рівень (Collective Layer) відповідає за глобальну інтеграцію різних наборів ресурсів, на відміну від ресурсного рівня,

сфокусованого на роботі з окремо узятими ресурсами. У колективному рівні розрізняють **загальні** і **специфічні** (для додатків) **протоколи. До загальних протоколів** відносяться, в першу чергу, протоколи виявлення і виділення ресурсів, системи моніторингу і авторизації співтовариств. **Специфічні протоколи** створюються для різних додатків Грід (наприклад, протокол архівації розподілених даних, або протоколи управління завданнями збереження стану і тому подібне).

Функції і сервіси, які реалізуються в протоколах колективного рівня рівня:

· **сервіси каталогів** дозволяють віртуальним організаціям виявляти вільні ресурси, виконувати запити по іменах і атрибутах ресурсів, таким як тип і завантаження;

· **сервіси сумісного виділення, планування і розподілу ресурсів** забезпечують виділення одного або більше ресурсів для певної мети, а також планування виконуваних на ресурсах завдань;

· **сервіси моніторингу і діагностики** відстежують аварії, атаки і перевантаження;

· **сервіси дублювання (реплікації) даних** координують використання ресурсів пам'яті в рамках віртуальних організацій, забезпечуючи підвищення швидкості доступу до даних відповідно до вибраних метрик, таких як час відповіді, надійність, вартість і т. д.;

· **сервіси управління робочим завантаженням** застосовуються для опису і управління багатокроковими, асинхронними, багатокомпонентними завданнями;

· **служби авторизації співтовариств** сприяють поліпшенню правил доступу до ресурсів, які розділяються, а також визначають можливості використання ресурсів співтовариства. Подібні служби дозволяють формувати політики доступу на основі інформації про ресурси, протоколи управління ресурсами і протоколи безпеки зв’язуючого рівня;

· **служби обліку і оплати** забезпечують збір інформації про використання ресурсів для контролю звернень користувачів;

· **сервіси координації** підтримують обмін інформацією в потенційно великому співтоваристві користувачів.

Прикладний рівень.

Прикладний рівень (Application Layer) описує призначені для користувача застосування (додатки), які працюють в середовищі віртуальної організації. Додатки функціонують, використовуючи сервіси, визначені на рівнях, які пролягають нижче. На кожному з рівнів є певні протоколи, які забезпечують доступ до необхідних служб, а також прикладні програмні інтерфейси (Application Programming Interface – API), відповідні даним протоколу.

Питання для самоконтролю

1. Що таке Грід?
2. На що націлені Грід - обчислення ?

3. Охарактерізувати програмне забезпечення Грід та його особливості
4. Архітектура протоколів Грід
5. Що таке Fabric Layer?
6. Connectivity Layer – призначення та протоколи
7. Що таке **Ресурсний рівень?**
8. Що таке **Прикладний рівень?**
9. Назвати основні класи протоколів ресурсного рівня
10. Що таке Collective Layer?

Література

1. Петренко А.І., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.
2. Петренко А.І., Застосування GRID технологій в науці та освіті: додатковий матеріал до вивч. курсу для студ. спец. «Інформаційні технології проектування». − К.: НТУУ «КПІ», 2008. − 144 С.
Режим доступу: http://moodle.ntu-kpi.kiev.ua
3. Петренко А.І., Булах Б.В., Хондар В.С. *Семантичний Грід для науки і освіти*. //– К. : НТУУ «КПІ», 2010. – 180 с.

2.3 **Стандарти побудови Грід**

Термін «грид» був введений в використання Яном Фостером на початку 1998 року публікацією книги «Грід. Нова інфраструктура обчислень»: *Грід – це система, яка координує розподілені ресурси за допомогою стандартних, відкритих, універсальних протоколів і інтерфейсів для забезпечення нетривіальної якості обслуговування (QoS – Quality of Service).*

Хотя за останнє десятиліття базова ідея грід не зазнала суттєвих змін, всеохватного визначення грід не існує і досі.

Грід є технологією забезпечення гнучкого, безпечного і скоординованого загального доступу до ресурсів. У термінології Грід сукупність людей і організацій, які спільно розв'язують ту чи іншу загальну задачу і надають у користування один одному свої ресурси, називають **віртуальною організацією.**

Є два основні критерії, що виділяють Грід -системи серед інших систем, які забезпечують доступ до ресурсів, що розділяються:

· Грід - система координує розрізнені ресурси. Ресурси не мають загального центру управління, а Грід -система займається координацією їх використання, наприклад, балансуванням навантаження. Тому проста система управління ресурсами кластера не є системою Грід, оскільки здійснює централізоване управління всіма вузлами даного кластера, маючи до них повний доступ.

· Грід - система будується на базі стандартних і відкритих протоколів, сервісів та інтерфейсів. Не маючи стандартних протоколів, неможливо легко і швидко підключати нові ресурси в Грід -систему, розробляти новий вигляд сервісів і так далі.

Ключовим моментом в розробці Грід - додатків являється стандартизація, яка позволяє організовувати пошук, використання, розміщення та моніторинг різних компонентів, складаючих єдину віртуальну систему, навіть якщо вони предоставляются різними постачальниками послуг, чи керуються різними організаціями.

Найбільш важливим стандартом, покликаним визначити загальну, стандартну і відкриту архітектуру Грід, є стандарт Open Grid Services Architecture (OGSA). OGSA є сервіс-орієнтованою архітектурою, в якій специфікується набір розподілених обчислювальних патернів, що реалізовуються з використанням Web-сервісів. Стандарт призначається для визначення всіх основних сервісів, які можуть використовуватися в додатках e-business або e-science, включаючи управління роботами і ресурсами, комунікації і безпеку.

Стандарти, які використовуються для побудови архітектури Грід

Існує декілька стандартів, використовуваних для побудови архітектури Грід. Ці стандарти утворюють базові блоки, які дозволяють посилати запити додаткам і базам даних, а також дозволяють розгорнути програмне забезпечення, що дозволяє спростити управління бізнес - процесом.

До стандартів Грід і зв'язаних з ними стандартів слід віднести:

- Комунікації «програма - програма» (SOAP, WSDL, UDDI); (SOAP (Simple Object Access Protocol)- простий протокол для обміну ; WSDL (Web Services Definition Language)- опис інтерфейсу взаємодії компонентів розподіленої системи (мова описів веб-сервісів); UDDI (Universal Description, Discovery, and Integration)-стандартний протокол опису веб-сервісів і їх пошуку).
- Сумісне використання даних (мова XML).
- Передача повідомлень(SOAP, WS - Addressing, MTOM (для додатків));
- Надійна передача повідомлень (WS - Reliable Messaging);
- Управління робочим процесом (WS - Management);
- Управління транзакціями(WS - Coordination, WS - AtomicTransaction, WS - Business - Activity);
- Розподіл ресурсів (WS - RF або система ресурсних Web – сервісів);
- Забезпечення безпеки (WS - Security, WS - SecureConversation, WS - Trust, WS - Federation, Система безпечних зв'язків Kerberos для Web - сервісів;
- Обробка метаданих(WSDL, UDDI, WS - Policy).
- Orchestration (стандарти, використовувані для абстрагування бізнес - процесів від логіки додатків і джерел даних і для встановлення правив, які дозволяють бізнес - процесам взаємодіяти між собою);
- Верхній рівень управління бізнес - процесом (інженерна мова бізнес - процесу для Web сервісів - BPEL4WS)
- Події, що запускають бізнес - процес (WS - Notification).

Горизонтальна і вертикальна інфраструктура програмного забезпечення, яка необхідна для забезпечення безпеки, пошти, обміну повідомленнями, робочого потоку, колаборації, обмінів програма - програма, а також середовища

сумісного використання даних може бути знайдена в інфраструктурних пропозиціях таких компаній як IBM (WebSphere), Microsoft (.NET), BEA (WebLogic) і Sun (ONE). Web - сервіси і XML реалізації містяться в пропозиціях інших постачальників.

1. Сервіс-орієнтована архітектура

Фундаментальною концепцією OGSA (*Open Grid Services Architecture*) є те, що сервісна архітектура складається з компонентів служб Грід, які працюють як спеціальні Web - сервіси, що надають ряд інтерфейсів, які відповідають спеціальним вимогам. .

SOA (Service Oriented Architecture – архітектура, орієнтована на сервіс) визначає, як взаємодіють два обчислювальні об'єкти, для того, щоб один об'єкт дав можливість другому виконати визначену роботу на користь першого. Ця робота співвідноситься з сервісом, а дії сервісу визначаються мовою описів. Кожна взаємодія самостійна і практично не залежить від іншого. Бізнес-додатки створюються для автоматизації різних бизнес - процесів, але часто без здійснення в них можливості адаптуватися до потреб, що змінюються, доопрацювання бізнес- процесів в даному середовищі достатньо трудомістке завдання. Все тому, що бізнес- додатки традиційно створюються як одиничні, монолітні, такі, які включають всі інструменти. Тому будь - які зміни в них достатньо дорогі і займають багато часу. У середовищі SOA, додатки створюються у вигляді набору сервісів, кожен з яких має свої завдання і властивості. У міру зміни потреб, деякі сервіси можуть додаватися, деякі видалятися або доопрацьовуватися.

Web - сервіси володіють наступними характеристиками:

- Це Інтернет- додаток, що виконує спеціальні завдання і що підкоряється стандартним специфікаціям.
- Сервіс описується на XML і доступ до нього може бути здійснений за допомогою XML - повідомлень.
- Він може бути анонсований, виявлений і викликаний в розподіленому обчислювальному середовищі.
- Він не залежить від платформи або мови.

Web - сервіс є системою ПЗ, URL якого ідентифікується, а інтерфейси і зв'язки визначені і описані за допомогою XML. Він може бути виявлений іншими системами ПЗ. Ці системи, у свою чергу, можуть взаємодіяти з Web - сервісом, використовуючи XML- повідомлення, що передаються за допомогою Інтернет - протоколів. Мова описів Web - сервісів (WSDL) фактично є заснованим на XML стандартом для опису Web - сервісів. Простий протокол доступу до об'єктів (SOAP) є заснованим на XML стандартним мережевим протоколом для обміну повідомленнями між Web - сервісами (описи W3C).

2. Мова описів Web - сервісів

WSDL документ (мова описів Web - сервісів) визначає Web - сервіс, використовуючи приведені нижче основні елементи:

Таблиця. Деякі позначення елементів у мові описів Web – сервісів

Елемент	Визначає
<portType>	Постачальники повинні показувати, чи можливо ефективно управляти процесами Грід, включаючи модель на випадок збитків
<Message>	Повідомлення, використовувані в Web - сервісі. Абстрактне визначення передаваних даних
<Types>	Типи даних, використовувані в Web - сервісі. Надає інформацію про будь - які складні типи даних, вживаних в документах WSDL. У разі використання простих типів даних цей елемент не потрібний.
<Binding>	Протоколи з'єднання, використовувані в Web - сервісі. Описує, як викликається операція за допомогою визначення протоколу і формату даних
<Port>	Визначає одиночну крайову точку у вигляді адреси для з'єднання, тобто крайову точку з'єднання
<Service>	Визначає адреси портів з'єднання. Служба – сукупність мережевих крайових точок або портів

WSDL документ містить описи елементів, які складаються з блоків types, message, portType, binding, і service elements, що описане в таблиці вище. Основна структура документа WSDL виглядає таким чином:

<definitions>
<types>
опис типів. . .
</types>
<message>
опис повідомлень . . .
</message>
<portType>
опис порту . . .
</portType>
<binding>
опис з'єднання . . .
</binding>
</definitions>

3. Web Services Inspection Language

Web Services Inspection Language (WSIL) – простий механізм виявлення Web - сервісів. WSIL – формат XML документу, створений для полегшення збору і виявлення Web - сервісів. Створений IBM і Microsoft і виданий в кінці 2001 року, WSIL є привабливим за рахунок своєї простоти, в порівнянні з UDDI він простий і

краще «піднімає» існуючі Web - сервіси. Модель WSIL децентрализована і «піднімає» існуючі Web - сервіси прямо на місці.

4. Universal Description, Discovery, and Integration

Universal Description, Discovery, and Integration (UDDI) – стандартний протокол опису Web - сервісів і протокол їх пошуку. Реєстр (UDDI) може містити метадані для будь - яких видів сервісів, разом з варіантами «якнайкращої практики», вже визначеними для сервісів, описаних за допомогою WSDL. За рахунок розбиття Web - сервісів на групи, які взаємодіють з категоріями і бізнес- процесами, UDDI здатний ефективно шукати Web - сервіси. Специфікація UDDI визначає ієрархічну схему XML, що забезпечує модель для анонсування, перевірки і виклику інформації про Web - сервіси. Вибір ліг на XML, оскільки його формат представлення даних не залежить від платформи і відображає ієрархічні взаємозв'язки. У UDDI використовуються технології, засновані на загальних інтернет- протоколах TCP/IP, HTTP, XML і SOAP. Існує 2 види UDDI реєстрів: публічні реєстри UDDI – для збору різних бізнесів, повідомлення про їх сервіси і приватні реєстри UDDI, які роблять те ж саме але для організацій.

UDDI реєстр містить наступні структурні типи даних:

- **businessEntity**. XML - элемент верхнього рівня в бізнес запису UDDI. businessEntity збирає дані по запиту інформації про бізнес-обслуговування, категорії продукту або виробництва, географічному положенні, а також контактну інформацію. Він підтримує пошук по організаціях, продуктах і географічному положенні.
- **businessService**. Логічне продовження структури даних businessEntity і родоначальник структури bindingTemplate. businessService містить описову інформацію бізнес- послуг з груп споріднених технічних послуг, включаючи ім'я групи, коротку інформацію про групу, опис технічної послуги, інформацію про категорію.
- **bindingTemplate**. Логічне продовження структури businessService. bindingTemplate містить дані, які відносяться до додатків, які необхідно запустити або пов'язати з Web - сервісом. Ця інформація містить URL Web - сервіса, посилання на специфікації інтерфейсу і ін.
- **tModel**. Містить описи специфікації Web - сервісів або систематики, які формують основу для технічних ідентифікаторів. Роль tModel полягає в наданні технічних специфікацій Web - сервісів, що дозволяє полегшити пошук Web - сервісів, сумісних з певною технічною специфікацією. Користувачі Web - сервісів можуть легко визначити інші сумісні Web - сервіси, грунтуючись на описі специфікацій в структурі tModel. Наприклад, для того, щоб послати бізнес - партнеру RFP, запрошуюча служба повинна знати не тільки URL/месцеположення служби, але і в якому форматі повинен бути посланий RFP, які протоколи використовувати, врахувати вимоги безпеки, яку форму відгуку має на увазі відсилання RFP.

5. Протокол SOAP (Simple Object Access Protocol)

SOAP – це протокол обміну XML-повідомленнями.(Відгук від сервісу повертається SOAP- серверу, використовуючи SOAP- протокол, а це повідомлення повертається SOAP - клієнту, який послав запит).

SOAP – простий, заснований на XML протокол для обміну інформацією в децентралізованому, розподіленому середовищі. SOAP підтримує різні стилі обміну інформацією, включаючи:

- Обмін інформацією, яка формується після віддаленого виклику процедури. Цей тип обміну робить доступним процес запит - відповідь, в якому крайовий користувач отримує процедурне повідомлення і дає відповідь відповідним повідомленням.
- Інформаційний обмін на основі механізму обміну повідомленнями. Цей тип обміну використовують в організації додатків, яким потрібно обмінюватися бізнес - документами, послане повідомлення не має на увазі негайний відгук на нього.

SOAP характеризується:

- Протокольною незалежністю.
- Мовною незалежністю.
- Незалежністю від ОС і платформи.
- Підтримкою SOAP XML - повідомлень взаємодіючих частин (використовуючи багатоскладну структуру MIME).

Повідомлення SOAP складається з

- SOAP конверта, який містить дві структури даних,
- SOAP - заголовка і тіла SOAP
- інформації про імена для їх опису.

Заголовок є необов'язковою частиною, він передає інформацію про запит, визначений в тілі SOAP. Наприклад, він може містити інформацію по безпеці, ділову інформацію, або профіль користувача. Тіло містить запит Web – сервісу, або відповідь на нього.

Специфікація описує структуру і тип даних при обміні повідомленнями, використовуючи XML – схему. Спосіб, в якому SOAP використовується для посилання запитів і отримання відповідей від Web - сервіса:

- Клієнт SOAP використовує документ XML, який узгоджується із специфікацією SOAP і містить запит про послугу.
- Клієнт SOAP посилає документ серверу SOAP, а той оброблює його за допомогою HTTP, HTTPS.
- Web - сервіс отримує повідомлення SOAP, направляє його у вигляді службового запиту додатку, який надає запрошувану послугу.
- Відгук від сервісу повертається SOAP- серверу, використовуючи SOAP-протокол, а це повідомлення повертається SOAP - клієнту, який послав запит.

Питання для самоконтролю

2 Що таке координація розрізнених ресурсів?
3 На що націлені Грід - обчислення ?
4 Назвати особливості програмного забезпечення Грід
5 Стандарти, які використовуються для побудови архітектури Грід
6 Охарактеризуйте SOAP, WSDL, UDDI
7 Прокоментуйте концепцію OGSA (*Open Grid Services Architecture*)

8 Дати визначення **WSDL** та types, message, portType, binding, і service elements.
9 Що таке Web Services Inspection Language (WSIL)?
10 Назвати основні класи протоколів ресурсного рівня.
11 Що таке Universal Description, Discovery, and Integration?
12 Охарактеризуйте Протокол SOAP (Simple Object Access Protocol).

Література основна

1. Петренко А.І., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.
   Режим доступу: http://moodle.ntu-kpi.kiev.ua
2. Петренко А.І., Застосування GRID технологій в науці та освіті: додатковий матеріал до вивч. курсу для студ. спец. «Інформаційні технології проектування». − К.: НТУУ «КПІ», 2008. − 144 С.
   Режим доступу: http://moodle.ntu-kpi.kiev.ua
3. Петренко А.І., Булах Б.В., Хондар В.С. *Семантичний Грід для науки і освіти*. //– К. : НТУУ «КПІ», 2010. – 180 с.

Література допоміжна

4. Foster I., Kesselman C. The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.
5.Foster, C. Kesselman, S. Tuecke, J.M. Nick. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. - Morgan Kaufmann Publishers, 2002.
6.Foster, H. Kishimoto, A. Savva, D. Berry et al. The Open Grid Services Architecture. - Global Grid Forum, 2005
7. Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks

2.4 Забезпечення безпеки в Грід

Компонента Globus Security Infrastructure (GSI) забезпечує захист, включаючий шифрування даних, аутентифікацію (перевірка, що користувач, чи ресурс дійсно являється тим, за кого себе видає) і авторизацію (процедура перевірки, що аутентивікований користувач, чи ресурс дійсно має вимагаючі права доступу) з використанням цифрових сертифікатів X.509.

Можна впевнено стверджувати, що при використанні грід-систем інфраструктура безпеки являється найбільш важливим елементом, так як від неї напряму залежить безпека всієї системи.

Властивості системи безпеки.

Ними являються: конфіденційність, цілісність і аутентифікація. В ідеалі захищений обмін включає всі три складові.

Конфіденційність. Безпечний діалог повинен бути конфіденційним: тільки відправник і отримувач повинні розуміти зміст обміну. Якщо хтось веде прослуховування, він не повинен в діалозі виявити будь-який смисл. Це досягається алгоритмом шифрування-дешифрування.

Цілісність. Безпечна комунікація повинна гарантувати цілісність передаваємих повідомлень. Це означає, що на приймальному кінці мають бути переконані, що прийняте повідомлення в дійсності є те, яке було направлене. Слід враховувати, що зловмисники можуть перехоплювати повідомлення для модифікації його змісту, а не тільки для використання і зробити це можна навіть не розуміючи його змісту.

Аутентифікація. Безпечна комунікація повинна гарантувати, що сторини обміну є тими, за кого себе видають.

Авторизація. Авторизація відноситься до механізмів, які вирішують чи дозволено користувачу виконувати певну задачу. Авторизація зв'язана з аутентифікацією, тому що потрібно пересвідчитися, що користувач є тим, за кого він себе видає перш ніж вирішувати допускати його до роботи.

Криптографія з відкритим ключем. Забезпечення безпеки в Грід є однією з головних задач і в документації по Грід питанням безпеки відводиться близько 30%. Системи з одним ключем в Грід недопустимі із-за необхідності передачі отримуючі стороні ключа, який тим самим стає доступним третім особам. В грід використовується системи шифрування з відкритим ключем, котрий не потрібно передавати по каналу зв'язку.

Цифровий підпис. Цілісність в системах з відкитим ключем гарантується цифровим підписом. Цифровим підписом є блок даних, який поєднується з повідомленням для вияснення невтручання в повідомлення під час передавання. Цифровий підпис для повідомлення генерується в два кроки:

- спочатку генерується цифрове резюме повідомлення: це певна сума повідомлення. Вона завжди менша за повідомлення і навіть найменші зміни в повідомленні дають інше резюме.
- резюме шифрується закритим ключем відправника. Результуюче зашифроване резюме повідомлення називається цифровим підписом.

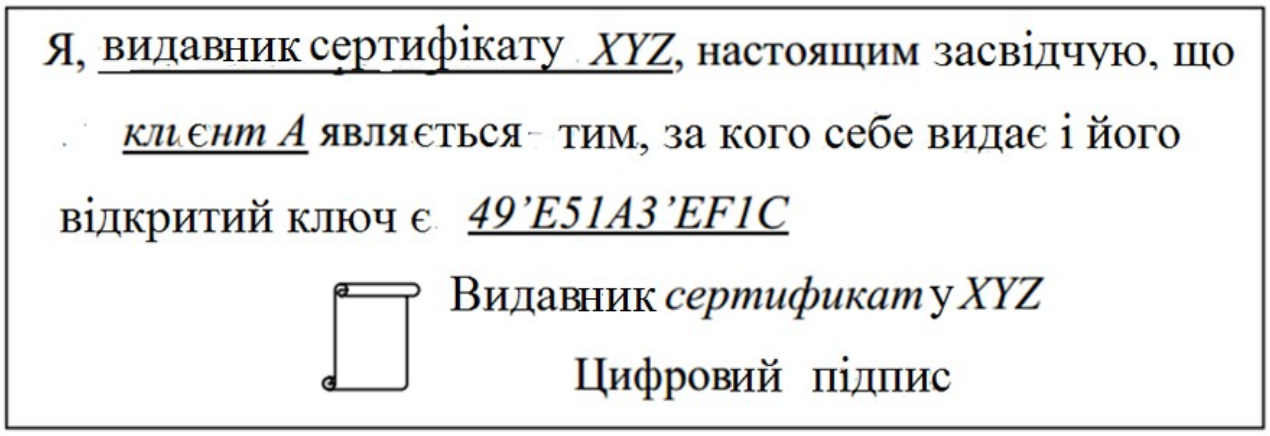
Цифровий підпис з'єднується з повідомленням і відправляється адресату. Адресат після цього робить наступне:

- використовуючи відкритий ключ відправника, дешифрує цифровий підпис.
- використовує той же самий алгоритм шифрування цифрового резюме, що і відправник, щоб отримати цифрове резюме прийнятого повідомлення.
- порівнює обидва резюме. Якщо між ними є хоч найменша розбіжність, значить було втручання третьої сторони. Таким чином гарантується і цілісність.

Аутентифікація в системах з відкритим ключем

Наведений вище приклад в деякій мірі гарантує аутентичність відправника, оскільки тільки відкритий ключ відправника може дешифрувати цифровий підпис. Але реально відправник може бути не тим, за котрого він себе видає. В багатьох випадках «слабкої аутентифікації» достатньо. Але часом потрібна аутентифікація, при котрій нема сумнівів в відправнику. Це досягається на

основі цифрових сертифікатів. Цифровий сертифікат є цифровий документ, який засвідчує, що певний відкритий ключ належить володарю закритого ключа. Цей документ підписується третьою стороною , яка називається certificate authority (CA). Формат CA представлений нижче.



Формат цифрового сертифікату

Звичайно, сертифікат кодується в цифровому форматі. Підпис видавця є в дійсності цифровим підписом, згенерованим за допомогою закритого ключа CA, тому ми можемо провірити цілосність сертифікату, використовуючи відкритий ключ CA. Наявність сертифікату позволяє виконати аутентифікацію. Якщо ви підписуєте ваше повідомлення вашим закритим ключем і посилаєте адресату копію вашого сертифікату, то він може бути впевненим, що повідомленя було послано вами.

X.509 формат. Сертифікат є текстовий файл, який включає багато інформації, представленої в специфічному синтаксисі. X.509 має наступні чотири найбільш важливі речі:

• Subject: це «ім'я» користувача. Воно кодується як унікальне ім'я.
• Subject's public key: Це включає не тільки сам ключ, але і певну інформацію, наприклад, який алгортм використаний для генерації відкритого ключа.
• Issuer's Subject: уникальне ім'я CA.
• Digital Signature: Сертифікат включає цифровий підпис всієї інформації в сертифікаті. Цифровий підпис генерується з використанням відкритого ключа CA.

CA ієрархія. Системи з відкритим ключем будуть мати список всіх CA, котрим можна довіряти, включаючи і ваш CA. Ви повинні вирішити, хто зробить такий список.

Хто підписує CA сертифікати? Відповідь – інший CA. Це позволяє створити ієрархію CA, в котрій ви можете довіряти самому верхньому CA. Ви можете по ступеням ієрархії провірити всі CA, крім останнього, котрому вам приходиться довіряти абсолютно. Сертифікат - відкритий документ, який доступний іншим користувачам для перевірки вашої ідентичності. Об'єднання відкритого і закритого ключів називається в загальному випадку мандатом (credentials) користувача.

Система входу користувача в систему досить складна. Це визначено багатьма факторами, але головною проблемою являється вирішення питання безпеки (загрози вторгнень і атак зловмисників). Аутентифікація і авторизація користувачів являються шляхом для рішення цієї проблеми. Аутентифікаційні рішення для середовищ віртуальних організацій повинні мати наступні властивості:

• Єдиний вхід. Користувач повинен реєструватися і аутентифікуватися тільки один раз на початку сеансу роботи, отримуючи доступ до всіх доступних ресурсів базового рівня архітектур Грід.

• Делегування прав. Користувач повинен мати можливість запуску програм від свого імені. Таким чином, програми отримують доступ до всіх ресурсів, на котрих авторизований користувач. Користувацькі програми можуть, при необхідності, делегувати частину своїх прав іншим програмам.

• Довірливе відношення до користувача. Якщо користувач запросив одночасну роботу з ресурсами декількох постачальників, то при конфігурації захищеного середовища користувача система безпеки не повинна вимагати взаємодії постачальників ресурсів одного з одним. Для входу в Грід-систему користувач повинен:

• бути легальним користувачем обчислювальних ресурсів в своїй організації;

• мати персональний цифровий сертифікат, підписаний центром сертифікації;

• бути зареєстрованим хоч би в одній віртуальній організації.

Інфраструктуру безпеки в дії можна продемонструвати наступним рис.2.4.1, де питання безпеки поставлено таким чином: “Створити процеси на вузлах A і B, які потім обмінюються файлами з вузлом C.”

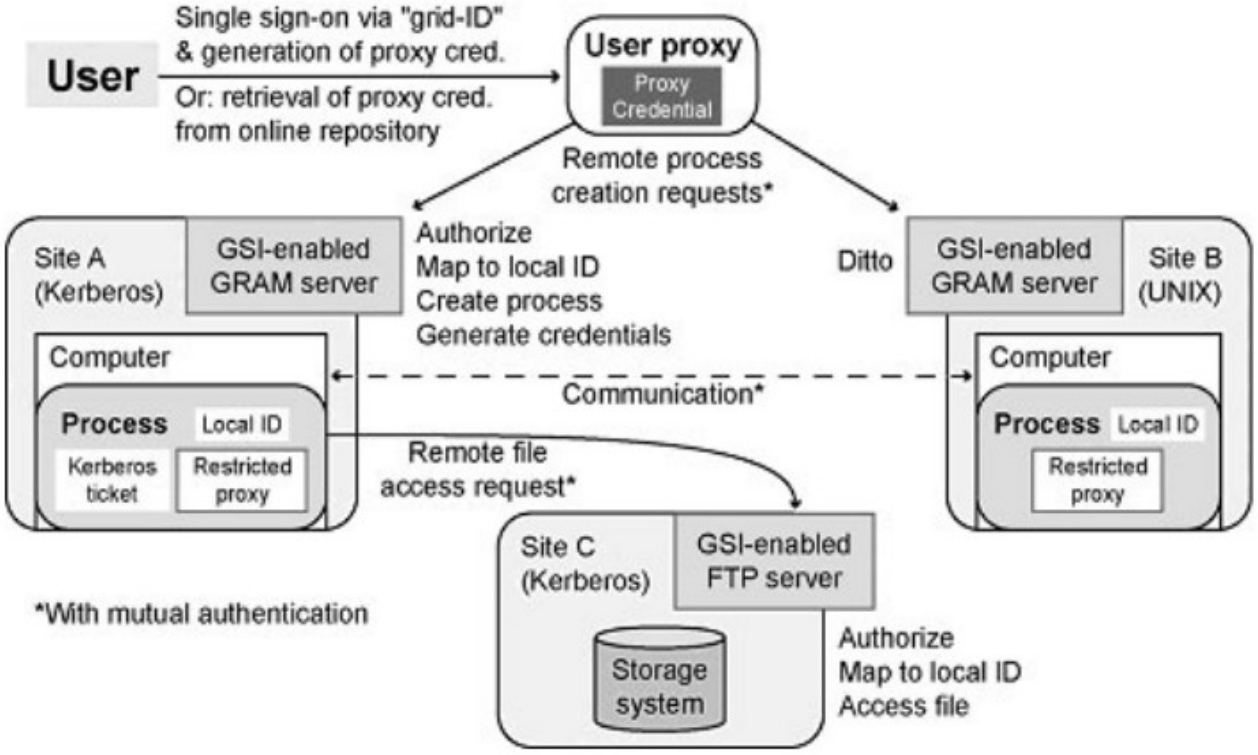


Рис.2.4.1 Інфраструктуру безпеки

Питання для самоконтролю

1. Назвіть особливості програмування Грід.
2. Призначення проміжного програмного забезпечення.
3. Призначення системи безпеки Грід.
4. Що таке Globus Security Infrastructure (GSI)?
5. Назвати основні властивості системи безпеки Грід.
6. Що таке конфіденційність, цілісність і аутентифікація?
7. Що таке авторизація?
8. Що таке цифровий підпис?
9. Охарактеризуйте аутентифікацію в системах з відкритим ключем.
10. Що таке Цифровий сертифікат?
11. Що таке X.509 формат?
12. Хто підписує CA сертифікати?
13. Що таке Єдиний вхід і делегування прав?

Література основна

1.Петренко А.ИИ., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.

Література допоміжна

2.Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с.
3.Computational problems - Gridcafe. E-sciencecity.org. Retrieved 2013-09-18.
4.Foster, Ian; Carl Kesselman (1999). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers. ISBN 1-55860-475-8.
5.www.globus.org
6.www.glite.org
7.www.unicore.org

2.5 ***Програмування та особливості планування в Грід-системах***

1. Суперкомп'ютери, грід-системи та хмарні середовища

Наразі суперкомп'ютери, грід-системи та хмарні середовища є найпотужнішими обчислювальними засобами. Але користувачі часто не усвідомлюють, якого типу застосунки слушно виконувати на одному з цих засобів. Тому далі розглянемо особливості та відміни цих архітектур з точки зору найбільшої придатності для реалізації застосувань.

1.1. Порівняння Грід-систем та комп'ютерних кластерів

Під Грід-системою розуміють обчислювальну систему з декількох обчислювальних елементів (СЕ), зв'язаних мережею, які виконують паралельно складне завдання. Така обчислювальна система має три характерні ознаки: її СЕ слабо позв'язані, можуть бути розташовані географічно на великій відстані та можуть бути різного типу (гетерогенними). На відміну від Грід-системи, система з тісно зв'язаних СЕ, така як комп'ютерний кластер (КК), як правило, має пам'ять, що розділяється між СЕ та інші загальні ресурси, які зв'язані між собою через швидкісну систему коротких міжз'єднань.

Як правило, слабко зв’язана система будується з автономних СЕ за принципом майстер - виконавці. Причому СЕ-майстер розділяє велике завдання на незалежні підзавдання, розподіляє їх серед СЕ-виконавців, запускає їх на виконання, збирає від них результати та обробляє їх для формування кінцевого результату.

Грід-система – це система з слабко позв’язаних СЕ. Ці СЕ можуть виконувати окремі незалежні завдання, допоки операційна система (ОС) не об’єднає їх для виконання функції Грід-системи. Причому такий СЕ може продовжувати виконання свого автономного завдання у багатопотоковому режимі, виконуючи функції частини Грід-системи. Цей слабкий зв'язок між СЕ дає змогу географічно віддаленного розміщення СЕ. Оскільки СЕ зв’язані між собою через стандартну мережу (найчастіше - Інтернет), дійсно, не має значення відстань між цими СЕ. Така географічно розподілена мережа має ряд переваг. Загальне завдання у такій мережі вирішується менш витратно, оскільки вона дає змогу краще завантажити недовантажжені СЕ. Ця мережа не має проблем з електричним живленням, оскільки різні СЕ живляться від різних електричних мереж, які не здатні відмовити одночасно. СЕ мережі можуть мати різні ОС та різні продуктивності, хотя багато з Грід-систем побудовані на однотипних СЕ.

Високопродуктивні суперкомп'ютери аналогічні Грід-системам в тому, що серед них багато зв’язаних систем, як наприклад, КК. Але ця зв’язаність є щільною. Якщо між СЕ у Грід-системі передача даних відбувається як асинхронна пересилка повідомлень з підтвердженням їх приймання, то у суперкомп’ютерах ця пересилка відбувається синхронно без очікування такого підтвердження. Через це СЕ суперкомп’ютерів мають бути зконцентровані в одному місці і з’єднані між собою швидкодіючими каналами зв’язку, а не через звичайну локальну мережу чи Інтернет. Причому ці СЕ мають бути однорідними.

1.2 Програмування Грід-систем та комп’ютерних кластерів

Відміни між архітектурами КК та Грід-систем роблять суттєво різними технології створення програмного забезпечення для них. Тому досі створення методів розробки матзабезпечення для цих архітектур залишається в галузі досліджень та розробки.

Однією з ключових концепцій в написанні застосувань для цих архітектур є ідея батьківських і дочірніх процесів, яка полягає у наступному. Велике завдання аналізується на можливість розділення його на дрібніші завдання. Диспетчер розподіленого обчислювального середовища розділяє завдання на частини, які розподіляються серед вільних СЕ. Перш ніж продовжити обробку, головна задача ("батько"), чекає дані, які повинні бути повернуті від кожного з дрібних підзадач («дітей»). Найбільш важливою проблемою при програмуванні Грід-систем чи КК є кодування у програмі способу, який розділяє велике завдання на дрібні під завдання так , щоб дочірні процеси завершувались якомога швидше. Таке програмування є досить складним. Найбільшою проблемою є нова архітектура, для якої програміст повинен думати в термінах батьківських та дочірніх процесів, які розподіляються по СЕ при певних

умовах. Тут програміст має турбуватись не про зацикленість програми, а про баланс кількості та складності дочірніх процесів та кількості вільних ресурсів. Може здаватися, що Грід-система та КК мають однакові архітектури з точки зору програмування, але Грід-система є більш складною для програмування через асинхронність пересилок даних, які мають різні затримки. Крім того, слід мати на увазі, що як канали зв'язку, так і деякі СЕ Грід-системи можуть вимикатись на тривалий час. Тому користувацька програма повинна мати властивість динамічно змінювати хід свого виконання.

1.3 Планування завдань в Грід-системах та комп'ютерних кластерах

Як Грід-система, так і КК мають у своїх ОС планувальники завдань. Планувальник повинен мати не тільки інформацію про програми, які запускаються, але й про алгоритми, які керують породженням дочірніх процесів, про наявність вільних ресурсів в кожній ділянці обчислювального середовища, кількість яких може рахуватися сотнями та тисячами. Крім того він займається перевіркою аутентичності завдань, розподілом їх по ресурсах і фактичним ходом їх виконання.

В області високопродуктивних обчислень часто використовується планувальник, створений у Job Submission Description Language Working Group (JSDL-WG) та Maui Scheduler. Причому перший набув поширення серед Грід-систем, а другий – серед КК.

JSDL-WG - це мова управління завданнями, яка орієнтована на Грід-системи та є незалежною від будь-якої мови програмування, яка може бути використана користувачем. У описі JSDL можна використовувати код XML, щоб визначити різні аспекти запиту ресурсів, що робить його відкритим стандартом. Це поняття відкритого доступу є критичним при розгляданні переходу від програмування Грід-систем до програмування хмарних обчислень.

Планувальник Maui Scheduler був розроблений в 1990-х роках як планувальник з відкритим вихідним кодом для ОС високопродуктивних КК, які використовують ОС Unix. Maui Scheduler дозволяє адміністраторам встановлювати розклад і пріоритети, резервувати ресурси для конкретних програм, або їх частин, а також виконувати інші завдання, які роблять системи більш функціональними.

Після поширення Maui Scheduler була розроблена в Центрі високопродуктивних обчислень Університету Юта його комерційна версія Moab. Вона додатково підтримує різні ОС, у тім рахунку Linux, Mac OS X та Windows. Також вона має такі функції, як менеджер робочих навантажень, менеджер кластеру, порталу доступу і керування ресурсами, які не доступні в Maui Scheduler. Ця тенденція, коли за основу береться програмне забезпечення з відкритим кодом і дороблюються програми, які додають можливостей, можна побачити у розвитку хмарних обчислень. У планувальнику Moab Scheduler завдання розглядаються з урахуванням політики справедливості пріоритетів, яка враховує тривалість знаходження в черзі, можливостей розширення завдання, історію використання СЕ користувачем і яка конфігурується адміністратором КК. Планувальник Moab Scheduler дає дозвіл, щоб певні СЕ були зарезервовані для певних користувачів, груп, рахунків, або проектів, що

мінімізує кількість витрачених ресурсів комп'ютера, з урахуванням очікуваного простою.

Зараз Moab Scheduler використовується як інструмент для дослідження обчислювальних процесів в КК. Цей планувальник збирає велику кількість статистичних даних і може проаналізувати цю інформацію, щоб визначити ефективність того чи іншого підходу до планування, використання наявних ресурсів[1].

1.4 Особливості планування у Грід-системах

Перед тим, як розглядати обчислювальні процеси у хмарному середовищі, слід ознайомитись з процесом планування завдань в обчислювальній мережі Грід-систем. Слід зазначити, що велику роботу по стандартизації планування Грід-систем зроблено на форумі Open Grid Services Architecture (OGSA)[2]. Також значний внесок зробила фірма IBM під час розробки КК *Deep Blue*, яка випустила документацію в серії Red Book.

Процес планування в Грід-системі можна розділити на три етапи, які мають 11 кроків і описуються нижче.

Етап I: Визначення ресурсів

Крок 1: Фільтрація авторизуючих запитів.

Перед тим, як розмістити своє завдання у Грід-системі, користувач повинен пройти аутентифікацію. Крім того, інформація, необхідна для аутентифікації, включає в себе профіль користувача, що описує ресурси, які можуть бути використані, інформацію про оплату використання ресурсів. Цей профіль може також включати в себе деякі облікові дані для одного, або декількох ресурсів, які підключаються до мережі. Код одноразового доступу, як правило, перевіряється на початковому етапі фільтрації авторизуючих запитів, дозволяючи обліковим даним, які зберігаються у одному СЕ в якості аутентифікаційних документів, бути застосованими для інших СЕ в рамках кластеру.

Крок 2: Визначення застосування.

На цьому етапі користувач вказує мінімальний набір ресурсів, які розглядає планувальник. Ця інформація включає в себе об’єм необхідної пам’яті, робочого простору (місця на дисках), перелік спеціалізованих бібліотек тощо і описується як скрипт для планувальника. Вона потрібна для того, щоб планувальник знав, які ресурси необхідні для завдання, щоб воно було виконано успішно.

Поширення стандартного опису XML, який використовується для обміну даними, зробило опис застосувань більш стандартизованим та відкритим для більшості кластерів і мереж. Такий опис не потрібно виконувати вручну, бо він, як правило, формується автоматично за допомогою відповідних додатків до Грід-системи.

Крок 3: Виділення мінімальних вимог до середовища.

В залежності від ресурсів, заявлених авторизованим користувачем, планувальник намагається знайти перший вільний слот (СЕ та інтервал часу), у якому буде виконуватися завдання. Незалежно від завдання і розміру КК, цей крок пов’язаний з вирішенням задачі пошуку відповідності між потребами в

ресурсах і наявними ресурсами, щоб задовольнити потреби користувачів у встановлені терміни.

Етап II: Вибір системи CE

Крок 4: Збір інформації.

Хоча оптимальне рішення задачі призначення завдання на певний набір апаратних засобів може бути знайдене за одним з відомих алгоритмів, але місцеві правила можуть змусити користувача використати інший набір ресурсів. Наприклад, для університетської Грід факультети університету, здавалось, мають рівноправний доступ до Грід-ресурсів. Але часто окремі факультети, які , наприклад, виконують роботи по удосконаленню системи, можуть мати більше прав доступу і їм планувальник буде виділяти більшу квоту.

Слід зазначити, що існує можливість для зміни ресурсів до того моменту, коли завдання буде представлене для вирішення. Значна кількість досліджень робиться у галузі прогнозування розподілу ресурсів, щоб створити адаптоване виділення ресурсів, що дозволить збільшити точність прогнозу цих ресурсів.

Крок 5: Вибір системи CE.

Алгоритми, які вирішують, яку множину CE використовувати для конкретного завдання, варіюються по складності. Але у багатьох проектах розробки планувальників використовується одна і та сама тенденція. Це використання методів, основаних на алгоритмах пошуку паропоєднань Condor, обчислювальної економіки та багатьох інших.

Алгоритм пошуку паропоєднань Condor заснований на класичній схемі оголошень. Обчислювальні ресурси, - це за анологією - "продавці", які рекламують свої послуги та можливості через клас ClassAds, де вони можуть також вказати запитувану ціну для ресурсу. Користувачі та завдання - то "покупці" , які можуть створювати власні класи ClassAds, де зазначаються їх вимоги до ресурсів та заявки оплати, які вони готові платити, щоб їхні вимоги були виконані. Алгоритм Condor перебирає класи ClassAds від продавців і покупців і робить кращі паропоєднання, на основі вказаних специфікацій. Пошук паропоєднання є одноразовим статичним процесом, хоча у будь-який момент за алгоритмом Condor може відбутися повторне сканування заявок та ресурсів, щоб визначити, чи можна знайти кращі паропоєднання.

Алгоритми обчислювальної економіки включають в себе різні методи, які використовуються, щоб вирішити, які процеси призначити на певні ресурси. Ці методи запозичені з теорії класичної економіки, щоб усунути нестабільність вирішення завдання призначення, яка виникає при певних обставинах. Після застосування алгоритму обчислювальної економіки, ресурси і завдання можуть бути динамічно переоцінені в залежності від попиту на ресурси і попиту на результати обчислень, а центральний планувальник постійно приймає рішення щодо запуску завдань, які ґрунтуються на останніх оцінках. Система планування на основі алгоритмів економіки повинна мати на увазі що в будь-який момент ресурси мають використовуватися найбільш ефективно, а саме завдання повинне виконуватись з найбільшою ефективністю.

Етап III: Виконання завдання

Крок 6: Попереднє резервування ресурсів

Попереднє резервування ресурсів у Грід-системі може бути таке саме за складністю, як організація об'єднання комп'ютерів через локальну мережу, або як вирішення задачі масового планування. Але більшість користувачів замовляє ресурси у Грід-системі з надлишком, тобто «про всяк випадок» і це також слід враховувати.

Крок 7: Представлення завдання

Часто завдання користувача представлене з розбіжностями відносно стандартів, на які налаштована дана Грід-система. Тому представлення такого завдання потребує узгодження. Через це Грід-системи, основані на відкритому матзабезпеченні, завжди будуть позаду комерційних Грід-систем через недостатню увагу до стандартизації оформлення завдань.

Крок 8: Підготовка завдання

Як тільки починається виконання завдання, ресурси, такі як набори даних будуть переміщені в тимчасові файли і батьківський процес починає заповнювати інформацію для кожного дочірнього процесу. При цьому для розсилання даних використовуються FTP (File Transfer Protocol), SCP (Secure Copy Protocol), і SFTP (Secure File Transfer Protocol), а також протокол Torrent.

Крок 9: Моніторинг виконання завдання

Може так статись, що якийсь дочірній процес "зависне", наприклад, через виключний стан, відсутність вхідних даних з відповідним простоюванням ресурсів. У гіршому випадку така поведінка може тягнути за собою "зависання" усього завдання. Тому планувальник повинен відслідковувати подібні ситуації і переривати виконання завдання.

Крок 10: Завершення роботи

Бідь-яке завдання має метою розрахувати певного роду результати і припинити своє виконання, що має бути зафіксовано планувальником.

Крок 11: Очищення середовища

Планувальник турбується про стирання тимчасових файлів у разі, коли завдання користувача не зробила це самостійно.

1.5 Перехід від Грід-систем, комп'ютерних кластерів до хмарних середовищ

Є багато причин, чому вибирають для обчислення задачі саме Грід-систему, а не КК. Це може бути економія витрат, або більша доступність для використання. Є чимало задач, які за своїми розмірами не поміщаються у КК, але можуть бути виконані у Грід-системі. З іншого боку, володіння КК дає простір для реалізації великої множини паралельних алгоритмів, які обчислюються у Грід-системі неефективно. На певному кроці розвитку як Грід-систем, так і КК, стало ясно, що доцільно сумістити ці архітектури з концепцією віртуальних процесорів.

Вперше пішла мова про ідею хмарних обчислень на конференції LinuxWorld 2003. Тоді звернули увагу на систему Scyld, яка була призначена для реалізації сайтів та бізнес-застосувань. Її особливістю була та, що будь-який потік запитів до сайту завжди обслуговувався за рахунок того, що призупинялись малопріоритетні процеси, а важливі процеси динамічно переміщувались на ресурси КК, що вивільняються. Відміною від ОС паралельних систем була та, що система Scyld могла виконувати балансування

навантаження великої кількості різних СЕ, які мають власні ресурси та ОС різного типу.

Як і багато нових технологій, хмарні обчислення розвивались поетапно. На першому етапі хмарні обчислення розвивались як нова послуга - Software as a Service (SaaS), тобто матзабезпечення як послуга в рамках Інтернет - стандарту Web 2.0. Почало з'являтись багато хостинг-сайтів, які були призначені для зберігання сайтів, веб-застосунків осіб та компаній. З'явились системи керування контентом (content management systems, CMS) і відповідно, традиційні сайти почали масово замінюватись на сайти на базі CMS. Тобто, тут CMS є типовим матзабезпеченням як послуга на віддаленому сервері. Крім того, швидке зростання вартості електроенергії змушує багато фірм та організацій мінімізувати витрати, в тому числі використовуючи віддалений доступ до сервера.

Подальші етапи будемо розглядати в розділах хмарних технологій.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення Грід.
2, Перерахуйте особливості програмування Грід.
3. Що таке координація розрізнених ресурсів?
4. На що націлені Грід - обчислення ?
5. Назвати обчислювальні ресурси Грід.
6. Що таке Сумісніять midlleware Грід.
7. Призначення Middleware.
8. Порівняйте суперкомп'ютери та грід-системи.
9. Назвіть відміни між архітектурами КК та Грід-систем.
10. В чому полягає ідея батьківських і дочірніх процесів?
11. Особливості програмування Грід-систем та комп'ютерних кластерів.
12. Охарактеризуйте особливості алгоритмів планування розміщення завдання в Grid і комп'ютерних кластерах.
13. Планування завдань в Грід-системах та комп'ютерних кластерах.
14. Що таке Фільтрація авторизуючих запитів?

Література

1. CHPC Software: Moab Scheduler . Режим доступу: www.chpc.utah.edu/docs/manuals/software/moab.html
2. Towards Open Grid Services Architecture. Режим доступу: http://toolkit.globus.org/ogsa/
3. IBM Redbooks. Режим доступу: www.redbooks.ibm.com
4. Chee B.J.S., Franklin C.Jr. Cloud Computing. Technologies and Strategies of the Ubiquitous Data Center. CRC Press: Taylor & Francis Group. - 2010. -263 p.

2.6 **Програмні рішення Грід**

В Грід використовуються основні елементи технології WebServices. Це позволяє оформити функції Грід-технлогії (службові і ресурсні) в вигляді веб-сервісів, використовуючи для роботи з ними всі стандартні технології WebServices. На рис.2.6.1 схематично показаний простий сервісно-

орієнтований грід, в якому сервіси використовуються і для віртуалізації і для забезпечення інших функціональних можливостей Грід.

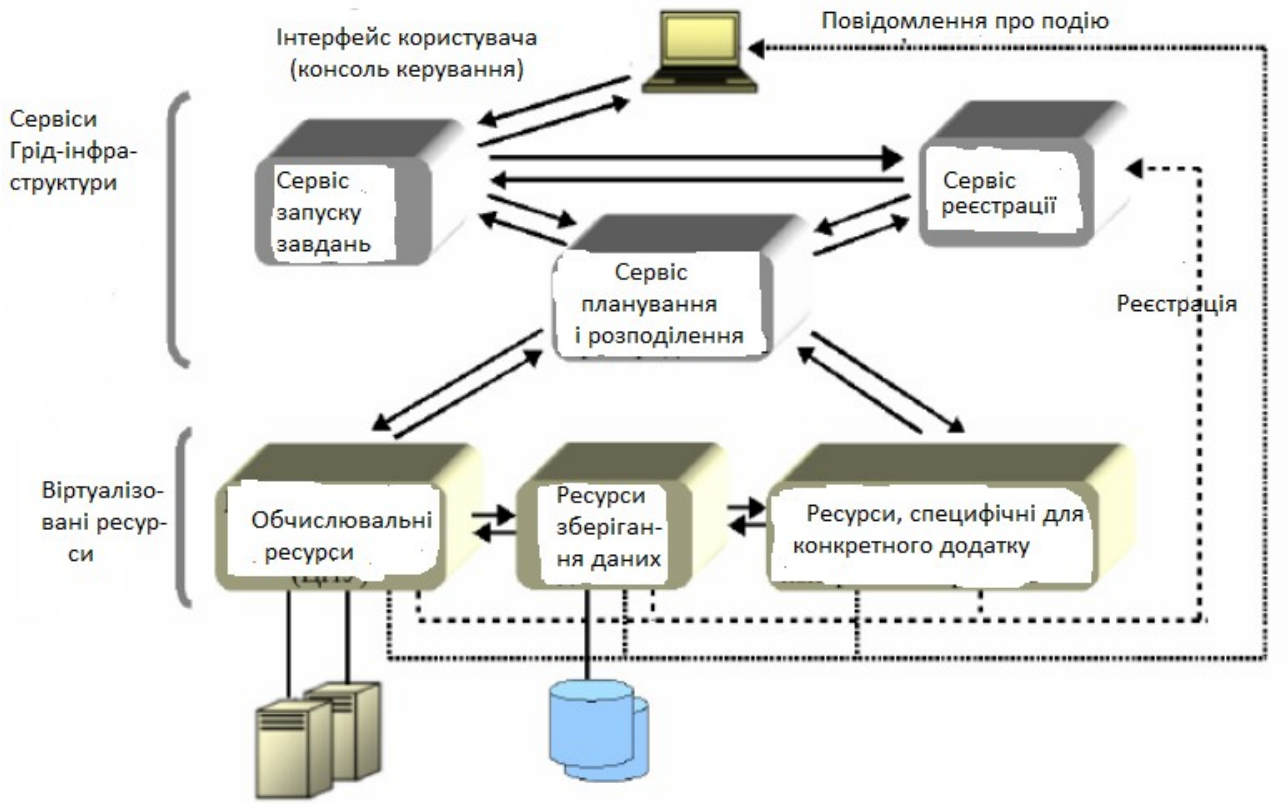


Рис.2.6.1 Схема сервісно-орієнтованого грід

На схемі показана єдина консоль і для запуску завдань в Грід і для керування Грід-ресурсами. Програмне забезпечення інтерфейсу користувача (консолі) звертається до сервісу реєстрації для отримання інформації про існуючі Грід-ресурси. Після цього користувач за допомогою консолі входить в контакт з сервісами, які представляють (віртуалізують) кожний ресурс, щоб запросити періодичне отримання даних про роботу ресурсів і отримати повідомлення про суттєвих змінах в їх стані (наприклад, якщо ресурс стає недоступним, чи сильно завантаженим).

Користувач направляє запит на запуск в службу запуску, яка передає запит службі розподілення завдань (яку ще називають планувальником). Ця служба контактує зі службою, яка представляє додаток і запрошує інформацію про вимоги до ресурсів виконання завдання. Далі служба розподілення запитує (направляє запит) в служби реєстрації інформації про всі необхідні ресурси в Грід і напряму контактує з ними, щоб пересвідчитись в їх доступності.

Якщо необхідні ресурси доступні, планувальник вибирає найкращу доступну сукупність ресурсів і передає інформацію про них сервісу додатку з запитом на початок виконання. В іншому випадку планувальник ставить завдання користувача в чергу і виконує його, коли необхідні ресурси стають доступними. Коли виконання завдання закінчується, сервіс додатку повідомляє про результат планувальнику, який повідомляє про це сервіс запуску завдань. Сервіс запуску завдань, в свою чергу, увідомлює користувача. Слід зауважити, що наведений приклад дуже спрощений: функціонування реального Грід набагато складніше.

В середовищі Грід явним чином присутні наступні елементи:

- Програми користувача
- Ресурси (“залізо”, ОС, кластерне ПЗ та ін.)
- Проміжне програмне забезпечення (Middleware), яке виступає в ролі посередника між користувацькими програмами і ресурсами.

Middleware включає великий обсяг програмного забезпечення, яке створюється великими організаціями і строго стандартизується, щоб забезпечити його використання різними користувачами. В число найбільш відомих пакетів middleware входять:

•Globus Toolkit. Він вважається американським напрямком;

•gLITE. Являється європейським проектом і підтримується Європейським центром ядерних досліджень (CERN).

•UNICORE. Теж являється європейським проектом.

Вони розрізняються між собою, але мають багато спільного, оскільки використовують, як правило, систему Globus Toolkit.

Різні middleware–системи можуть розрізнятися по рівню. Наприклад, gLITE ставиться поверхи Globus Toolkit.

В глобальних Грід-системах в якості middleware використовують інструментарій Globus Toolkit, розроблений американськими вченими, який став de facto світовим стандартом. Він включає в себе спеціальний протокол на основі HTTP для використання обчислювальних ресурсів GRAM (Grid Resource Allocation Management); розширену версію протоколу для передачі файлів GridFTP; службу безпеки GSI (Grid Security Infrastructure); розподілений доступ до інформації на основі протоколу LDAP; віддалений доступ до даних через інтерфейс GASS (Globus Access to Secondary Storage). Ці служби дозволяють побудувати повнофункціональну Грід- систему.

Далі коротко розглянемо розроблене і використовуємe middleware Грід.

Програмне вирішення GLOBUS

Набір програм Globus Toolkit (GT) – програмний продукт з відкритим початковим кодом і набором бібліотек, розроблений в США в національній лабораторії. Він містить набір стандартних блоків і інструментів, які можуть бути використані розробниками і системними інтеграторами. За декілька років вийшли кілька версій програми Globus: оригінальна – в кінці дев'яностих, GT2 – в 2000, GT3 – в 2003, і GT4 – в 2005, GT5, GT6. Версія GT2 послужила базисом для безлічі Грід-розробників по всьому світу. GT3 – стала першою повноцінною реалізацією інфраструктури Грід, побудованої на технології Web - сервісів, з використанням проміжної ланки GGF’s OGSI. GT4 – перша версія, повністю сумісна з основними Web - сервісами так само, як GRID - сервіси засновані на WSDL і WSRF. Інструментарій підтримує операційні системи Linux (RHEL 5-7, CentOS 5-7, Debian 6-7 та ін.), MacOS (10.6 та новіші) та Windows (Windows 7 та новіші).

В стабільному релізі GT 4 Final зафіксовані всі зовнішні інтерфейси.

На сайті Globus Alliance приводиться опис складу служб, відомості про них, плани і стан реалізації, але підтримка завершилася в січні 2018 року, натомість користувачам пропонується використовувати платформу Globus Connect [4]).

На рис. 2.6.2 приведені компоненти GT 4.

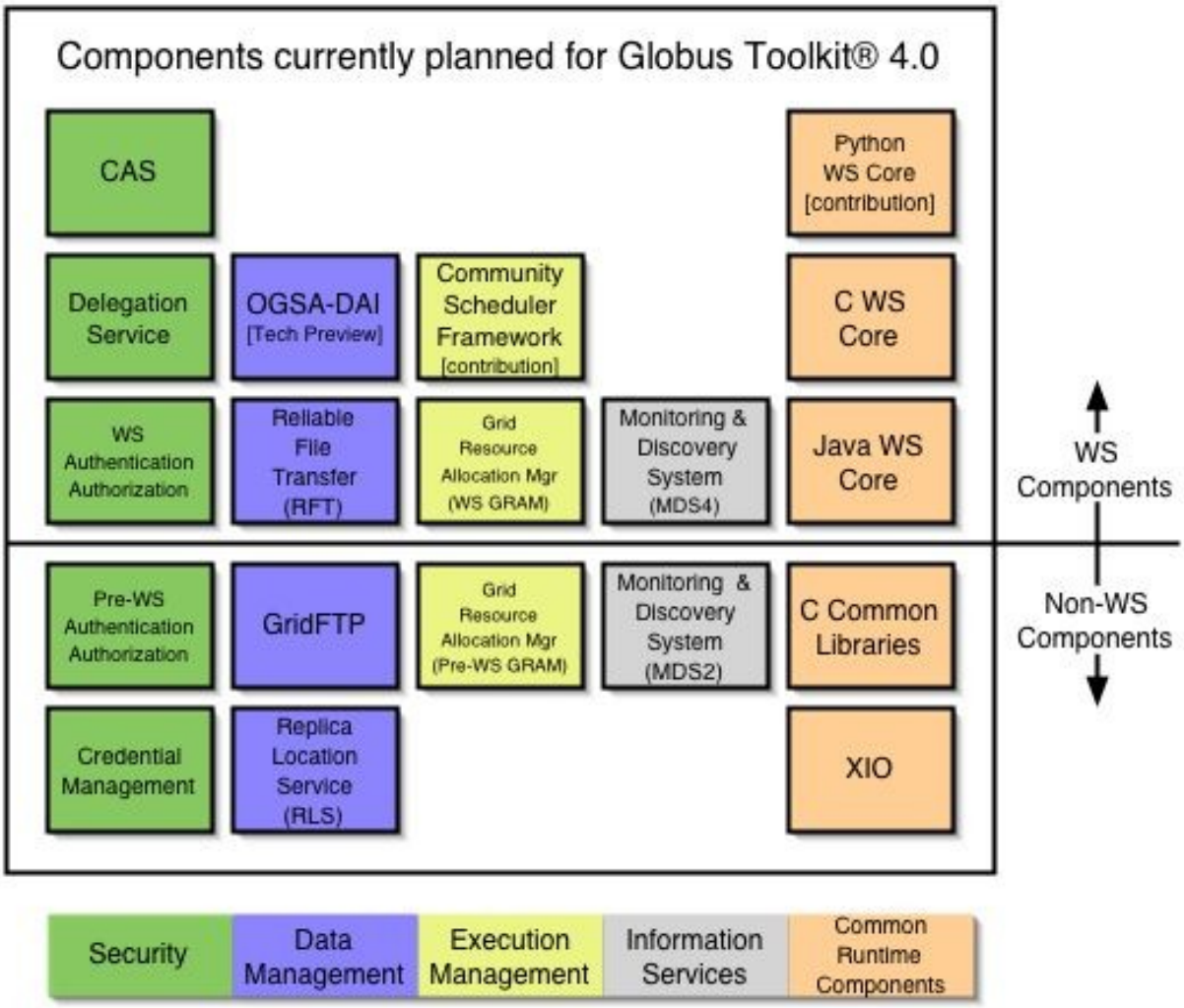


Рис. 2.6.2. Компоненти Globus Toolkit GT4

Для управління виконанням пакет надає можливість виявлення і управління ресурсами Грід, управління робочим простором і засобом планувальника співтовариства користувачів.

Для забезпечення безпеки пакет надає сервіси аутентифікації і авторизації, надання віддаленого доступу і авторизації співтовариств користувачів.

Для управління даними в програмі закладені функції надійної передачі файлів, інтеграції і доступу до даних і їх тиражування. Для забезпечення інформаційних служб, в програму закладені функції моніторингу і виявлення різних сервісів системи. Для підтримки спільної роботи система містить різні ядра Web – сервісів, бібліотеки і розширені функції підтримки введення/виводу.

GT4 містить набір стандартних служб:

1. Управління завданнями: Пакет програм виявлення і управління ресурсами (GRAM);.
2. Надійна файлопередача (RFT);.
3. Делегування функцій.

4. Система моніторингу і виявлення вільних ресурсів – індекс (MDS - index);
5. Система моніторингу і виявлення – MDS - trigge.
6. Система моніторингу і виявлення – збір даних (MDS - aggregate);
7. Авторизація співтовариства (CAS);
8. Інтеграція і доступ до даним (OGSA - DAI);
9. Робота в реальному масштабі часу.

Слід зазначити, що Globus Toolkit не містить брокера ресурсів, залишаючи завдання його реалізації розробникам, що створюють системи Грід на його основі.

Склад Globus Toolkit 5.0 (GT5) представлений на рис.2.6.3.

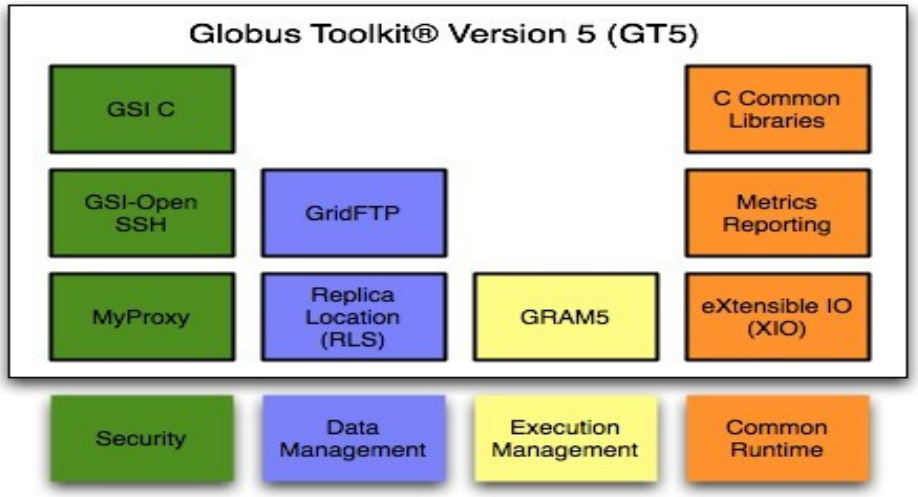


Рис.2.6.3. Склад програмного забезпечення GlobalToolkit GT5

У порівнянні з версією 4, версія 5.0 зазнала наступних основних змін [3]:
- вдосконалення підтримки управління кореневими сертифікатами у компоненті MyProxy v5.0;
- клієнт GSI-OpenSSH типово намагаються виконувати лише автентифікацію GSI (при цьому підтримка інших методів зберігається);
- в RLS досконалена підтримка 64-бітних операційних систем, сумісність із специфікаціями ODBC при збереженні повної сумісності з GT 4 RLS;
- GRAM5, у порівнянні з GRAM2 та GRAM4, є одночасно надійним та масштабованим, при цьому збережена сумісність з GRAM2;
- покращена система журналювання в GRAM5, знижено споживання ресурсів щонайменше в 10 разів.

Версія GT6 зберігає сумісність з версіями 5.0 та 5.2 на рівні протоколів взаємодії, джерельного коду (API) та двійкових програмних інтерфейсів (ABI). Основні компоненти:
- Grid Security Infrastructure in C (GSI C) – надає інтерфейси прикладного програмування та інструменти забезпечення автентифікації, авторизації та управління сертифікатами;
- MyProxy – програмне забезпечення управління інфраструктурою публічних ключів X.509 (PKI) облікових записів (сертифікати та приватні ключі);
- GridFTP – високоефективний, безпечний та надійний протокол передачі даних, що є розширеним варіантом протоколу FTP;

- Grid Resource Allocation and Management (GRAM5) – компонент, що розподілення, моніторинг та управління завданнями в ресурсах мережі Grid;
- GSI-OpenSSH – модифікована версія OpenSSH, яка надає підтримку автентифікації та делегування з використанням X.509 проксі-сертифікатів;
- C Common Libraries – загальні бібліотеки для мови C;
- XIO – розширювана бібліотека введення/виведення написана мовою C, яка надає єдиний інтерфейс прикладного програмування (open/close/read/write) з підтримкою різноманітних протоколів взаємодії додатків. Підтримка протоколів інкапсульована у вигляді "драйверів", до стандартних входить підтримка: TCP, UDP, файли, HTTP, GSI, GSSAPI_FTP, TELNET та черги;
- SimpleCA – надає спрощену реалізацію центра сертифікації, який може видавати сертифікати X.509 для користувачів та служб Globus Toolkit.

Система Gridge

Gridge - програмний продукт PSNC з відкритим початковим кодом, покликаний допомогти користувачам у використанні проміжного ПО Грід і в створенні ефективних інфраструктур Грід. Всі програмні компоненти системи Gridge були сполучені в єдину розподілену систему, що працює по єдиних інтерфейс - специфікаціям, ліцензіям і гарантіям якості. Компоненти програми Gridge, так само як і інші представники проміжного ПЗ Грід, пройшли успішне тестування різними версіями системи Globus. Компоненти програми Gridge розповсюджуються безкоштовно з повною комерційною підтримкою.

PSNC пропонує:

1. технічну підтримку, консультування, навчання і послуги розробників для програм Gridge і Globus.
2. сприяння в розробці, використанні і настройці проміжного ПО.
3. Установку і інтеграцію компонентів Gridge і Globus.
4. семінари і тренінги з технологій Грід.

Програмний продукт Gridge містить наступні інструменти і служби:

Служба авторизації Грід (GAS) – система авторизації, яка може служити єдиною точкою ухвалення рішень для всіх компонентів системи. Політика безпеки для всіх компонентів системи розміщена в GAS. Використовуючи умови даної політики GAS здатна анулювати рішення про авторизацію на прохання користувача. Служба GAS розроблена таким чином, що легко може інтегруватися із зовнішніми компонентами і здатна підтримувати безпеку комплексних систем. Здатність взаємодіяти з різними компонентами Globus і операційних систем, роблять GAS привабливим рішенням для Грід-додатків.

Система управління даними Грід - один з основних компонентів пакету управління даними Gridge (GDMSuite) – платформа проміжного ПО, яка надає уніфікований інтерфейс для з'єднання з різнорідними сховищами даних по всій мережі. GDMSuite є основою всього середовища Gridge, в рамках якого всі обчислювальні служби виконують всі свої операції. Пакет управління даними Gridge містить набір блоків, розроблених для створення повного і добротного середовища управління даними. Він розроблений так, щоб відповідати всім глобальним вимогам середовища Грід, таким як, безпека, сумісність і ефективність.

Мобільні служби Грід. Розробка програмного забезпечення для мобільних пристроїв повинна фокусуватися на додатках, які здатні встановити взаємодію між мобільними пристроями (мобільними телефонами, КПК, лэптопами) і службами Грід.

Журнальна система Toth. Цей компонент був розроблений для вирішення проблеми збору даних про події, які генеруються розподіленими службами середовища Gridge. Деякі сервіси при рішенні загальних завдань віддають перевагу системі на основі бібліотеки LOG4J, тому Toth повністю з нею сумісний.

Система моніторингу Грід. Система моніторингу Mercury розроблена в рамках проекту GridLab і надає собою головну інфраструктуру моніторингу Грід. Вона була спеціально розроблена для того, щоб відповідати специфічним вимогам моніторингу Грід: проведення моніторингу даних, представлених у вигляді метрик через моделі pull/push семантики доступу до даним, а також надавати можливість моніторингу управління.

Портали Грід розроблені спеціально для різних сценаріїв використання середовища співтовариствами кінцевих користувачів. Портали Gridge розроблені за допомогою наступних інструментів і додатків: GridSphere, сумісної оболонки Java і провайдера Грід, які дозволяють швидко розгортати і використовувати додатки, засновані на Грід, а також легко регулювати доступ до середовища декількох незалежних порталів.

Система управління ресурсами Грід. Цей компонент є системою мета - планування з відкритим початковим кодом, він дозволяє розробникам створювати і використовувати системи управління ресурсами для великих, важко керованих обчислювальних інфраструктур.

Програмне забезпечення LCG

Для побудови повністю функціональної Грід -системи необхідне програмне забезпечення проміжного рівня, що побудоване на базі існуючих інструментальних засобів і надає високорівневі сервіси завданням і користувачам. Прикладом такого програмного забезпечення може служити ПЗ LCG (LHC Computing Grid), що розроблявся в Європейському центрі ядерних досліджень (CERN). Спочатку метою проекту LCG була розробка повністю функціонуючої Грід -системи на базі Globus Toolkit для обробки даних у фізиці високих енергій. З часом область застосування LCG розширилася, і в даний час це – один з найпоширеніших пакетів ПО Грід.

Пакет LCG складається з декількох частин, званих елементами. Кожен елемент є самостійним набором програм (одні і ті ж програми можуть входити в декілька елементів), які реалізовують деякий сервіс і призначений для встановлення на комп'ютер під управлінням ОС Scientific Linux.

Нижче перераховані основні елементи LCG і їх призначення:

- **CE (Computing Element)** – набір програм, призначений для встановлення на вузол, який керує, обчислювального кластера. Даний елемент надає універсальний інтерфейс до системи керування ресурсами кластера і дозволяє запускати на кластері обчислювальні завдання;

- **SE (Storage Element)** – набір програм, призначений для встановлення на вузол зберігання даних. Даний елемент надає універсальний інтерфейс до системи зберігання даних і дозволяє керувати даними (файлами) в GRID-системі;
- **WN (Worker Node)** – набір програм, призначений для встановлення на кожен обчислювальний вузол кластера. Даний елемент надає стандартні функції і бібліотеки LCG завданням, що виконуються на даному обчислювальному вузлі;
- **UI (User Interface)** – набір програм, які реалізовують призначений для користувача інтерфейс Грід-системи (інтерфейс командного рядка). У цей елемент входять стандартні команди керування завданнями і даними, деякі з яких розглянуті в наступному розділі;
- **RB (Resource Broker)** – набір програм, які реалізовують систему встановлення завантаженням (брокер ресурсів). Це найбільш складний (і об'ємний) елемент LCG, що надає всі необхідні функції для скоординованого автоматичного керування завданнями в Грід -системі;
- **PX (Proxy)** – набір програм, що реалізовують сервіс автоматичного оновлення сертифікатів (myproxy);
- **LFC (Local File Catalog)** – набір програм, що реалізовують файловий каталог GRID-системи. Файловий каталог необхідний для зберігання інформації про копії (репліках) файлів, а також для пошуку ресурсів, які містять необхідні дані;
- **BDII (Infornation Index)** – набір програм, що реалізовують інформаційний індекс GRID-системи. Інформаційний індекс містить всю інформацію про поточний стан ресурсів, отримувану з інформаційних сервісів, і необхідний для пошуку ресурсів;
- **MON (Monitor)** – набір програм для моніторингу обчислювального кластера. Даний елемент збирає і зберігає в базі даних інформацію про стан і використання ресурсів кластера.
- **VOMS (VO Management Service)** – набір програм, які реалізовують каталог віртуальних організацій. Даний каталог необхідний для керування доступом користувачів до ресурсів Грід-системи на основі членства у віртуальних організаціях.

На один комп'ютер можлива установка відразу декількох елементів LCG, якщо це дозволяють його потужності (об'єм пам'яті і продуктивність). Мінімальна кількість вузлів, необхідних для розгортання повного набору ПЗ LCG, рівна трьом. Слід відмітити, що установка всіх сервісів на один вузол, хоча і можлива технічно, але настійно не рекомендується. Брокер ресурсів, з міркувань безпеки, слід розташувати на окремому вузлі. Обчислювальні вузли також слід виділити окремо, оскільки навантаження, що створюється на них працюючими завданнями, приведе до дефіциту ресурсів для решти сервісів. Решта всіх елементів може бути встановлені спільно.

У основі ПО LCG лежать розробки, виконані в рамках європейського проекту EDG (European DataGrid), які використані в більш функціональній

інфраструктурі програмного забезпечення, що носить назву gLite, зі збереженням сумісності.

Важливо відзначити, що все програмне забезпечення в рамках проекту LCG, може вільно використовуватися. На основі цього програмного забезпечення можливе створення національних і регіональних Грід -систем для ефективного розподілу локальних ресурсів. LCG став технологічною базою для інфраструктури в рамках проекту EGEE.

Middleware gLite

Навіть враховуючи появу в Globus Toolkit нових служб (CAS, RLS), слід визнати, що головний напрям його розвитку – засоби розробки і підтримки служб. Основна відмінність комплексу gLite в тому, що крім інструментальних засобів, в нього входить ширший набір служб. gLite істотно спирається на досвід ряду великих європейських проектів: EDG, LCG, Alien, NorduGrid і створювався колективно - в його розробці брали участь більше 80 фахівців з 11 дослідницьких центрів.

Приведемо огляд основних підсистем gLite.

Обчислювальний елемент (Computing Element - CE) - це служба, яка представляє ресурсний вузол GRID і що виконує на нім функції керування завданнями (запуск, видалення і так далі). Звернення до CE можуть надходити або від інтерфейсу користувача, або від Менеджера завантаження (Workload Manager -wm), який розподіляє завдання по безлічі CE. У glite функціональність CE розширена в порівнянні з аналогічною службою Lcg-2. Якщо в Lcg-2 CE може працювати тільки у відповідності до Push моделі (WM самостійно ухвалює рішення про посилання завдання на CE), то в glite можливий режим роботи CE також і в Pull моделі, коли CE запрошує завдання у WM.

Крім функцій управління завданнями, CE також виробляє інформацію про стан ресурсів. У Push моделі її публікує інформаційна служба, і вона використовується WM для вибору CE, на якому запускатиметься завдання. У Pull моделі інформація вбудовується в посилане WM повідомлення "CE доступний".

Підсистема управління даними (Data Management Subsystem - DM) включає три служби, що підтримують доступ до файлів: елемент пам'яті (Storage Element - SE), служби каталога (Catalog Services - CS) і диспетчер даних (Data Scheduling -ds). Всі служби працюють з даними на файловому рівні, в протилежність, наприклад, системам баз даних, які оперують такими елементами як записи і поля. У розподіленому середовищі Грід призначені для користувача файли можуть зберігатися в безлічі екземплярів - реплік, розміщених в різних місцях, і завдання CS і DS полягає в тому, щоб зробити процес управління репліками прозорим для користувача, так щоб додатки діставали доступ до файлів по їх іменах, або дескрипторах метаданих.

Доступ до даних файлів реально відбувається через SE, але DM підтримує також концепцію віртуальних наборів даних. Це відкриває нові цікаві можливості, засновані на абстракції глобальної файлової системи: при навігації по файлах клієнтське застосування може бути влаштоване як командна оболонка Unix, використовуючи команди зміни директорій, проглядання файлів

і тому подібне Захист файлів забезпечується в DM списками контролю доступу ACL (Access Control Lists).

Підсистема обліку (Accounting Subsystem - DGAS) акумулює інформацію про використання ресурсів Грід окремими користувачами, групами користувачів і віртуальними організаціями. Зібрана інформація дозволяє побудувати загальну картину діяльності в Грід, на основі якої може формуватися політика розподілу ресурсів і стягуватися плата за їх використання.

Підсистема протоколювання (Logging and Bookkeeping - LB) відстежує кроки обробки завдання, виконуються в різних точках Грід, фіксуючи події (запуск, розподіл на відповідний CE, початок виконання і так далі), що відбуваються і запам'ятовуючи їх. Інформація про події (протокол) поставляється компонентами WM і CE, для чого в ці компоненти вбудовуються звернення до LB. Протоколи збираються в два прийоми. Спочатку події передаються в локальну службу (Locallogger) і записуються у файл на диску. Locallogger відповідає за передачу протоколу одному з серверів зберігання (Bookkeeper), який "збільшує" події, даючи загальну картину зміни станів завдання (Submitted, Running, Done.). Крім того, Bookkeeper зберігає різні атрибути завдання: його опис (JDL); CE, на якому воно виконувалося; коди завершення і так далі. Як протокол станів, так і протокол подій можна отримати або за допомогою спеціального інтерфейсу WM, або через повідомлення при певних змінах стану, наприклад, при закінченні завдання.

Підсистема інформаційного обслуговування і моніторингу Грід (Relational Grid Monitoring Architecture - R-GMA) вирішує задачу збору і управління даними про стан Грід, отримуючи інформацію від безлічі розподілених джерел - постачальників. У її основі лежить розроблена однією з груп Global Grid Forum (GGF-PERF) схема "Споживач-Постачальник", яка описує спосіб взаємодії цих компонентів. Оскільки схема достатньо загальна, вона застосовна як для зберігання даних про Грід (які ресурси і служби доступні, які їх характеристики), так і для моніторингу додатків. R-GMA є реляційною реалізацією цього загального підходу. За наявності безлічі розподілених постачальників з погляду інформаційних запитів R-GMA діє як одна велика реляційна база даних. "Реляційність" виявляється у формі представлення даних: постачальники оголошують про склад публікованої інформації за допомогою конструкції SQL CREATE TABLE, публікують її за допомогою SQL INSERT, а споживачі отримують дані через SQL SELECT.

Підсистема управління завантаженням (Workload Management System - WMS) складається з ряду компонентів, відповідальних за розподіл завдань між ресурсами Грід, а також що забезпечують управління завданнями. Центральною компонентой є Менеджер завантаження (WM), який отримує від своїх клієнтів запити по управлінню завданнями. Зокрема, обробляючи запит типу "запуск" WM визначає відповідний для виконання CE, зважаючи на вимоги і переваги, задані в описі завдання.

Система безпеки. Вона розглядається як засіб захисту Web-служб і реалізована у вигляді додаткових модулів, що розміщуються в контейнерах (Apache Axis, Tomcat). Розроблені пропозиції по архітектурі безпеки:

https://edms.cern.ch/document/487004/ і сформульовані основні цілі: модульність, розширюваність, відповідність стандартам Web-служб, що розвиваються (Ws-security).

Інфраструктуру безопеки в дії можна продемонструвати наступним малюнко, де питання безпеки поставлено таким чином: “Створити процеси на взлах A і B, які потім обмінюються файлами з вузлом C.” (рис.2.6.4)

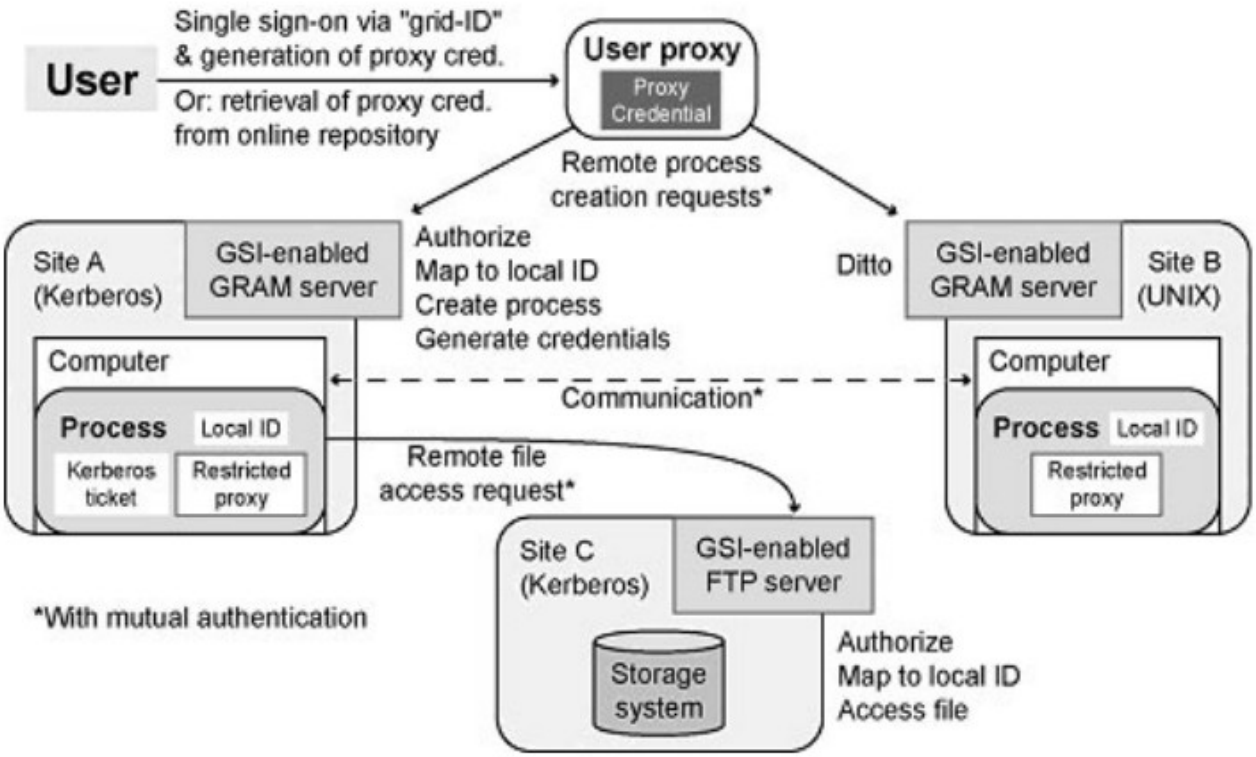


Рис.2.6.4 Інфраструктура безпеки

Остання версія gLite – 3.2 – була випущена в червні 2012 року, а з травня 2013 остаточно припинено підтримку проекту.

ППЗ ARC (Advanced Resource Connector)

Проект NorduGrid створений скандинавськими країнами (Данія, Норвегія, Швеція, Фінляндія) і перша його тестова експлуатація розпочалася в 2002р. Основні архітектурні рішення ARC відповідають загальноприйнятим вимогам побудови Грід. Це найпростіше ППЗ, яке підтримує всі необхідні функції для роботи Грід-інфраструктури, працює дуже стабільно і, нарешті, супроводжувалося (і супроводжується) найбільш повною і зрозумілою документацією.

В ARC реалізовані слідуючі функції:

– інформаційні (збір інформації про ресурси Грід-системи);
– динамічне підключення ресурсів в Грід -систему;
– розподілення завдань по ресурсам;
– пересилання завдань на виконання та керування цими завданнями;
– мониторинг Грід-системи.

Всі функції ARC реалізовані в вигляді окремих служб, кожна із яких основується на програмні засоби з відкритим кодом: OpenLDAP, OpenSSL,

SASL. ARC – це коннектор для проекту країн Північної Європи, які орієнтовані на підтримку додатків фізики високих енергій. Його реалізація виконана за допомогою бібліотек Globus Toolkit 2 (GT2), безпека досягається шляхом використання протоколів та інфраструктурних рішень GSI. Для опису запитів на виконання завдань в NorduGrid ARC використовується розширена мова запиту ресурсів XRSL (eXtended Resource Specification Language).

До переваг NorduGrid ARC можна віднести простоту в використанні (клієнтські компоненти пакету легко встановлюються, а серверні компоненти не вимагають реконфігурування сайту), пакет являється вільно розповсюджуваним, з відкритим вихідним кодом. Знакова особливість NorduGrid ARC – власнийй набір сервісів, які замінюють GT2. Однако багато функцій GT2 в ARC реалізуються власними засобами. Крім того, розширена мова опису ресурсів xRSL.

Слід зазначити, що українська Грід-інфраструктура використовує проміжне програмне забезпечення ARC.

UNICORE

Зараз ведеться розробка UNICORE 8.0, але остання актуальна версія UNICORE - 7.4.1, яка була випущена в 2018 році. UNICORE 7 має такі характеристики:

• клієнти:
  o веб-портал: виконання завдань, керування даними та підтримка базового робочого процесу, розширюваність;
  o графічний клієнт на основі Eclipse: Grid браузер, редактор робочих процесів, керування робочими процесами, розширюваність;
  o клієнт командного рядка: виконання робіт, передача даних, сценарії, пакетний режим, розширюваність;
• компоненти ядра сервера:
  o шлюз: маршрутизація веб-служб, обхід брандмауера;
  o UNICORE/X: Сервісний контейнер підтримує веб-служби SOAP, сервіси WSRF 1.2 і RESTful;
  o Grid сервіси: виконання завдань, доступ до сховища, метадані, передача файлів, реєстр послуг;
  o XUUDB: база даних користувачів, спільна між вузлами Grid, підтримує декілька входів і проектів;
  o гнучка система керування виконанням (XNJS): підтримка JSDL та HPC-BP, робота з розгортанням параметрів;
  o змінні механізми передачі файлів: HTTPS або ByteIO за замовчуванням, додатково UFTP, GridFTP;
  o інтерфейс цільової системи: підтримка загальних пакетних систем (LSF, LoadServer, SGE, Torque, SLURM);
• UNICORE Security:
  o на основі відкритих стандартів: X.509 Інфраструктура відкритого ключа, TLS, SAML, XACML;
  o гнучка аутентифікація користувачів: X.509, ім’я користувача/пароль, OpenID Connect;

- багато параметрів авторизації: XUUDB, Unity файли локальних карт;

• робочий процес:

- розширюваність: Custom resource brokering strategies;
- служба оркестратора: брокеринг ресурсів, виконання завдань, масштабованість за допомогою розгортання декількох екземплярів;
- потужні функції робочого процесу: графіки, петлі, умови, змінні, утримання/продовження;

• додатки:

- автономний пакет UFTP для високопродуктивнох передачи даних (незалежно від UNICORE);
- Unity: сервер керування ідентифікацією, сумісний з SAML, графічний інтерфейс адміністратора.

EUROPEAN MIDDLEWARE INITIATIVE

European Middleware Initiative (EMI) – результат кооперації чотирьох постачальників проміжного ПЗ для ґрід-обчислень – gLite, ARC, UNICORE та dCache. За функціональністю ППЗ розподілене на 4 основні групи сервісів: сервіси обчислення, сервіси даних, сервіси безпеки, інфраструктурні сервіси.
Сервіси обчислення включають служби ППЗ і клієнтські компоненти, які забезпечують обробку і керування запитами користувачів по виконанню обчислювальних завдань. Вони задовольняють взаємодію з локальними системами управління за допомогою єдиного інтерфейсу доступу до обчислювальних ресурсів і забезпечують доступність високорівневого метапланування, виконання потоку завдань та контроль стану завдання. Важливими є реалізація стандарту JSDL (Job Submission and Description Language, мова опису завдань в ґрід-системах) та специфікації OGF OGSA-Basic Execution Service (BES), яка планує відправлення і виконання завдань на обчислювальних ресурсах. OGSA-BES визначає операції, які мають бути реалізовані в ППЗ та необхідні для відправлення завдання на обчислення. До них входять операції з контролю стану завдання і завантаження результатів розрахунків.

Сервіси даних включають служби ППЗ і клієнтські компоненти, пов'язані з обробкою запитів користувачів з доступу до даних, зберіганням та реплікацією даних. Сервіси даних EMI повинні забезпечувати підтримку наступних стандартів:

- елементи зберігання даних мають забезпечувати POSIX доступ до даних через протокол NFS 4.1 (pNFS), дозволяючи інтегрувати розподілені системи передачі та обробки даних у простір імен локальної файлової системи;
- всі дані повинні бути доступними за допомогою веб-порталів HTTP(s) та WebDav;
- керування даними має бути забезпечено через протокол SRM;
- протоколи доступу мають бути захищені веб-стандартом аутентифікації X509/ SSL, який забезпечує взаємодію із широко розповсюдженими клієнтами (веб-браузерами).

Для забезпечення стандартизації інформаційної моделі виконується впровадження підтримки всіма сервісами GLUE2-специфікації, яка описує подання інформації про грід-інфраструктуру.

Для системи обліку використання ресурсів зараз доступні системи APEL та SGAS, які забезпечують зберігання інформації про використання ресурсів користувачами та забезпечують публікацію цієї інформації на веб-сайті.

Питання для самоконтролю

13 Що таке координація розрізнених ресурсів?
14 На що націлені Грід - обчислення ?
15 Програмне забезпечення Grid та його особливості.
16 Програмне забезпечення інтерфейсу користувача.
17 Призначення Middleware.
18 Перерахуйте найбільш відомі пакетіи проміжного програмного забезпечення Грід.
19 Дати визначення сервісно-орієнтованому гріду.
20 Охарактеризуйте Globus Toolkit.
21 Перерахуйте основні підсистеми gLite.
22 Що являє собою Підсистема управління завантаженням (Workload Management System - WMS)?
23 Охарактеризуйте GRAM (Grid Resource Allocation Management).

Література основна

1. Петренко А.І., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.

Література допоміжна

2. Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с.
3 www.globus.org
4 Support for Open Source Globus Toolkit Ends January 2018 | globus [Electronic resource] – Режим доступу: https://www.globus.org/blog/support-open-source-globus-toolkit-ends-january-2018 – Accessed: 04.05.2019 – Назва з екрану.
5 GT 5.0.0 Release Notes [Electronic resource] – Режим доступу: http://toolkit.globus.org/toolkit/docs/5.0/5.0.0/rn/#rn-changesummaries – Accessed: 04.05.2019 – Назва з екрану.
6 Globus Toolkit 6.0 Release Manuals [Electronic resource] – Режим доступу: http://toolkit.globus.org/toolkit/docs/6.0/ – Accessed: 04.05.2019 – Назва з екрану.
7 http://lcg.web.cern.ch/LCG/
8 www.glite.org
9 www.unicore.org
10 “LCG – Deployment”. Lcg.web.cern.ch. Retrieved July 29, 2010.
11 gLite - Lightweight Middleware for Grid Computing [Electronic resource] – Режим доступу: http://grid-deployment.web.cern.ch/grid-deployment/glite-web/ – Accessed: 11.05.2019 – Назва з екрану.

12 What's EMI? - European Middleware Initiative [Electronic resource] – 2016. – Режим доступу: https://web.archive.org/web/20161015155746/http://www.eu-emi.eu/what-is-emi – Accessed: 11.05.2019 – Назва з екрану.

2.7 **Проміжне програмне забезпечення Globus Toolkit, gLite, Unicore.**

В попередній лекції ми стисло познайомилися з ***midlleware Grid.*** В цій лекції більш детально розглянемо ППЗ Globus Toolkit, ARC, Unicore, gLite.

Globus Toolkit

Управління даними

Більшість існуючих систем забезпечують передачу великої кількості даних, типи даних різні, часто передаються в режимі реального часу. Елементами системи управління даними являються:

-GridFTP - стандартний FTP протокол, розширений можливостями паралельного пересилання даних і можливостями автоматичної повторної передачі в випадку виникнення помилки.

- Reliable File Transfer - надбудова над протоколом GridFTP, яка позволє працювати з виростанням web-сервісів.

- Replica Location Service- утиліта, яка позволяє зберігати інформацію про файли чи копії файлів в системі. Представляє собою реєстр, в якому міститься інформація про фізичне розташування файлів і їх репліцірованих копій. Реєстр може бути розділеним на декілька машин в мережі, що забезпечує стійкість до збоїв обладнання.

- Data Replication Service - сервіс більш високого рівня, ніж RFT і RLS. Він об'єднує в собі ці компоненти. Задачею сервісу являється перевірка на наявність деякого набору файлів на сервері. В випадку негативної відповіді він повинен відшукати файли шляхом запиту до сервісу RLS і ініціювати передавання файлів через RFT.

Керування виконанням завдань

Архітектура керування ресурсами GRMA (Globus Resource Management Architecture) має багаторівневу структуру (рис.2.7.1)

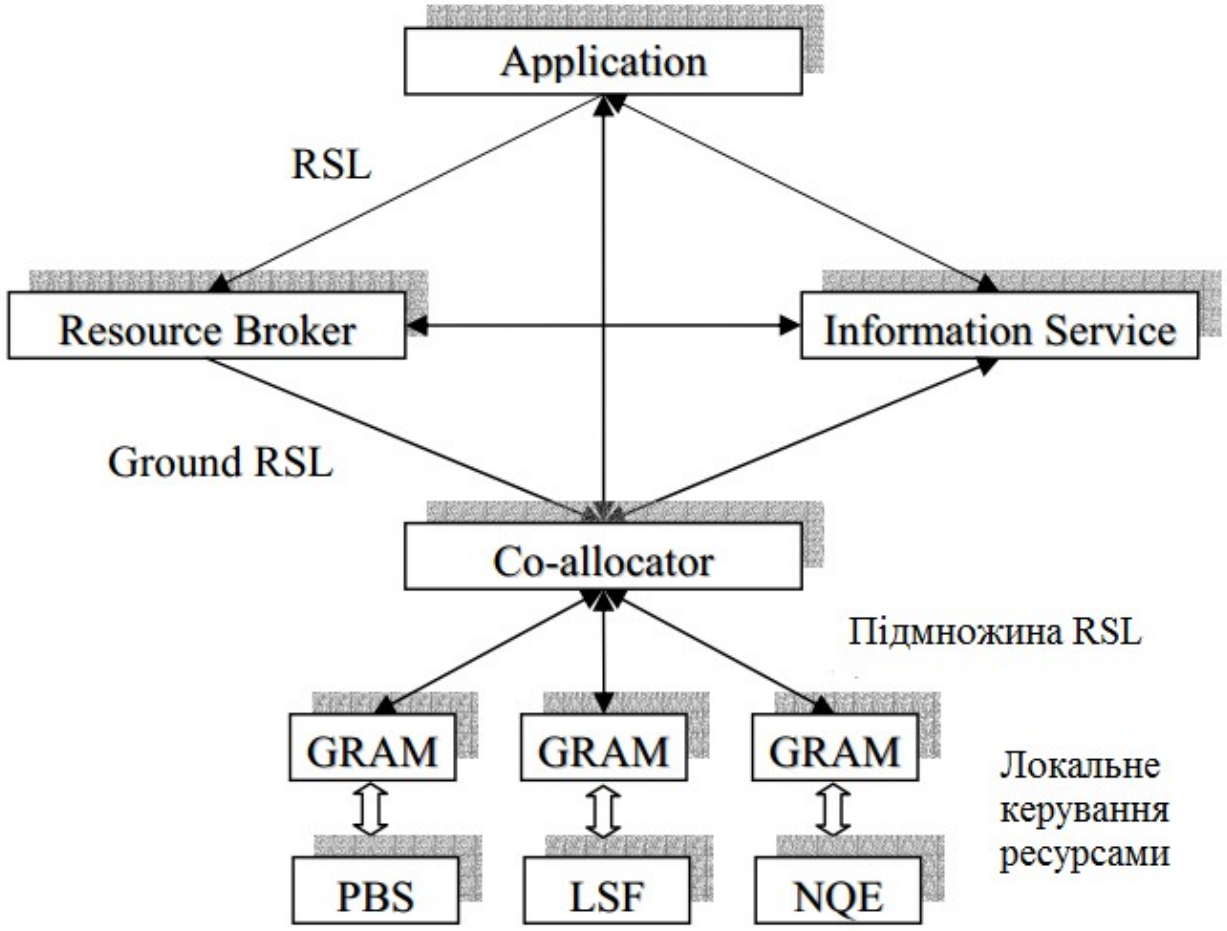


Рис.2.7.1 Структукра GRMA

Запити користувацьких додатків виражаються на RSL і передаються брокеру ресурсів, який відповідає за високорівневу координацію використання ресурсів (балансування навантаження) в певному домені. На основі переданого користувацьким додатком запиту та політики (права доступу, обмеження по використанню ресурсів) відповідального адміністративного домену брокер ресурсів приймає рішення про те, на яких обчислювальних вузлах буде виконуватися задача, який процент обчислювальної потужності вузла вона може використовувати та ін. При вибору обчислювального вузла брокер ресурсів має визначити, які вузли доступні в цей момент, їх загрузку, продуктивність та інші параметри, вказані в RSL-запиті, вибрати найбільш оптимальний варіант (один обчислювальний вузол, чи декілька), згенерувати новий RSL-запит (ground RSL) і передати його високорівневому менеджеру ресурсів (co-allocator). Цей запит буде містити вже більш конкретні дані, такі, як імена конкретних вузлів, необхідна кількість пам'яті та ін. Основні функції високорівневого менеджеру ресурсів GRMA перераховані нижче:

• колективне виділення ресурсів;
• збільшення/видалення ресурсів до раніше виділених;
• отримання інформації про стан задач;
• передача начальних параметрів задачам. GRMA виконує декомпозицію запитів ground RSL на множині простих RSL запитів і передає ці запити GRAM. Далі задача користувача запускається на виконання. Якщо один із GRAM повертає помилку, задача або знімається з виконання, або запускається

повторно. Високорівневий менеджер ресурсів виконує декомпозицію запитів ground RSL на множину більш простих RSL-запритів і передає ці запити GRAM. Далі, при відсутності повідомлень про помилки від GRAM, задача користівача запускається на виконання. Менеджер GRAM предоставляє верхнім рівням универсальний API для керування ресурсами вузла Грід. Сам GRAM взаємодіє з локальними засобами керування ресурсами вузла. Вузлом може бути, наприклад, робоча станція чи обчислювальний кластер.

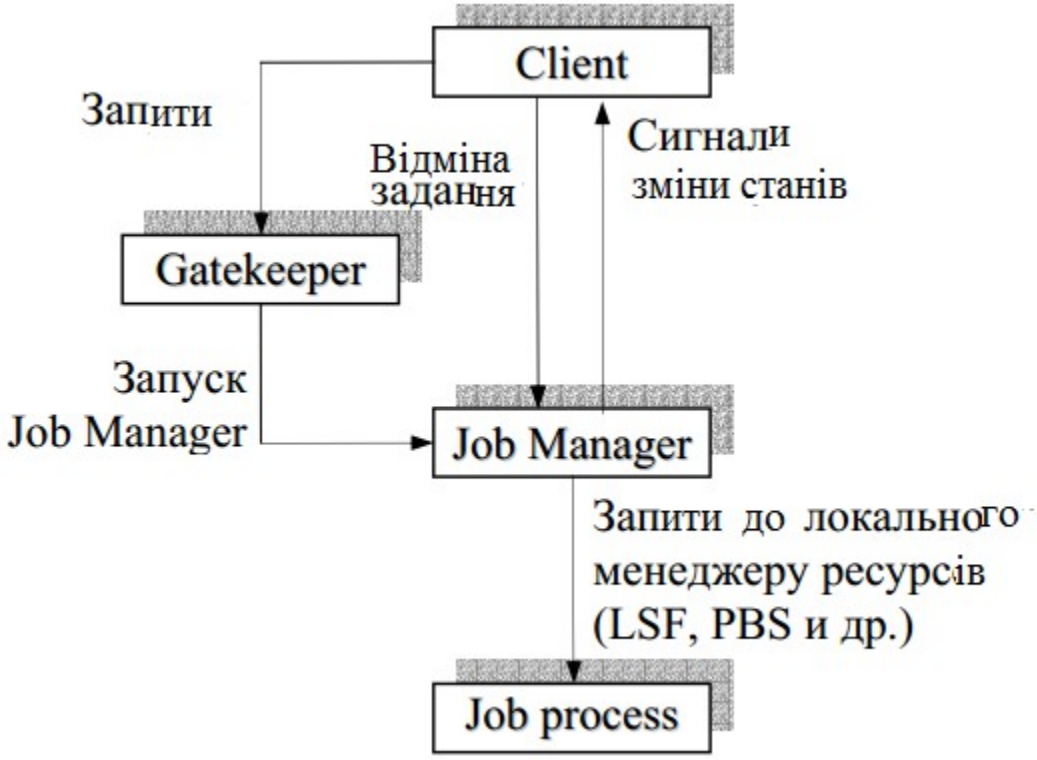


Рис 2.7.2 Схема взаємодії GRAM

GRAM – достатньо низькорівневий компонент Globus Toolkit, який являється інтерфейсом між високорівневим менеджером ресурсів і локальною системою керування ресурсами вузла. Цей інтерфейс може взаємодіяти з наступними локальними системами керування ресурсами:
• PBS (Portable Batch System) – система керування ресурсами і загрузкою кластерів. Може працювати на великій кількості різних платформ: Linux, FreeBSD, NetBSD, Digital Unix, Tru64, HP-UX, AIX, IRIX, Solaris. Існує вільна і володіюча більш широкими можливостями реализація PBS, називаєма Torque.
• Condor – вільно доступний менеджер ресурсів, розроблений в основному студентами різних університетів Европи і США Працює на різних платформах UNIX і Windows NT. Для того, щоб на даному обчислювальному вузлі можна було віддалено запускати на виконання програми, на ньому має виконуватися спеціальний процес, який називається Gatekeeper. Gatekeeper працює в привілейованому режимі і виконує наступні функції:
• проводить взаємну аутентифікацію з клієнтом;
• аналізує RSL запит;
• відображає клієнтський запит на облікову запись деякого локального користувача;

• запускає від імені локального користувача спеціальний процес , називаємий Job Manager, і передає йому список потрібних ресурсів. Після того, як Gatekeeper виконає свою роботу, Job Manager запускає завдання (процес чи декілька процесів) і проводить його подальший мониторинг, повідомляючи клієнту про помилки та інші події. Gatekeeper запускає тільки один Job Manager для кожного користувача, котрий керує всіма завданнями даного користувача. Коли завдань більше не залишається, Job Manager завершує роботу.

Організація GRAM. Коротко опишемо реалізацію GRAM (рис 2.7.3).

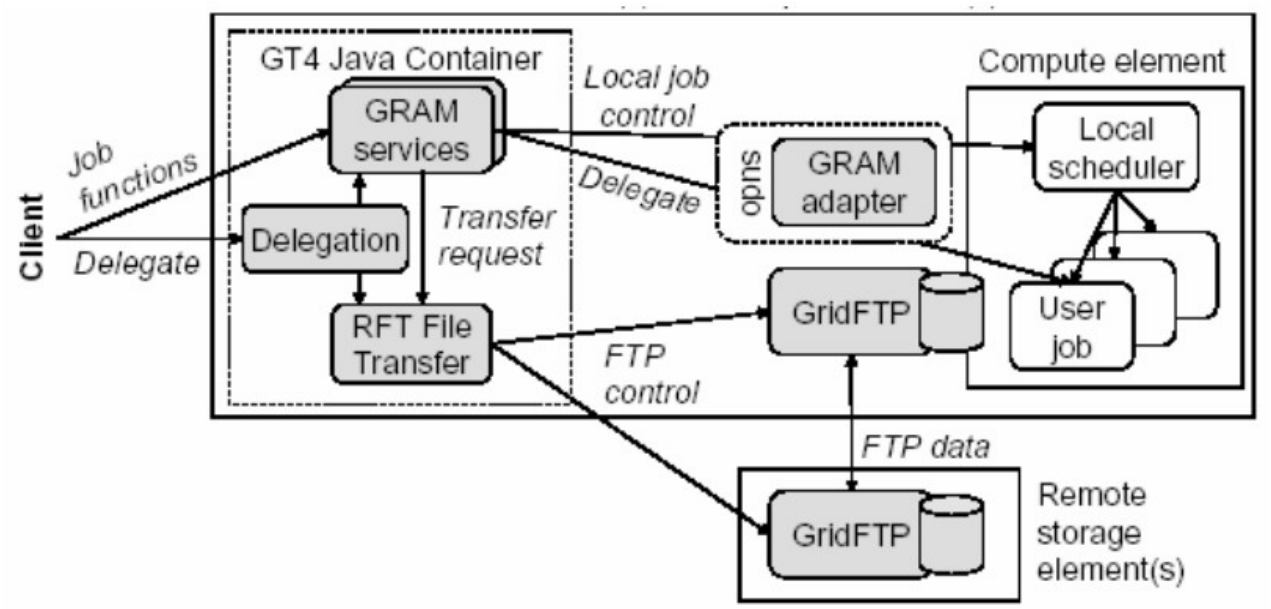


Рис 2.7.3 Компоненти GRAM

Основними елементами GRAM є:

• GRAM специфічні служби для створення, спостереження і керування роботами.

• Загальноцільові служби для передачі прав і надійної передачі даних RFT.

• Плануючі GRAM адаптери, предназначені для передачі запитів в відповідні локальні планувальники.

• GridFTP сервери для виконання конвейерних команд.

• Програми командного рядка (Таб.1). Інтерфейси користувача позволяють визвати компоненти безпосередньо із програми користувача, написаної на Java, C, Python.

Таб.1 - Кожний рядок представляє ряд програм

Команди	Опис
cas-*	Загальна служба авторизації
globus-credential-*	Інсталяція і відновлення прав в сужбі передачі прав
myproxy-*	Служба MyProxy
grid-ca-sign	Простий видавець для генерації X.509 прав
gsi*	OpenSSH с ssh, scp, stpf клієнтами
grid-*	Управління X.509 проксі мандатами і файлами розміщення
globus-url-copy	GridFTP клієнт
rft*	Клієнт надійної передачі файлів
globus-rls-*	Клієнт RLS, адміністрація, сервер
globusrun-ws	Клиент GRAM
mds-servicegroup-add	Добавлення входу в MDS групу служб
globus-*-container	Запуск і зупинка контейнера Globus
wsrf-*	Взаємодія з WSRF властивостями ресурсів
wsn-*	Створення і керування WS-Notification subscriptions.

В таб.2 вказані види статусної інформації, яку користувач на свій розсуд в любий момент може запросити, працюючи, наприклад, в пакетному режимі.

Таб. 2 - Стани робіт в GRAM

С	Опис
Unsubmitted	Робота ще не завдана
StageLn	Робота очікує етапу виконання чи вхідних файлів для закінчення
Pending	Локальний планувальник ще не запланував роботу на виконання
Active	Робота виконується
Suspended	Виконання роботи призупинено
StageOut	Виконання роботи завершено, вихідні файли виведені
CleanUp	Виконання роботи завершено, виконується очистка задач
Done	Робота завершена успішно
Failed	Робота потерпіла неудачу

Система Condor

Еволюція створення ППЗ привела до того, що алгоритми планування розміщення завдання на розподілених ресурсах були виділені в окрему службу, яка забезпечувала додаткові можливості по вибору ресурсів, моніторингу проходження завдань і доступу до даних з додатків. Прикладами таких програмних реалізіцій є Nimrod/G, AppLe Parameter Sweep Template, Condor-G, Gridbus Resource Broker, GridWay.

Condor – одна із багатьох спеціалізована пакетна система для керування виконанням інтенсивних комп'ютерних робіт, яка може використовуватися під GRAM. Як і інші системи подібного роду, Condor забезпечує механізм черговості, політику планування, схему пріоритетів і класифікацію ресурсів. Користувач задає комп'ютерну роботу для Condor d RSL, Condor ставить роботу в чергу, запускає її і потім інформує користувача про результати. Порядок виконання роботи такий:

Підготовка коду. Робота для Condor має бути придатною для виконання в фоновому пакетному режимі. Програма, яка виконується в фоновому режимі, не придатна для інтерактивного вводу-виводу. Condor може перенаправити консольний вивід і ввід з клавіатури в/із файла. Для цього треба створити потрібний файл, котрий містить необхідні рядки для програмного входу.
Середовище Condor. Condor має декілька средовищ для виконання (Universe): Standard, Vanilla, PVM, MPI, Globus, Java, Scheduler. Користувач має вибрати одне із них.
Задання файлу опису. Цей файл містить такі дані, як ім'я програми, яка буде виконуватися, файли для використання даних екрану і клавіатури, тип платформи, необхідної для виконуємої програми і багато іншої інформації.
Задання роботи. Робота для Condor задається командою *condor_submit*. Після цього Condor самостійно виконує дії по запуску і виконанню завдання.

Інформаційний сервіс

Основним елементом системи інформаційного сервісу являється підсистема оновлення і відшукування сервісів MDS (Monitoring and Discovery Service), в яких зареєстровані всі ресурси Грід. Інформація про ресурси може містити, наприклад, як дані про конфігурацію чи стану як всієї системи, так і окремих її ресурсів (тип ресурсу, доступний дисковий простір, кількість процесорів, об'єм пам'яті, продуктивність та інше). Вся інформація логічно організована в вигляді дерева і доступ до неї проводиться по стандартному протоколу LDAP (Lightweight Directory Access Protocol). Кожна запись в структурі LDAP містить властивості окремого ресурсу. Користувацький додаток може запросити сервер для встановлення типу архітектури, підтримуваємої операційною системою, даних про менеджер задач, які доступні на заданому ресурсі. Кожний організаційний домен, який входить в Грід, має запустити сервіс інформації про ресурси (GRIS). Функціональність даного сервісу включає в себе:

• Огляд і отримання інформації, яка може бути точно співставлена з конкретним ресурсом чи об'єктом.
• Можливість робити запити і пошук інформації для отримання набору зв'язаних ресурсів чи об'єктів.
• Створення нової інформації по запиту, яка не збережена в системі.
• Переадресація подій для передачі інформації, основаної на динамічних подіях в середині інфраструктури Grid.
• Збір інформації для тематичного отримання і організації інформації.
• Фільтрація інформації для зменшення її кількості і збільшення продуктивності.
• Зберігання, резервне копіювання, кешування інформації для більш простого наступного доступу од неї і для підвищення продуктивності.
• Безпека, захист і шифрування для забезпечення роботи важливих задач, таких як контроль доступу, аутентифікація.
• Для підтримки даної функціональності користувачами Грід-системи можуть бути використані кілька сервісів. MDS має децентралізовану,

легкомасштабовану структуру і працює як зі статичними, так і з динамічними даними, необхідними користувацьким додаткам і різним сервісам Грід-системи. Ієрархічна структура MDS представлена на рис.2.7.4.

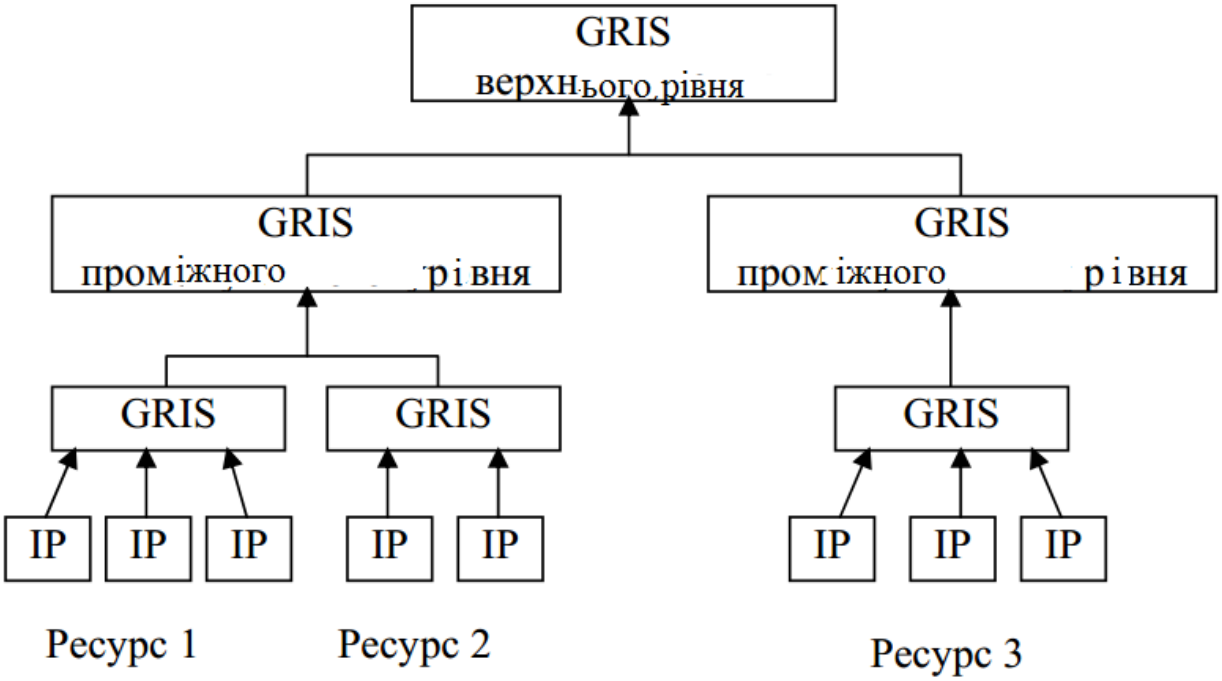


Рис.2.7.4 Ієрархічна структура MDS

MDS складається із трьох основних компонент:

1. IP (Information Provider) – являється джерелом інформації про конкретний ресурс.

2. GRIS (Grid Resource Information Service) – предоставляє інформацію про вузол Грід-системи, який може бути як обчислювальним вузлом, так і будь-яким іншим ресурсом. GRIS опитує індивідуальні IP і об'єднує отриману від них інформацію в рамках єдиної інформаційної схеми.

Пакет gLite

gLite - європейський пакет middleware для Грід.

Нижче перераховані основні елементи і їх призначення:

CE (Computing Element) - набір програм для установки на керуючий вузол обчислювального кластеру. Даний елемент надає універсальний інтерфейс до системи управління ресурсами кластеру і дозволяє запускати на кластері обчислювальні завдання;

SE (Storage Element) - набір програм, призначений для установки на вузол зберігання даних. Даний елемент надає універсальний інтерфейс до системи зберігання даних і дозволяє керувати даними (файлами) в Грід-системі;

WN (Worker Node) - набір програм, призначений для установки на кожний обчислювальний вузол кластеру. Даний елемент надає стандартні функції і бібліотеки LCG завданням, які виконуються на даному обчислювальному вузлі; (LCG – це Large Hadron Collider Computing Grid, предназнач. для великого адронного колайдера)

UI (User Interface) - набір програм, що реалізують інтерфейс користувача Грід-системи (інтерфейс командного рядка). В цей елемент входять стандартні команди керування завданнями і даними, деякі з яких розглянуті далі;

RB (Resource Broker) - набір програм, які реалізують систему керування завантаженням (брокер ресурсів). Це найбільш складний (і об'ємний) елемент LCG, що надає всі необхідні функції для скоординованого автоматичного керування завданнями в Грід- системі;

PX (Proxy) - набір програм, які реалізують сервіс автоматичного оновлення сертіфікатів (myproxy);

LFC (Local File Catalog) - набір програм, які реалізують файловий каталог Грід-стеми. Файловий каталог необхідний для зберігання інформації про копії (репліки) файлів, а також для пошуку ресурсів, які містять необхідні дані;

BDII (Infornation Index) - набір програм, які реалізують інформаційний індекс Грід-системи. Інформаційний індекс містить всю інформацію про поточний стан ресурсів, отримувану з інформаційних сервісів, і необхідну для пошуку ресурсів;

MON (Monitor) - набір програм для моніторингу обчислювального кластеру. Даний елемент збирає і зберігає в базі даних інформацію про стан і використання ресурсів кластеру.

VOMS (VO Management Service) - набір програм, які реалізують каталог віртуальних організацій. Даний каталог необхідний для управління доступом користувачів до ресурсів Грід-системи на основі членства в віртуальних організаціях.

До базових систем gLite відносяться такі системи:

• **Система управління завданнями WMS** (Workload System). Завданням підсистеми управління загрузкою є прийняття запитів на запуск завдань, пошук відповідних ресурсів і контроль їх виконання. Завдяки роботі WMS, складність управління додатками і ресурсами в Гріді прихована від користувачів. Їх взаємодія з WMS обмежена описом характеристик і вимог запиту через орієнтовану на користувача мову опису завдань (Job Description Language, JDL) і до направлення такого запиту через надані інтерфейси.

• **Система керування даними**. Прикладне завдання повинно бути в змозі звернутися до своїх даних, незалежно від фактичного місця розташування обчислювального ресурсу. Найбільш поширеним таким інтерфейсом є Менеджер ресурсів зберігання даних (Storage Resource Manager, SRM).

• **Система інформаційного обслуговування і моніторингу** Грід-системи вирішує завдання збору і управління даними про стан Грід, отримуючи інформацію від безлічі розподілених джерел - постачальників. Підсистема призначена для постійного контролю функціонування Грід і забезпечення своєчасного реагування на виникаючі проблеми.

• **Система безпеки і контролю прав доступу**. Щоб Грід був ефективною структурою для розподілених обчислень, призначені процеси і служби Грід для користувача повинні працювати в безпечному середовищі. Для цього взаємодії між компонентами Грід повинні бути взаємно аутентифіковані (аутентифікація - процес підтвердження заявлених властивостей об'єкту на підставі засвідчень його документів); будь-яка дія повинна відбуватися тільки

після відповідної авторизації - зіставлення об'єкту, який здійснює дію, і набору прав, наданих цьому об'єкту для роботи в Грід-середовищі.

• **Система протоколювання**. Підсистема протоколювання відстежує процес виконання завдань, здійснюваний під управлінням WMS. Вона збирає сповіщення про події від різних компонентів WMS і обробляє їх, щоб представити узагальнений поточний стан (статус) завдання.

• **Система обліку**. Ця підсистема призначена для обліку використання обчислювальних ресурсів (таких як процесорний час, використання оперативної пам'яті і так далі), і, зокрема, може використовуватися для формування інформації про вартість використання даного Грід-ресурсу даним користувачем, якщо взаємини користувачів і провайдерів ресурсів засновані на економічній моделі.

Управління виконанням завдань

Основна і найбільш розвинена частина в складі комплексу gLite - це система управління завданнями (Workload Management System, WMS). Її призначення - підтримка виконання програм на розподілених і організованих у вигляді гріда комп'ютерах. Структура системи керування завданнями приведена на рис2.7.5.

Для користувача Грід обчислювальна діяльність виглядає таким чином: він запускає завдання, яке доставляється на один, або в разі багатопроцесорних завдань - на кілька комп'ютерів із загального ресурсного пулу Грід; завдання виконується, а результати можуть бути отримані на робоче місце, з якого здійснювався запуск.

Рис2.7.5 Структура системи керування завданнями

Реалізовані в WMS технології підтримують розподілену обробку такого роду, забезпечуючи:

- автоматичне виділення відповідних виконавчих комп'ютерів (це основна відмінність WMS від базових засобів керування завданнями, в яких потрібна явна вказівка виконавчих ресурсів);
- переміщення програми, вхідних і діагностичних файлів;
- створення середовища виконання, в тім рахунку домашньої директорії, на виконавчому комп'ютері.

Форми користувацької діяльності в Грід і на окремому комп'ютері відрізняються в декількох аспектах:

• Програмний код завдання виконується на виконавчому комп'ютері без участі користувача (в пакетному режимі). Сам код не вимагає адаптації до умов Грід. В деяких випадках може знадобитися його доповнення прологом / епілогом, які виконують підготовчі / завершальні операції, наприклад, доставку оброблюваних даних.

• Завдання представляється WMS у вигляді формалізованого опису, складеного на мові JDL (Job Description Language).

• Ресурси Грід використовуються колективно безліччю користувачів, тому Завдання не обов'яково починає виконуватися відразу після запуску: воно може чекати звільнення ресурсів, зайнятих іншими завданнями. Очікуючі ресурсів завдання зберігаються в чергах (WMS або ресурсних центрів CE).

• Питання безпеки в Грід gLite вирішуються на основі принципів GSI. Перед тим як почати працювати з WMS (або будь-якою іншою системою gLite), користувач повинен згенерувати тимчасовий проксі-сертифікат за допомогою команд grid-proxy-init або voms-proxy-init.

Компоненти системи керування виконанням завдань

Після відправки за допомогою інтерфейсу користувача запит обробляється декількома компонентами WMS. Їх точний склад може варіюватися в залежностісті від конкретного ППЗ, але зазвичай в нього входять:

• менеджер завантаження (можливо з додатковим проксі-сервером);
• планувальник / брокер ресурсів;
• адаптер завдань;
• модуль, відповідальний за виконання фактичних операцій з керування завданнями (напрямок на виконання, видалення завдання, і т.д.);
• монітор log-файлів WMS.

Менеджер завантаження - основний компонент WMS. Після отримання запиту, він повинен вжити відповідних заходів, щоб виконати його. Щоб зробити це, він взаємодіє з іншими компонентами WMS, набір яких залежить від типу запиту. Для підвищення продуктивності системи (кількості запитів, які можуть бути оброблені за одиницю часу), цей компонент доповнюється проксі-менеджером завантаження, відповідальним за прийом поступаючих запитів від інтерфейсу користувача (наприклад, на запуск або видалення завдання), аналіз

їх коректності та, якщо результат аналізу позитивний, передачу запиту менеджеру завантаження.

Основним типом запитів до WMS являються запити на виконання завдань. А головними результатами їх обробки підсистемою являються:

• правильний підбір грід-ресурсу для виконання конкретного завдання на підставі опису завдання на мові JDL та інформації про доступні ресурси.

• направлення завдання на обраний ресурс.

Першу задачу вирішує компонент, який називається планувальником або брокером ресурсів (Resource Broker, RB). Існування відповідних ресурсів для даного завдання залежить не тільки від стану ресурсів, але також і від політики їх використання, якою слідують адміністратори ресурсу і / або адміністратор віртуальної організації, до якої належить користувач.

Брокер ресурсів може слідувати різній стратегії при розподілі завдань по ресурсам:

• один крайній варіант стратегії - після надходження запиту на виконання завдання, як можна швидше підбираються найбільш підходящі ресурси та, як тільки рішення прийнято, завдання передається на обраний ресурс для виконання («нетерпляче планування», в англомовній літературі називається також push-model);

• інший крайній варіант - завдання знаходяться в менеджері завантаження, поки будь-який ресурс не стає доступним, після чого вивільненому ресурсу підбирається найбільш відповідне завдання (що чекали) і передається цьому ресурсу для виконання («лінива політика планування», pull-model);

• можливі проміжні комбіновані варіанти стратегії підбору ресурсів.

Для виконання своїх завдань брокер використовує базу даних про ресурси, яка оновлюється внаслідок або отримання повідомлень від ресурсів, або активного опитування ресурсів системою інформаційного обслуговування і моніторингу, або деякої комбінації цих двох механізмів оновлення. Крім того, спеціальний компонент WMS, який відповідає за взаємодію з інформаційною підсистемою, може бути налаштований так, щоб певні повідомлення про зміну станів ресурсів могли викликати початок роботи брокера. Іншим важливим компонентом, що підвищує стійкість роботи WMS, є модуль керування чергою завдань, який дає можливість деякий час зберігати запит на виконання завдання, навіть якщо ніякі ресурси, відповідні його вимогам, не доступні негайно. Запити, по яким не вдалося відразу підібрати ресурси, будуть повторені або пеперіодично (при «нетерплячому плануванні»), або як тільки повідомлення про нові доступні ресурси з'являються в сховищі даних (при «лінивій політиці планування»). В протилежному випадку, такі ситуації могли б привести до негайного припинення виконання завдання через відсутність відповідного ресурсу.

Далі **слідує модуль, відповідальний за виконання фактичних операцій** з керування завданнями (направлення на виконання, видалення завдання, і т.д.), які виконуються за запитами менеджера завантаження. У gLite в якості цього модуля використовується Condor C.

Монітор log-файлів відповідальний за перегляд повідомлень модуля керування завданнями і перехоплення подій про зміну станів завдання.

Опис завдання

Користувач взаємодіє з WMS зі свого робочого місця, на якому встановлено інтерфейс (User Interface, UI) цієї системи, що включає наступний набір команд:
glite-job-submit <jdl_file> запуск завдання, опис якого міститься в файлі <jdl_file
glite-job-cancel <job_Id> зняття завдання в будь-який момент процесу обробки.
glite-job-status <job_Id> отримання стану завдання.
glite-job-output <job_Id> отримання сумарної діагностики від всіх х етапів обробки.

Як видно з цього списку, набір команд досить повний: він дозволяє запустити завдання, відслідковувати хід обробки і при необхідності зняти його. Повертаючись до особливостей роботи в Грід, звернемо увагу на те, що обробка команд проводиться розподіленим чином різними службами WLMS, які взаємодіють між собою по мережі. У зв'язку з цим час виконання команд досить великий, близько хвилини.

Файл опису завдання - JDL-файл задається як параметр команди job-submit і містить характеристики завдання у вигляді пар: <атрибут> = <значення>. (наприклад)

```
[Type = "Job";

JobType = "Normal";

Executable = "myexe";

StdInput = "myinput.txt";

StdOutput = "message.txt";

StdError = "error.txt";

InputSandbox = {"/users/pacini/example/myinput.txt",
"/users/pacini/example/myexe"};

OutputSandbox = {"message.txt", "error.txt"};

Requirements = other.GlueCEInfoLRMSType == "PBS";
Rank = other.FreeCPUs;]
```

Серед атрибутів можна виділити три основні групи, які визначають: тип завдання, використовувані в завданні файли і спосіб вибору ресурсів.

Тип завдання описується двома атрибутами: Type і JobType. WMS підтримує роботу як з простими завданнями, так і з складовими, атрибут Type має, відповідно, значення "Job" або

"DAG". Тип DAG (direct acyclic graph of dependent jobs) відповідає завданням, в якому повинні бути виконані в певній послідовності ряд простих завдань.
Для простого завдання застосуємо другий атрибут - JobType. Воно може бути звичайним ("Normal"), а також:
• інтерактивним ("Interactive"). Завдання такого роду підтримує зв'язок з точкою запуску
• паралельним (MPICH), тобто вимагає для виконання декількох процесорів і ін.

Прості завдання

WMS забезпечує переміщення файлів між комп'ютером робочого місця і виконавчим комп'ютером, але обмежується, проте, мінімально необхідним набором стандартних файлів операційної системи (Unix): виконуваним файлом завдання, вхідних даних, діагностичних повідомлень і файлом повідомлень про помилки.

Вхідні файли (виконуємий і файл даних), які лежать в файловій системі робочого комп'ютера користувача UI, передаються спочатку на сервер WMS, а потім при запуску завдання на виконавчому комп'ютері завантажуються в створену для цього завдання домашню директорію.

Вихідні файли діагностики і помилок породжуються при виконанні завдання в домашній директорії виконавчого комп'ютера, і доставляються по його закінченні не в точку запуску, а на сервер WMS, звідки їх можна отримати командою glite-job-output. Аналогічно вхідним файлам, вихідні повинні бути описані в атрибутах OutputSandbox, StdOutput і StdError.

Оскільки переміщення файлів відбувається через тимчасовий буфер на сервері WMS, передбачається, що всі вони будуть невеликого розміру. Тим самим, ці засоби доставки файлів спрямовані на мінімально необхідну підтримку віддаленого виконання програм.

Що стосується прикладних даних, які обробляються або виробляються додатком, то для роботи з ними у програмі повинні використовуватися засоби системи **керування даними**.

Одне з найважливіших досягнень WMS - технологія віртуалізації, яка реалізує автоматичний розподіл завдань на виконавчих ресурсах. Розподіл проводиться з точністю до ресурсного центру, тобто визначається не конкретний виконавчий комп'ютер, а деякий CE. При цьому в певній мірі враховується стан і склад його ресурсів (число завдань в черзі, платформа виконавчих комп'ютерів і т.д.). Ці відомості представляються диспетчеру завдань WMS інформаційною службою gLite.

Схема обробки завдань

Наступний рис.2.7.6 ілюструє процес виконання в Грід деякого простого завдання.

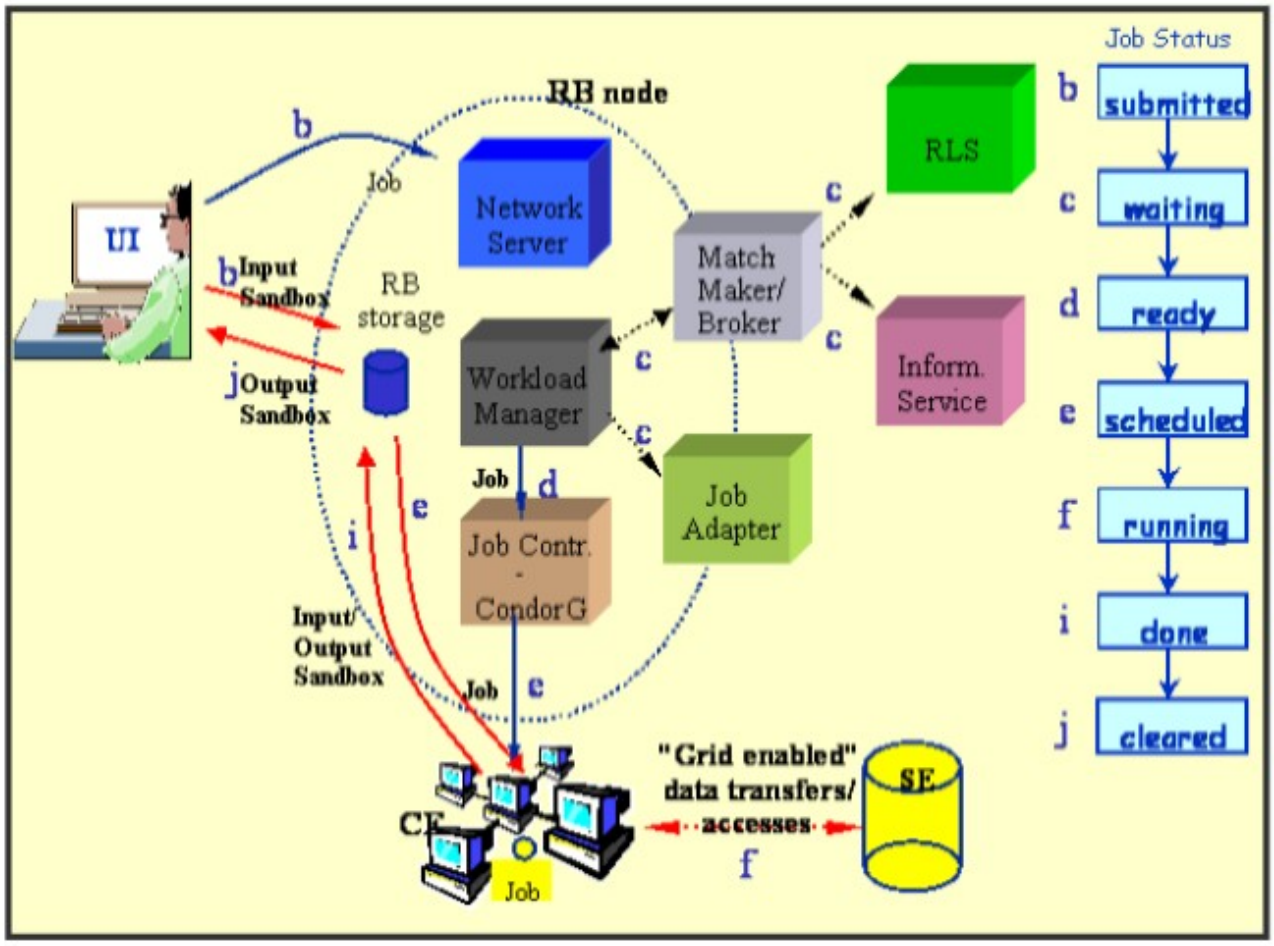


Рис.2.7.6Виконання завдання в gLite. Справа представлені стани виконання завдання

1. Після отримання цифрового сертифікату від Certification Authority, реєстрації в ВО (Віртуальній Організації), користувач може використовувати gLite. Він реєструється в UI і створює проксі-сертифікат, щоб представляти себе в операціях з безпеки.

2. Користувач запитує роботу через UI до Resource Broker. В описі роботи вказані один або більше файлів, які потрібно скопіювати в RB. Цей ряд файлів називається Input Sandbox. Подія реєструється в LB і роботі присвоюється статус **запрошено**.

3. WMS шукає найкращий доступний CE для виконання роботи. Щоб зробити це, він запитує **Information Supermarket (ISM**, внутрішній кеш інформації, який в цій системі читає з BDII, щоб визначити статус обчислювальних ресурсів і ресурсів пам'яти і в File Catalogue, щоб знайти місце будь-якого необхідного вхідного файлу. Інша подія реєструється в LB і встановлюється статус роботи ОЧІКУВАННЯ.

4. RB готує роботу для запиту виконання, створюючи оболонку скрипту, який буде пересланий разом з іншими параметрами в обраний CE. Подія реєструється в LB і роботі присвоюється статус ГОТОВИЙ.

5. CE отримує запит і посилає роботу на виконання в локальний LRMS.

6. Подія реєструється в LB і роботі присвоюється статус СПЛНОВАНО.

7. LRMS керує виконанням роботи на Worker Nodes. Файли вхідного Sandbox копіюються з RB в доступний WN, де робота виконується. Подія реєструється в LB і роботі присвоюється статус ВИКОНАННЯ.

8. Під час виконання роботи файли Грід можуть бути доступні з SE, використовуючи або RFIO, або протокол перекриття, або після копіювання їх в локальну файлову систему на WN за допомогою Data Management.

9. Робота може створювати нові вихідні файли, які можуть бути перезавантаженими в Грід і стати доступними для інших користувачів. Це може бути досягнуто за допомогою Data Management. Перезавантаження файлу означає копіювання його в Storage Element і реєстрацію в каталозі файлів.

10. Подія реєструється в LB і роботі присвоюється статус ЗРОБЛЕНО.

11. З цієї точки користувач може отримувати вихідний файл його роботи в UI, подія реєструється в LB і роботі присвоюється статус **ОЧИЩЕНО**.

12. Запити на статуси роботи можуть бути адресовані LB з UI. Отже, з UI можливо робити черги BDII для статусів ресурсів.

13. Якщо сайт, до якого надіслана робота, нездатний прийняти, або виконати роботу, робота може бути автоматично і багаторазово переадресовано іншим відповідним CE. Користувач може отримати інформацію про історію роботи, запитуючи службу LB.

Система **UNICORE**

Проект UNICORE (Uniform Interface to Computing Resources - єдиний інтерфейс до обчислювальних ресурсів) зародився в 1997 році і до цього моменту являє собою комплексне рішення, орієнтоване на забезпечення прозорого безпечногого доступу до ресурсів Грід.

Архітектура складається з чотирьох рівнів:

1. Клієнтській рівень.
2. Шлюз.
3. Серверний рівень.
4. Системний рівень.

Клієнтська частина складеться з:

- UCC (Unicore Command Line Client) – клієнт командного рядка для UNICORE. Забезпечує постановку задачі і вивід отриманих результатів. Підтримує Json для опису задач, вміє працювати в фоновому режимі і працювати зі скриптами на Groove.
- URC (Unicore Rich Client) – багатофункціональний клієнт UNICORE. Заснований на Eclipce. Забезпечує перегляд ресурсів Грід-сітки. Дозволяє детально описати задачі і описати потоки задач. Дозволяє керувати безпекою і моніторити виконання потоку задач.
- HiLA (High Level API for Grid Applications) - високорівневий програмний інтерфейс для додатків грід. Є підтримка Java.
- Портали – надає доступ користувачам до Грід- ресурсів через інтернет.

Шлюз слугує компонентом, який забезпечує доступ до вузла UNICORE, через автентифікацію всіх вхідних повідомлень, використовуючи стандарт X.509.

Серверний рівень є проміжним шаром, який включає в себе всі сервіси і компоненти UNICORE, заснованих на стандартах WSRF і SOAP. Включає в себе реєстр, який забезпечує реєстрацію і пошук ресурсів, доступних в Грід-середовищі. Компонент XNJS (eXecution Network Job Supervisor) забезпечує керування завданнями і виконання ядра UNICORE. Системний рівень являє собою інтерфейс цільової системи (TSI – Target System Interface), який забезпечує взаємодію між UNICORE і окремими ресурсами Грід-сітки.

Слід зазначити, що проект UNICORE був ініційований для німецьких суперкомп'ютерних центрів в якості альтернативи для Globus Toolkit. Основи виробничої версії були закладені в продовжені проекту Unicore Plus, який закінчився в 2002 р. Наступні європейські проекти розширили функціональні можливості і були направлені на забезпечення реалізації стандартів Open Grid Forum. Це привело до выпуску UNICORE 6 28 серпня 2007 р. Зараз ведеться розробка UNICORE 8.0, але остання актуальна версія UNICORE - 7.4.1, яка була випущена в 2018 році. Архітектура поточної версії UNICORE складається з трьох рівнів: клієнтського, серверного та рівня цільової системи.

Клієнтський рівень представлений наступними додатками, які входять до складу UNICORE:

- Portal, що дозволяє користувачам отримати доступ до обчислювальних ресурсів та даних з використанням браузера без необхідності встановлювати додаткове програмне забезпечення на користувацький комп'ютер;
- URC (UNICORE Rich Client) – графічний клієнт, побудований на Eclipse Framework, для персональних комп'ютерів;
- UCC (UNICORE command line client) – клієнт для командного рядка з можливістю розширення набору команд скриптами мовою Groovy.

Також можливе створення користувацьких прикладних програм безпосередньо з використанням клієнтських інтерфейсів прикладного програмування мовою Java.

Серверний рівень представлений сервісами та компонентами, побудованими на стандартах WSRF 1.2, SOAP та WS-I:

- Gateway (шлюз) – зворотній HTTPS проксі-сервер, який виконує роль мережевого екрана та точки входу до порталу UNICORE, переадресовуючи запити підключених клієнтів на сервери за ним;
- UNICORE/X – центральний компонент UNICORE, він приймає запити клієнтів, які пройшли через шлюз, виконує їх автентифікацію, перевірку авторизації та викликає відповідний сервіс. Він надає набір інтерфейсів сервісів, що розміщуються в контейнері UNICORE Services Environment (USE) та виконують управління завданнями, передачу файлів, надають доступ до сховища, тощо. Сервер UNICORE/X приймає завдання на виконання, передає їх на локальні системи через TSI, дозволяє завантаження та вивантаження даних з використанням кількох протоколів. UNICORE/X може для зберігання даних використовувати такі ресурси, як сховище CDMI, S3 чи Apache Hadoop HDFS;
- Registry (реєстр) – сервер аналогічний до UNICORE/X, проте виконується як окремий сервіс в оточенні WSRF та дозволяє клієнтам виявляти доступні сервіси, розміщені на декількох серверах UNICORE/X;

- Workflow Engine та Service Orchestrator – компоненти, що відповідають за управління виконанням завдань в мережі Grid.

Рівень цільової системи (TSI) складається з інтерфейсів локальної операційної системи, файлової системи та системи управління ресурсами. Даний рівень використовується для виконання завдань, операцій з файловим введенням/виведенням та перевірки стану завдання.

Питання для самоконтролю

1. Що таке координація розрізнених ресурсів?
2. На що націлені Грід- обчислення ?
3. Охарактеризуйте програмне забезпечення Грід.
4. Перерахуйте особливості програмування GridGRID.
5. Що таке Управління виконання завдань?
6. Призначення Middleware.
7. Назвіть найбільш відомі пакетіи ППЗ Грід.
8. Охарактеризуйте Архітектуру керування ресурсами GRMA.
9. Охарактеризуйте Globus Toolkit.
10. Охарактеризуйте gLite.
11. Охарактеризуйте Unicore.
12. Що являє собою Підсистема управління завантаженням (Workload Management System - WMS)?
13. Охарактеризуйте GRAM (Grid Resource Allocation Management).

Література

1. www.globus.org
2. www.globus.org
3. Streit A. UNICORE – What lies beneath Grid functionality? // eStrategies. Vol.
4. 2008. P. 38-39.
5. Streit A. UNICORE: Getting to the heart of Grid technologies // eStrategies.Vol. 3. 2009. P. 8-9.
6. Nordugrid ARC (http://www.nordugrid.org)
7. Официальный сайт Unicore [Электронный ресурс]: — Режим доступа: https://www.unicore.eu/
8. Architecture | UNICORE [Electronic resource] – Режим доступу: https://www.unicore.eu/documentation/architecture/ – Accessed: 05.05.2019 – Назва з екрану.

2.8 Сумісніять проміжного програмного забезпечення Грід. European Middleware Initiative (EMI)

(Значна частина цієї лекції запозичена із довідника [1])

З попередніх лекцій Ви маєте знати, що Грід забезпечує вирішення складних завдань шляхом інтеграції розподілених ресурсів, ідентифікації користувачів, автоматичного виконання їх запитів і пошуку для них потрібних ресурсів. Базові функції Грід реалізує за допомогою спеціального програмного забезпечення, яке має назву ***проміжного програмного забезпечення (ППЗ),*** або

midlleware. ППЗ використовується для того, щоб приховати різнорідну природу Грід та створити прозорий доступ до прикладного програмного забезпечення. ППЗ дозволяє створити однорідне середовище шляхом використання стандартизованих інтерфейсів і цілого ряду сервісів. ППЗ перебуває між операційною системою і додатками.

Як і на початку появи перших персональних комп'ютерів кожна фірма окремо створювала власні ПК і про їх стандартизацію ніякої мови не було, так і створення різних Грід (як національних так і регіональних) супроводжувалося розробкою власного проміжного програмного забезпечення : ARC, Alien, LCG, Unicore, gLite, Globus Toolkit та ін. (прогляньте попередні лекції) Всі вони розрізняються між собою, але мають багато спільного, оскільки використовують, як правило, систему Globus Toolkit. Архітектура і програмна реалізація такого ППЗ різна і дослідницькі проекти обмежені можливостями однієї Грід-інфраструктури через відмінності в застосованому ППЗ. Але відмінність в ППЗ не єдина проблема. Використання віртуальних організацій (ВО) за галузевим напрямком (фізикам потрібно одне, а біологам інше) стало логічним продовженням розвитку Грід-комп'ютингу, проте ВО виявилися організаційно обмеженими, так як структури, які входять до їх складу, мають свої специфічні механізми доступу до ресурсів і процедури використання Грід, які визначаються різним ППЗ. Побудова і функціонування ВО як об'єднання ресурсів різних Грід, стала ще однією складністтю на шляху забезпечення функціональної сумісності.[1]

Забезпечення взаємодії Грід, які працюють під управлінням різного ППЗ, є актуальною задачею.

Функціональна сумісність (Interoperaility) визначається як здатність обмінюватися інформацією і здатність використовувати ту інформацію, якою обмінювалися.

Взаємодія (Interoperation) визначається як використання систем, які мають можливість взаємодіяти.

Грід-функціональна сумісність – це здатність ППЗ працювати спільно, а Грід-взаємодія – це здатність Грід-інфраструктур працювати спільно. Функціональна сумісність – це перший крок в напрямку до взаємодії Грід-систем. Правда, це з точки зору користувача Грід-системи, бо він повинен мати можливість виконати складне завдання, яке вимагає великих обчислювальних ресурсів, відправивши його на виконання в потрібний Грід, і отримати результат.

Моделі забезпечення функціональної сумісності можна поділити на три групи:

- Вирішення проблеми функціональної сумісності без зміни ППЗ. До цієї групи відносяться сценарії користувача і паралельне використання Грід.
- Моделі проміжної стадії. До цієї групи відносяться розробка адаптерів, трансляторів, шлюзів.
- Моделі, які забезпечують універсальне вирішення проблеми функціональної сумісності: розробка універсального інтерфейсу для доступу до Грід-систем, розробка єдиного стандарту Грід.

Сценарій користувача

В межах віртуальної організації, побудованої на основі різних Грід з різним ППЗ, можна досягти функціональної сумісності шляхом їх окремого використання. Для цього необхідно на персональному комп'ютері встановити і конфігурувати грід-орієнтований користувацький інтерфейс для кожної із цих систем, зареєструватися в ВО, які забезпечують доступ до різних Грід-інфраструктур і таким чином можна працювати з цими Грід. Але користувач сам повинен розподілити обчислення між Грід-інфраструктурами, відправляючи завдання на виконання в конкретну Грід залежно від поточного її завантаження. А ще потрібно мати різний опис одного і того ж прикладного завдання для кожної з Грід, які використовує ВО. Для віртуальної організації необхідна реєстрація в різних Грід, які вона повинна використовувати. При цьому недостатньо використовуються можливості Грід і проблематично локалізувати збої при виконанні завдань.

Паралельне використання

В межах організації, яка має потужний обчислювальний центр, отримати функціональну сумісність можна встановленням інтерфейсу доступу до різних Грід на окремому виділеному сервері, який входить до складу обчислювального кластеру. При цьому обчислювальний ресурс буде частиною різних Грід-інфраструктур і доступ до кожної з них буде шляхом використання встановлених Грід-сервісів. Для такої роботи користувач має бути зареєстрованим в кількох Грід, відправляючи завдання на виконання в ту чи іншу систему залежно від її завантаження. Тут так само потрібно мати різний опис прикладного завдання для кожної Грід, яку використовує ВО. Відмінність від попередньої моделі в тому, що доступ у різних Грід здійснюється через один ресурс, який має бути зареєстрованим в різних Грід. Основний недолік – адміністратор системи має стати експертом в питаннях інсталяції, конфігурування та підтримки працездатності сервісів різного ППЗ.

Шлюз

Шлюз (Gateway) - це специфічний сервіс, який забезпечує єдиний доступ до ресурсів грід-інфраструктур, які працюють під керуванням різного ППЗ, і робить різні грід схожими на єдиний обчислювальний ресурс. Для забезпечення функціональної сумісності ППЗ розроблюється окремий модуль – шлюз або Gateway, що забезпечує переклад завдання користувача, написаного на мові опису завдання для однієї Грід у формат мови опису завдання для іншої Грід-інфраструктури.

З точки зору користувача є один опис завдання для Грід, а переклад його на мову опису завдання, придатну для використання в іншій Грід-інфраструктурі виконується Gateway-модулем. Для такої системи характерно:

- кожна грід працює незалежно під керуванням власного ППЗ;
- функції по узгодженню форматів мови опису завдання, авторизації і аутентифікації користувача, передачі файлів завдання і даних на обчислювальні ресурси і відправлення результатів обчислення користувачеві реалізуються в окремому модулі;
- модуль Gateway є єдиною точкою обміну пакетами між Грід і користувачем.

Використання модулів Gateway є вузьким місцем з погляду універсальності, але такий підхід корисний як доказ концепції і демонстрації реального досягнення функціональної сумісномті. Цей метод був використаний в EGEE (*Enablig Grid for E-sciensE)* для забезпечення функціональної сумісності грід-інфраструктур UNICODE і OSG. (Перша промислова грід була побудована в 2004р в проекті EGEE).

Адаптери і транслятори

Цей метод забезпечення функціональної сумісності заснований на використанні адаптерів (adaptpor) і трансляторів, які є частиною інтнрфейсу користувача Грід.

Еволюція створення ППЗ привела до того, що алгоритми планування розміщення завдання на розподілених ресурсах були виділені в окрему службу, яка забезпечувала додаткові можливості по вибору ресурсів, моніторингу проходження завдань і доступу до даних з додатків. Такий проміжний шар названо Ресурс Брокер. Прикладами таких програмних реалізіцій є Nimrod/G, AppLe Parameter Sweep Template, Condor-G, Gridbus Resource Broker, GridWay. До них можна віднести і планувальники завдань, такі як Application Level Schedultrs.

Фактично користувач має один вхід в різні Грід-системи, однаково готує завдання на виконання і визначає необхідні для його виконання ресурси. Використання адаптерів дозволяє підключати декілька Грід, а трансляторів – змінювати інформацію опису завдань так, щоб вона була зрозуміла відповідній Грід-системі, що забезпечує роботу з різними Грід-інфраструктурами як з єдиним обчислювальним ресурсом. Така модель вимагає модифікації ППЗ, але існуючі інтерфейси можуть бути використані без істотних змін.

Розробка загального стандартного інтерфейсу для різних Грід-інфраструктур може розглядатися як універсальне рішення, яке зніме проблему функціональної сумісності, але так як кожна Грід вже сильно пов’язана зі своїм власним ППЗ, перехід на інший інтерфейс значно ускладнений. Розробка стандарту інтерфейсу для використання рядом Грід, виконується в рамках проекту European Middleware Initiative (EMI).[2]

Роботи із забезпечення функціональної сумісності і взаємодії Грід, які велися до 2010р практично у всіх країнах-учасниках Грід-проектів, не привели до створення єдиної інфраструктури, яка б дозволила використовувати обчислювальні ресурси різних національних Грід-проектів. Саме тому в 2010 р в рамках EGI був утворений окремий підпроект European Middleware Initiative (EMI) по створенню єдиного проміжного програмного програмного забезпечення Unified Middleware Distribution (UMD).

Для уніфікації ППЗ чотири консорціуми gLite, ARC, UNICORE і dCache об’єднали зусилля в спільному проекті розвитку проміжного програмного забезпечення EMI (офіційний сайт проекту http://www.eu-emi.eu/).

Зараз в Європі використовуються в промисловій експлуатації три види ППЗ: ARC, gLite, UNICORE. Вони використовуються в трьох головних грід Європи: EGI, DEISA і NDGF. Кожне із цих ППЗ виконує функціональні можливості по

наданню грід-сервісів в рамках своїх проектів, але жодне з них не покриває всі потреби в Грід-обчисленнях.

Використаний термін Unified Middleware Distribution (UMD) для позначення інтегрованого дистрибутива із ППЗ gLite, ARC, UNICORE і dCache.

Передбачено випустити три основні версії дистрибутиву UMD, які забезпечать функціональну сумісність основних типів ППЗ.

EMI 1 (Kebnekaise) :
ставилася задача забезпечити початковий етап інтеграції основних компонентів і сервісів в усіх чотирьох (gLite, ARC, UNICORE і dCache) типах ППЗ і дотримання політики формування версії основних дистрибутивів операційної системи. Функціональність компонентів ППЗ на цьому етапі не змінилася, але були зроблені зміни в структурі компонентів для більшої інтеграції з операційними системами і розроблені додаткові модулі забезпечення однорівневої взаємодії ППЗ. Це розширило підтримку інших операційних систем (крім Scientific Linux) у ППЗ.

EMI 2 (Matterhorne): завдання розширення і покращення функціональних можливостей ППЗ, збільшення кількості операційних систем, які підтримують надійну роботу ППЗ, розширення функціональності ППЗ, підтримка загальних інтерфейсів усіх чотирьох ППЗ і краща відповідність прийнятим стандартам.

EMI 3 (Monte Bianco): у третьому дистрибутиві - завершення стандартизації складових ППЗ.

Все ППЗ дистрибутива EMI за функціональністтю розподілене на чотири основні групи сервісів: Сервіси Обчислення (Compute Services), Сервіси Даних (Data Services), Сервіис Безпеки (Security Services), Інфраструктурні Сервіси (Infrastructure Services).

Сервіси обчислення: включають служби ППЗ і клієнтські компоненти, які забезпечують обробку і керування запитами користувачів по виконанню обчислювальних завдань. Вони задовольняють взаємодію з локальними системами управління за допомогою єдиного інтерфейсу доступу до обчислювальних ресурсів і забезпечують доступність високорівневого метапланування, виконання потоку завдань та контроль стану завдання.

Специфікація EMI Execution Service (ES) розроблена на основі аналізу використання обчислювальних сервісів у промислових грід-інфраструктурах. Ця специфікація прийнята в якості основної.

Іншим важливим стандартом є *Job Submission and Description Language (JSDL) –стандарт мови опису завдання в Грід-системах.* Адаптація цього стандарту до різних типів ППЗ, які входять в дистрибутив EMI, є найважливішим завданням. Тісно зв'язана зі стандартом JSDL специфікація OGF OGSA-Basic Execution Service (BES), яка планує відправлення і виконання завдань на обчислювальних ресурсах. OGSA- BES визначає операції, які мають бути реалізовані в ППЗ і які необхідні для відправлення завдання на обчислення. Сюди входять операції з контролю стану завдання і завантаження результатів розрахунків.

Реалізація специфікації OGSA-Basic Execution Service дозволяє забезпечити функціональну сумісність сервісів обчислення EMI.

Сервіси Даних: включають служби ППЗ і клієнтські компоненти, зв'язані з обробкою і керуванням запитами користувачів з доступу до даних, керуванню зберіганням і реплікацією даних. Сервіси даних EMI поєднують основні компоненти і досвід використання трьох головних європейських ППЗ (gLite, ARC, UNICORE) і dCache Storage Technology в роботі з великим обсягом даних. Сервіси даних EMI мають підтримувати наступні стандарти:

- елементи зберігання даних мають забезпечувати POSIX доступ до даних через протокол NFS 4.1 (pNFS), дозволяючи інтегрувати розподілені системи передачі і обробки даних у простір імен локальної файлової системи.
- всі дані мають бути доступні за допомогою веб-порталів HTTP (s) і WebDav , які широко застосовуються.
- керування даними має бути забезпечено через протокол SRM, який успішно використовується в LHC Грід-обчисленнях.
- протоколи доступу мають бути захищені веб-стандартом аутентифікації X509/SSL, який забезпчує взаємодію із широко розповсюдженими клієнтами (веб-браузерами).

Сервіси гарантування безпеки: включають служби проміжного програмного забезпечення і компоненти, які задовольняють безпеку при спільному використанні Грід-ресурсів. Вони відповідають за керування ідентифікаційними даними, членством в Віртуальних організаціях, аутентифікацію, авторизацію, делегування прав доступу і відновлення облікових даних.

Розвиток Грід зумовив популярність спрощеної авторизації – так званої Single Sign-On (SSO), що означає, що доступ до ресурсів виконується при одноразовому введенні логіну і паролю при першій авторизації, бо користувачі, які мають аккаунт в одній Грід, будуть авторизовані в іншій Грід-системі з використанням SSO.

Одним із завдань програмного забезпечення, заснованого на веб-сервісах (Security Token Service [STS]), є надання можливості виконати аутентифікацію з використанням централізованого сервісу гарантування безпеки. Сервіс STS дозволяє виконувати перетворення сертифікатів користувача з одного формату в інший, наприклад, з формату SAML (Security Assertion Markup Language), на якому грунтується SSO-механізм, у формат сертифікатів X.509.

Нступним важливим завданням є впровадження єдиного механізму опису атрибутів ролей користувачів на основі Security Assertion Markup Language – стандарту обміну інформацією з авторизації і атентифікації користувачів в розподіленій обчислювальній системі. SAML – це відкритий стандарт, який був розроблений для захищеного обміну інформацією в розподіленій системі з використанням XML-нотацій. Цей протокол використовується в ППЗ UNICORE як сервіс авторизації UVOS (як VOMS в gLite) і використовується для авторизації користувача на обчислювальному ресурсі з використанням WS-

Security. Цей механізм на основі SAML – стандарту інтегрується в усі компоненти дистрибутива EMI.

Extensible Access Control Markup Language (XACML) є аналогом SALM і забезпечує строгий опис політики гарантування безпеки та використовується під час прийняття рішень про авторизацію. Extensible Access Control Markup Language розроблений організацією OASIS і актуальний для EMI. Особливо це стосується визначення загальних атрибутів для політики авторизації в різних типах ППЗ, які входять в дистрибутив EMI. Extensible Access Control Markup Language є XML-стандартом опису політики доступу до розподілених обчислювальних ресурсів з використанням веб-сервісів.

Для цих цілей використовується загальна бібліотека модулів аутентифікації (CANL – Common Authenticatio Library), що дозволяє виконати прехід і підтримку всіх сервісів EMI, оптимізувати їх структуру і забезпечити повну функціональну сумісність. Ця бібліотека має підтримувати API-інтерфейс для мов C/C++ і Java. Ще один важливий напрямок – адаптація і впровадження загальної централізованої системи авторизації ARGUS для всіх сервісів ППЗ і використання загального XACML-профілю авторизації.

Структура централізованої системи авторизації ARGUS приведена на рис.2.8.1.

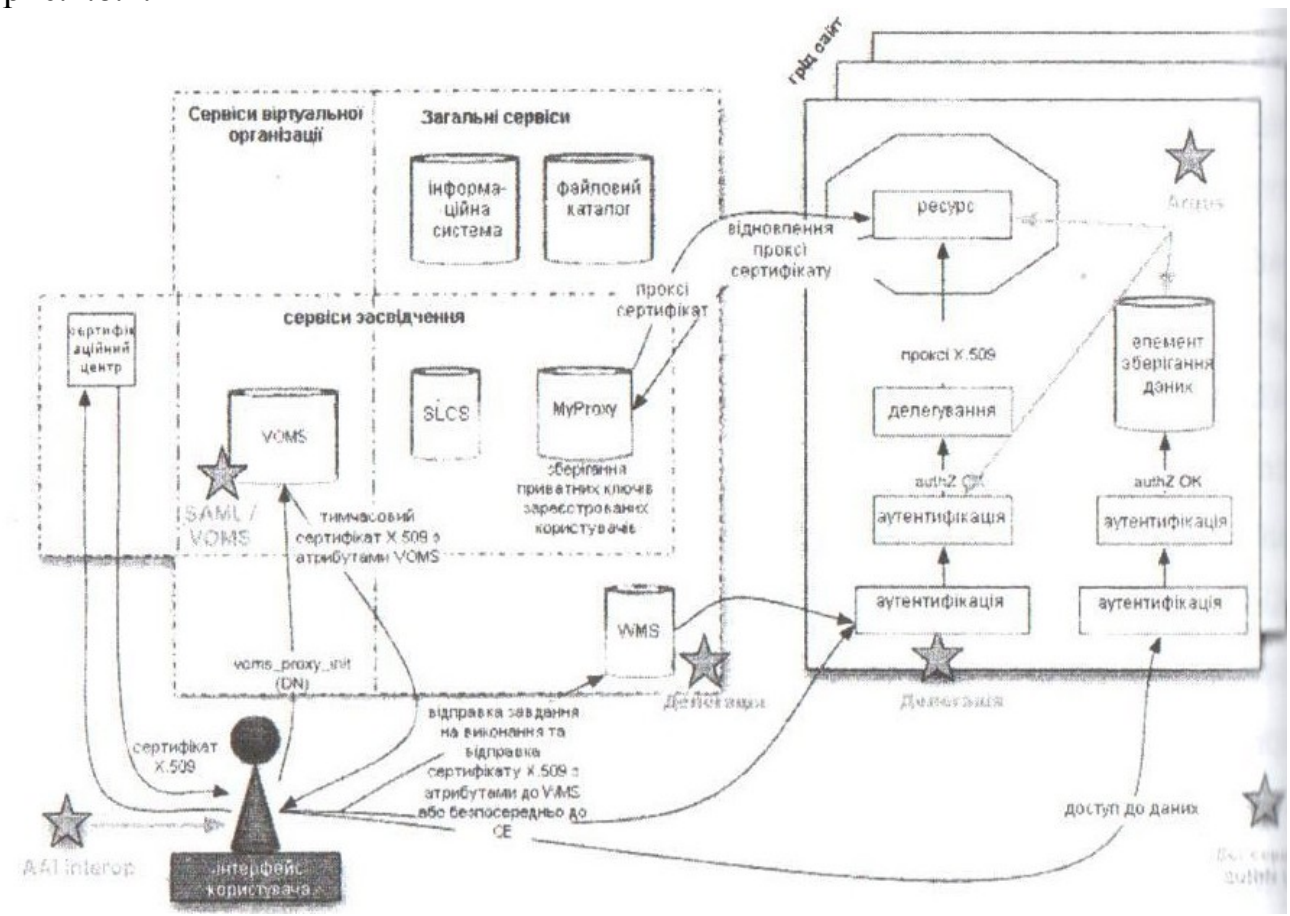


Рис.2.8.1 Структура централізованої системи авторизації ARGUS

Подальше гарантування безпеки – це реалізація загальгого методу делегування прав доступу на основі сертифікату X.509, тобто здійснення

загального методу обробки інформації проксі-сертифікату всіма компонентами дистрибутиву UDM.

Інфраструктурні сервіси: включають служби проміжного програмного забезпечення і компоненти, які надають загальну інформацію функціональності грід-сервісів.

Для забезпечення стандартизації інформаційної моделі потрібна адаптація і впровадження GLUE2- специфікації для всіх компонентів дистрибутива UMD. (що таке Glue? Коли були зроблені перші кроки до функціональної сумісності і взаємодії між великими Грід EGEE і OSG (*це було в 2004р*), була побудована матриця функціональної сумісності, яка показала подібні і відмінні ознаки між ППЗ в цих проектах. Основною відмінністтю було вказане розходження у використанні інформаційних систем і для забезпечення функціональної сумісності була розроблена інформаційна схема Glue. Через кілька місяців OSG змінив власну інформаційну систему на Glue-схему. В січні 2005р був інстальований OSG-сайт із новою схемою Glue. Більшість Грід почали використовувати Glue-схему.)

Основним ключовим стандартом є OGF-специфікація GLUE2, яка описує подання інформації про грід-інфраструктуру. Схема GLUE2 має підтримуватися всіма сервісами UMD.

Для системи обліку використання ресурсів зараз доступні система APEL і система SGAS. Вони забезпечують зберігання інформації про використання ресурсів користувачами і забезпечують публікацію цієї інформації на веб-сайті.

Іншим ключовим стандартом є OGF Usage Record Format (UR), який описує використання Грід-сервісів. Він дозволяє реалізувати міжінфраструктурний облік і позволяє закласти основу для повноцінного обліку при переході від академічного до комерційного використання Грід.

Контроль доступності сервісів грід виконується за допомогою періодичного виконання тестових завдань (SAN, MonALISA, Nagios).

Доступний для установки дистрибутив релізу EMI містить біля сотні програмних продуктів постачальників ППЗ ARC, gLite, UNICORE. Впровадження вже першої версії EMI позволило забезпечити функціональну сумісність основних сервісів ППЗ в рамках дослідницьких проектів WLCG/EGI і PRACE (рис.2.8.2).

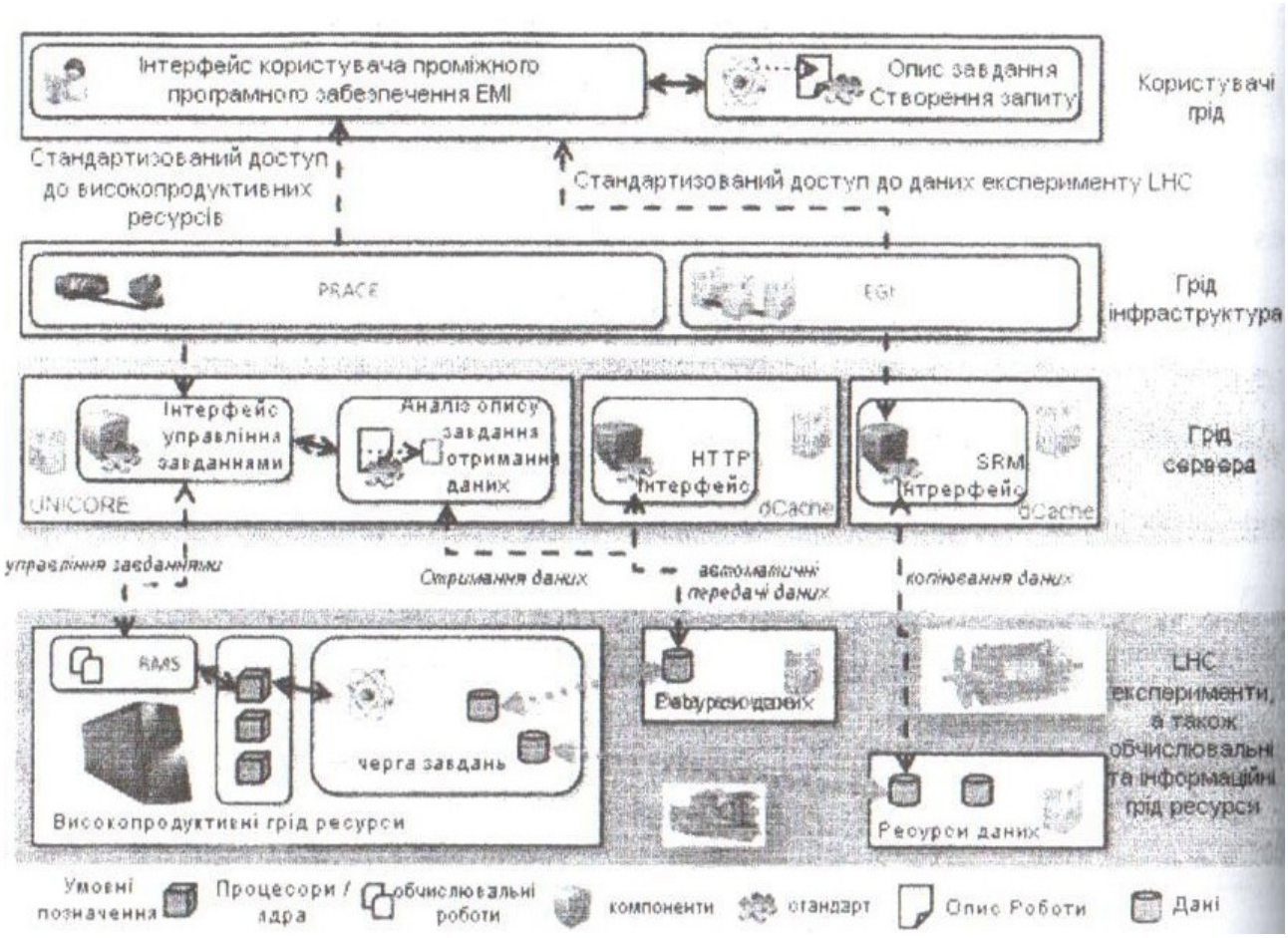


Рис.2.8.2 Приклад інтеграції між проектами WLCG/EGI і PRACE

Використання проміжного програмного забезпечення EMI

Для використання Грід-сервісів на комп'ютерах користувачів інсталюється спеціальне програмне забезпечення - інтерфейс користувача (***User Interface – UI),*** яке забезпечує взаємодію з Грід-середовищем. Користувач взаємодіє з сервісами gLite за допомогою утиліт командного рядка. Інтерфейс користувача надає команди: для відправки завдань в Грід-систему, запит стану завдань і кластерів, отримання даних від завершених завдань, переривання завдань та ін. Також є засоби для копіювання і видалення файлів в сховищах і каталогах реплік.

Інтерфейс користувача EMI дозволяє відправляти завдання кількома способами:

- з використанням WMS ***User Interface – UI*** за допомогою системи управління завантаженнями WMS (Workload Management System) на обчислювальні елементи CREAM CE і стандартний LCG-CE;
- за допомогою команд інтерфейсу користувача CREAM CLI безпосередньо на обраний обчислювальний елемент CREAM CE;
- за допомогою команд інтерфейсу користувача ARC CLI на обчислювальні елементи, які працюють під керуванням ППЗ ARC;
- за допомогою команд інтерфейсу користувача ARC CLI безпосередньо на вибраний обчислювальний елемент CREAM CE.

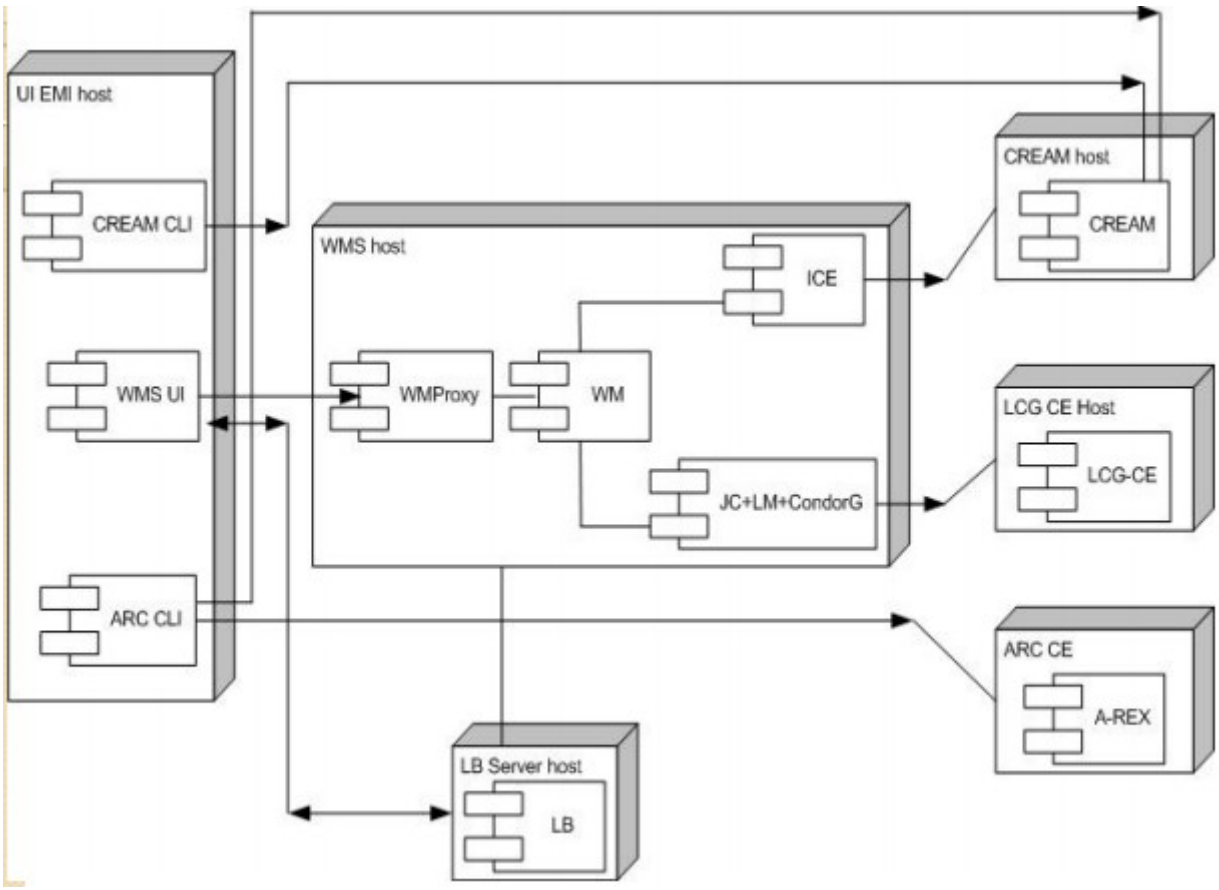


Рис.2.8.3 Загальна схема відправлення завдання в Грід

На рис.2.8.3 приведена загальна схема відправлення завдання в Грід з різними обчислювальними елементами CREAM CE, LCG-CE і ARC CE з одного інтерфейсу користувача.

Розгянемо стисло архітектуру ***Computing Resource Execution and Management (***CREAM), який включено в дистрибутив EMI-2. Більше інформації можна знайти на офіційному сайті.[2]

CREAM розроблений з використанням технології Web Services мовою програмування Java і працює як розширення Java-Axsis сервлету всередині серверу додатків Apache Tomcat.

CREAM і CEMonitor є частиною дистрибутива ППЗ gLite і використовується Грід-інфраструктурою EGI. (EGI – European Grid Infrastructure - проек, який прийшов на зміну EGEE) Користувач може встановити користувацький інтерфейс CREAM і відправляти завдання безпосередньо на обчислювальний елемент CREAM CE командами інтерфейсу користувача.

Крім того, gLite-користувач може відправляти завдання на обчислювальний елемент CREAM CE, використовуючи підсистему розподілу навантаження gLite Workload Management System (WMS). В цьому випадку має бути встановленим спеціальний компонент, названий Interface to Cream Environment (ICE). ICE приймає запити від gLite WMS і направвляє їх на CREAM CE. ICE обробляє стани завдання і реєструє їх в gLite Logging and Bookkeeping (LB) сервісі.[3]

Інтерфейс користувача CREAM UI містить набір команд, які дозволяють відправити завдання на виконання, зняти завдання, отримати звіт про поточний стан виконання завдання і скопіювати результати виконання на комп'ютер користувача.

Використання WMS дозволяє розподіляти завдання між обчислювальними елементами в Грід. При відправленні завдання безпосередньо на CREAM CE контроль стану завдання виконується спеціальним модулем ICE. ICE приймає команди від компонентів WM-системи за допомогою простого механізму повідомлень, заснованого на локальних файлах. ICE може отримувати інформацію про стан завдання двома способами. Перший спосіб використовує спеціальний компонент CEMonitor. CREAM інформує CEMonitor про кожну зміну стану завдання і потім ця інформація передається в ICE. Крім того, ICE може прямо запитувати сервіс CREAM про стан активного завдання, якщо інформація про нього не була отримана певний час. Цей механізм гарантує те, що ICE отримає інформацію про стан завдання навіть у випадку, якщо з модулем CEMonitor виникнуть проблеми.

Для забезпечення сумісності з різними Грід-системами реалізований додатковий сервіс CREAM service, заснований на Basic Execution Service (BES) специфікації. BES визначає основні операції по керуванню завданнями на обчислювальному елементі Грід і аналогічний механізм повинен бути реалізований в інших системах.

A-REX - ARC Compute Element (ARC CE)

Дистрибутив ППЗ EMI включає всі основні компоненти ARC і повністтю забезпечує функціональність з виконання завдань у Грід.

ARC Compute Element являє собою набір сервісів , які реалізують функцію доступу в грід для авторизованих користувачів, відправку завдання на виконання безпосередньо в локальну систему керування, обробку запитів користувачів і публікацію інформації про хід виконання завдання і роботу сервісу в інформаційній системі. Цей сервіс збудований на основі Grid Manager з дадаванням Web Service-інтерфейсу. Він заснований на відкритих стандартах сумісності (включаючи інструментарій OGF Basic Execution Service –BES) і специфікації Job Submission Description Language (JDSL) і підтримує LRMSs (Local Resource Management Systems). Взаємодія інтерфейсу користувача з A-REX забезпечується компонентом ARC Core – набір базових пакетів, які забезпечують реалізацію основних сервісів ARC. Основним пакетом є Hosting Environment Deamon (HED), який являє собою бібліотеку базових сервісів, необхідних для роботи пакетів ARC. HED забезпечує базову функціональність (комунікаційні канали, доступ до бібліотек, авторизацію, аутентифікацію, парсинг конфігураційних файлів для Грід-сервісів), що дозволяє виділити загальні функції Грід-сервісів із загальної логіки і розробляти нові Грід-сервіси дуже єфективно. Інформаційна система ARC Iformation System, яка являє собою набір сервісів для публікації, зберігання і візуалізації інформації, містить в собі ARIS (ARC Resource Information System)- – сервіс публікації інформації про A-REX і про завдання, які виконуються на ресурсі в LDAP-форматі, EGIIS (Enhanced Grid Information Indexing Service) – сервіс реєстрації ARISes і LDAP. Сервіси ARIS

відповідають за опис і характеристики ресурсів (ресурсів обчислень, ресурсів накопичених даних). Інформація про локальні сервіси генерується на самому ресурсі (також можливе її кешування). Доступ до цієї інформації можливий через LDAP-інтерфейс. Основним завданням сервісів EGIIS є зберігання динамічного списку ресурсів (їх LDAP-адреса). Індексні сервіси таким чином поєднуються між собою і реалізують конкретну топологію Грід-сегменту. Перед використанням кожної з команд інтерфейсу користувача EMI необхідно мати (Грід-користувачам) дійсний проксі-сертифікат, доступний системі керування завданнями, який можна створити командами ***grid-proxy-init, voms- proxy-init, arcproxy.*** Вибір команди залежить від типу викристов ППЗ. Якщи проксі-сертифікат вже є на комп'ютері зі встановленим користувацьким інтерфейсом, можна використати змінну середовища з X509_USER_PROXY з зазначенням шляху до нього. Крім цього, в каталозі *SHOME/.globus* треба мати пару ключів, тобто файли ***usercert.pem, userkey.pem*** з відповідними правами доступу.

WMS-інтерфейс користувача

Основні команди інтерфейсу WMS користувача системи gLite дистрибутиву EMI-2. (більш детальну інф. див. http://glite.web.cern.ch/glite/documenataion/)

Основні команди інтерфейсу WMS користувача

Команда	Опис команди
1	3
glite-wms-job-submit<jdl_file>	Відправка завдання на виконання
glite-wms-job-delegate-proxy < delegation_Id>	Делегування проксі-сертифікату до WMProxy та присвоєння відповідного ідентифікатора
glite-wms-job-status<job_Id>	Відображення стану виконання завдання
glite-wms-job-list-match<jdl_file>	Формування переліку обчислювальних ресурсів, які задовольняють вимогам виконання завдання і на яких авторизовано даного користувача
glite-wms-job-cancel<job_Id>	Відміна виконання завдання
glite-wms-job-output<job_Id>	Копіювання вихідних файлів після завершення завдання

Інтерфейс користувача CREAM CE EMI-2

Основні команди інтерфейсу користувача системи ***CREAM CE EMI*** . (більш детальну інф. див. http://glite.web.cern.ch/glite/documenataion/)

Основні команди *CREAM CE EMI*

Команда	Опис команди
1	3
glite-ce-job-submit<jdl_file>	Відправка завдання на виконання, опис якого міститься у файлі <jdl_file>
glite-ce-job-delegate-proxy < delegation_Id>	Делегування проксі-сертифікату до WMProxy та присвоєння відповідного ідентифікатора
glite-ce-job-status<job_Id>	Відображення стану виконання завдання
glite-ce-job-list <host[:port] >	Відображення переліку ідентифікаторів завдань, відправлених на виконання на CREAM CE користувачем
glite-ce-job-cancel<job_Id>	Відміна виконання завдання
glite-ce-job-suspend<job_Id>	Тимчасове припинення виконання раніше відправлених завдань користувача на CREAM CEs (кількість завдань має бути >=1)
glite-ce-job-resum<job_Id>	Відновлення виконання раніше відправлених завдань користувача на CREAM CEs (кількість завдань має бути >=1)
glite-ce-job-output<job_Id>	Копыювання вихыдних файлыв пысля завершення завдання
glite-ce-job-purge<job_Id>	Видалення інформації по N завданням користувача з CREAM CEs (кількість завдань має бути >=1). Після цієї операції інформація (вхідні та вихідні файли буде недоступна.
glite-ce-proxy-renew < delegation_Id>	Продовження дії проксі-сертифікату для відпралених на виконання завдань на CREAM CEs за ідентифікатором, який був створений по команді glite-ce-job-delegate-proxy
glite-ce-service-info <host[:port] >	Відображення інформації по сервісам CREAM (версія, статус та інше)
glite-ce-get-cemon-url <host[:port] >	Відображення інформації по сервісе CEMon CREAM CE
glite-ce-enable-submission <host[:port] >	Відновлення можливості відправки завдань на окремий CREAM CE

glite-ce-disable-submission <host[:port]>	Заборона на відправку завдань на окремий CREAM CE
glite-ce-allowed-submission<host[:port]>	Перевірка можливості відправки завдання на окремий CREAM CE

Перед використанням кожної з команд CREAM CE необхідно мати дійсний проксі-сертифікат, доступний системі управління завданнями. Його можна створити за допомогою команди ***voms-proxy-init.***

Опис стану завдання в CREAM

Стан завдання	Опис стану завдання
1	3
Registered	Завдання відправлено на виконання на обчислювальний елемент CREAM
pending	Завдання оброблюється
idle	Завдання знаходиться в локальній черзі LRMS та очікує відправки на виконання
running	Завдання оброблюється в LRMS
Really-running	Завдання виконується
held	виконання завдання призупинено командою користувача
Done-ok	виконання завдання завершилося успішно
Done-failed	виконання завдання завершилося з помилками
cancelltd	виконання завдання відмінено командою користувача
aborted	Помилка при відправленні завдання в локальну систему керування LRMS

Інтерфейс користувача ARC – EMI

Наведемо основні команди користувацького інтерфейсу ***ARC – EMI***. (більш детальну інф. див. http://www.nordugrid.org/ в розділі Документація)

Команда	Опис команди
1	3
arcproxy [options]	Генерування проксі-сертифікату
arclcs [options]	Генерування проксі-сертифікату IdP для SAML2SSO профілю
arcsub [options] [filename …]	Відправлення завдання на грід-ресурси для виконання. Завдання має бути підготовлене

	в одній із мов опису JSDL, xRSL, JDL
arctat [options] [job …]	Перегляд стану кластерів і завдань
arccat [options] [job …]	Перегляд потокового виведення результатів та повідомлень про помилки під час виконання завдання
arcget [options] [job …]	Копіювання вихідних файлів після завершення завдання
arcsync [options]	Синхронізація інформації про виконання завдань, яка зберігається на одному комп'ютері з інформацією на ARC clientкомп'ютері, з якого започаткована поточна сесія роботи
arcinfo [options]	Отримання статичної інформації про обчислювальний ресурс в Грід
arckill [options] [job …]	Відміна виконання завдання
arcclean [options] [job …]	Видалення файлів виведення результатів виконання завдання з кластеру, в якому вони виконувалися
arcrenew [options] [job …]	Відновлення проксі-сертифікату завдання
arcresume [options] [job …]	Відновлення роботи завдання після збою при виконанні
arcresub [options] [job …]	Повторна відправка завдання на виконання
arcmigrate [options] [job …]	відправка завдання на виконання в інший обчислювальний елемент з припиненням поточного виконання на ресурсі (використовується тільки для web-сервісу A-REX)
arcls [options] <URL>	Перегляд змісту та деяких атрибутів файлів на ресурсі зберігання даних в певній директорії, яка задається за допомогою URL
arccp [options] <source> <destination>	Команда копіювання файлів між грід-кластерами
arcrm [options] <URL>	Видалення файлів з ресурсу зберігання даних
arcmkdir [options] <URL>	Команда створення нової директорії на ресурсі зберігання даних, який задано за допомогою URL
arcsrmping [options] <service>	Тестування доступності SRM-сервісу (подібна команді ping в Unix)
arctest [options]	Тестування функціональності грід-ресурсу

Перед використанням кожної з команд користувацького інтерфейсу ARC – EMI потрібно мати дійсний проксі-сертифікат (створюється за допомогою команди ***arcproxy. Arcproxy*** дозволяє генерувати такі проксі-сертифікати, які

використовуються в різних грід: pre-RFC GSI proxy; RFC-compliant proxy (default); VOMS-extended proxy; MyProxy delegation.

При використанні грід-ресурсів, які вимагають членства користувача в Віртуальній організації, кманда ***arcproxy*** може генерувати проксі-сертифікат з розширенням VOMS, яке засвідчує належність користувача до певної ВО та надає право доступу до її Грід-ресурсів:

```
$ arcproxy --voms atlas
Your identity: /DC=***/O=***/CN=***
Enter pass phrase for /home/user/.cert/userkey.pem:
.....++++++
.....++++++
Contacting VOMS server (named atlas): voms.cern.ch on port: 15001
Proxy generation succeeded
Your proxy is valid until: 2018-05-24 03:08:35
```

При цьому потрібно додати опис віртуальної організації в файлі ~/***.voms/vomses.***

Опис завдання в ARC –EMI виконується мовою xRSL (Extended Rtsource Specification Language) і має містити всю необхідну для виконання на віддаленому ресурсі інформацію (файл виконання, параметри та ін.) і набір вимог, яким маэ відповідати кластер для виконання завдання (об'єм диску, встановлене програмне забезпечення) і куди мають бути записані результати виконання.

Приклад завдання:

```
&
(executable=task.sh)
(inputFiles=(task.sh "")( task.c ""))
(stdout="out.txt")
(stderr="err.txt")
(outputFiles=("out.txt" "")("err.txt" ""))
(jobname="test1")
```

Тут вказано, що в якості завдання використовується файл скрипту ***task.sh.*** На віддалений ресурс потрібно скопіювати файл ***task.sh*** і ***task.c***, а результати виконання будуть збережені в файлі ***out.txt,*** файл реєстрації помилок - ***err.txt.***

Якщо завдання успішно відправлене на виконання на віддалений обчислювальний ресурс, то йому (завданню) присвоюється ідентифікатор ***job ID,*** який відображається в термінальному вікні (standard output).

В ARC – EMI опис завдання можна виконувати мовою ***JSDL*** та ***JDL.*** Так, в прикладі ***myjob.jsdl*** наведено вивід "***Hello World***" з використанням мови ***JSDL.***

```
<version=”1.0” encoding =”UTF-8”?>
<JobDefinition
xmlns=”http://schemas.ggf.org/jsdl/2015/11/jsdl”
xmlns:posix=”http://schemas.ggf.org/jsdl/2015/11/jsdl- posix”>
<JobDescription>
        <JobIdentification>
                <JobName>Hello World job</JobName>
        </JobIdentification>
        <Application>
        <posix:POSIXApplication>
                <posix:Executable>bin/echo</posix:Executable>
                <posix:Argument>’Hello World’</posix:Argument>
                <posix:Output>out.txt</posix:Output>
                <posix:Error>err.txt</posix:Error>
        </posix:POSIXApplication>
      </Application>
</JobDescription>
</JobDefinition>
```

В наступному прикладі - опис завдання з виведенням "***Hello World***" виконано мовою JDL.

```
[
Executable = “script-jdl.sh”;
StdOutput =“std.out”;
StdError =“std.err”;
OutputStandbox = {“std.out”,“std.err”};
OutputStandboxDestURL = {“gsiftp://localhost/std.out”,”gsiftp://
localhost/std.err”};
]
```

Файл скрипту *script-jdl.sh* має вигляд:

```
#!/bin/sh
Echo “Example forJDL file”
```

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення Грід.
2. Перерахуйте особливості програмування Грід.
3. Що таке координація розрізнених ресурсів?
4. На що націлені Грід - обчислення ?
5. Назвати обчислювальні ресурси Грід.
6. Що таке Сумісніять midlleware Grid?
7. Призначення Middleware.
8. Перерахуйте найбільш відомі пакетіи ППЗ Грід.
9. Дати визначення Функціональної сумісністі (Interoperaility).
10. Охарактеризуйте Взаємодію (Interoperation).
11. Що таке Сценарій користувача?
12. Охарактеризуйте особливості алгоритмів планування розміщення завдання на розподілених ресурсах.
13. Що являє собою підпроект European Middleware Initiative (EMI)?
14. Охарактеризуйте архітектуру Computing Resource Execution and Management (CREAM).

Література основна

1. За редакцією Загороднього А.Г. та Згуровського М.З., На шляху до європейського грід: довідник (проект). - К.: НТУУ «КПІ», 2012. − 391 с

Література допоміжна

2.http://www.eu-emi.eu/
3.http://www.eu-emi-2-matterhorn
4. LCG middleware documentation: режим доступу: http://grid-deplomynent.web.cern.ch/
5.www.unicore.org
6.www.globus.org
7.www.gLite.org
8.http://www.nordugrid.org/

2.9 Як розпочати працювати в Грід

В даній лекції описуються основні кроки, які необхідно виконати для того, щоб розпочати практичну роботу в Грід. Представлені основи політики безпеки для користувачів, які реалізуються в Грід.

Інфраструктура Грід складається з ресурсних центрів (які надають користувачам Грід обчислювальні і дискові ресурси) і інфраструктурних центрів, призначених для координації функціонування інфраструктури.

Обчислювальні ресурси, як правило, представляють собою кластери, побудовані зі стандартних комп'ютерів, об'єднаних мережею. Розмір такого кластера може досягати декількох сотень і тисяч вузлів. Дискові ресурси створюються на основі роботизованих стрічкових бібліотек і / або великих дискових масивів на жорстких дисках. Обсяг дискових ресурсів може варіюватися від сотень гігабайт до декількох петабайт.

Для використання інфраструктури Грід користувачеві необхідно пройти процес реєстрації, після чого всі ресурси Грід стають доступні йому без будь-яких додаткових угод з окремими ресурсними центрами. Процес реєстрації нового користувача включає в себе два основних етапи:

1. Отримання персонального, призначеного для користувача сертифікату
2. Реєстрація в Віртуальній організації

Персональний (власний) сертифікат (Personal Certificate) - це свого роду електронний документ, що підтверджує особу користувача при доступі до Грід-ресурсів. Сертифікати видаються Центрами сертифікації (Certification Authority).

Віртуальна організація (Virtual Organization) - це співтовариство користувачів, які спільно використовують обчислювальні ресурси відповідно до узгоджених між ними і власниками ресурсів правилами. Ці правила регулюють доступ до всіх типів засобів, включаючи комп'ютери, програмне забезпечення та дані. Кожна віртуальна організація має свій власний Центр реєстрації. Також існує можливість почати роботу в Грід, скориставшись учбовими - і демо-проектами. Прикладом такого проекту є GILDA(Grid Infn Laboratory for Dissemination Activities) - https://gilda.ct.infn.it/

Основи безпечної роботи в Грід

Система входу користувача в Грід-систему досить проста, проте, вона вирішує кілька важливих завдань за допомогою методів комп'ютерної криптографії.

Перше завдання - як зашифрувати ту інформацію, яка передається в Грід, особливо параметри, які пов'язані зі входом в Грід-систему. Для вирішення цього завдання використовується технологія асиметричного шифрування (шифрування з «відкритим ключем»). Кожен користувач, або ресурс має пару ключів: відкритий (public), який доступний для всіх, і закритий (private), доступ до якого має тільки його власник. При цьому практично неможливо підібрати другий ключ з пари, володіючи тільки одним. Всі повідомлення кодуються і розкодовуються, використовуючи цю пару ключів. Шифрування виконується відкритим ключем одержувача, а розшифрування - закритим. Режим цифрового підпису реалізується в зворотному порядку: закритий ключ відправника - створення підпису, відкритий ключ - її перевірка, при цьому цифровий підпис може створити тільки власник закритого ключа. Однак варто зауважити, що володіння парою ключів автоматично не вирішує завдання аутентифікації - тобто перевірки аутентичності користувачів і ресурсів Грід з метою запобігання

проникнення зловмисників. Як бути впевненим в тому, що повідомлення, зашифроване закритим ключем, насправді належить тій особі, за кого вона себе видає? Цю другу задачу вирішують цифрові сертифікати.

Цифровий сертифікат - це відкритий ключ власника сертифікату з інтегрованою персональною інформацією, такою як ім'я користувача, його електронна адреса, місце роботи, термін дії сертифікату і т.п. Крім того, сертифікат містить цифровий підпис сертифікаційного центру (Certification Authority - CA), деякої третьої сторони, яка засвідчує приналежність сертифікату тому, чиї дані записані в сертифікаті.

Цифровий підпис - це деяка інформація про сертифікат, зашифрована за допомогою закритого ключа CA. Таким чином, прочитати цю інформацію можна тільки за допомогою відкритого ключа CA, який відомий всім. Довіра сертифікату будується на довірі сертифікаційному центру, який підписав цей сертифікат. Перед видачею (підписанням) сертифікату завдання CA - перевірити приналежність сертифікату даному індивідууму.

Кожен сертифікаційний центр проводить свою політику, яка визначає правила створення і підписання сертифікатів (рис.2.9.1). Зазвичай центри сертифікації існують або в рамках окремих організацій, які беруть участь в будь-яких проектах Грід (наприклад, CERN або INFN), або в рамках цілого проекту, або країни.

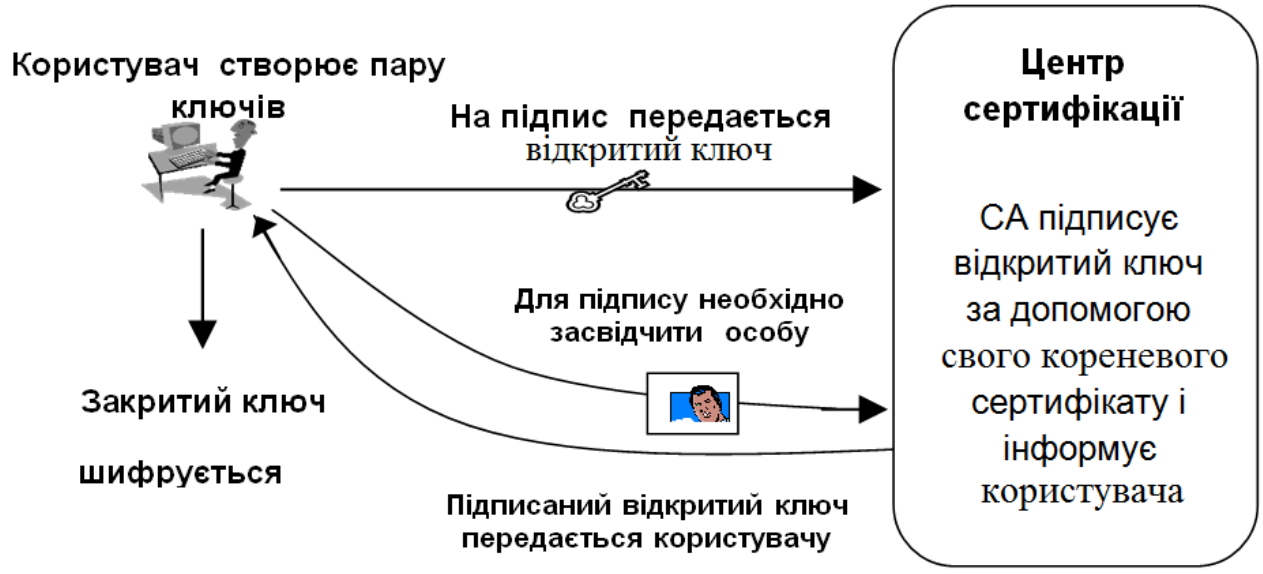


Рис. 2.9.1. Отримання цифрового сертифікату

Реалізація способу отримання сертифікату залежить від конкретного центру сертифікації. Зараз, як правило, для отримання сертифікату користувач повинен зайти на Web-сайт центру сертифікації і заповнити форму із запитом на видачу сертифікату. При цьому вказуються персональні дані і назва організації. Обов'язково потрібно вказати правильну адресу електронної пошти, так як відповідна інформація про отримання сертифікату буде надіслана за цією адресою. Іноді додатково потрібно більш детальна інформація, яка характеризує необхідність отримання сертифікату саме в цьому CA: про область наукових інтересів, проведених робіт, виконуваних проектах, і т.п. Але як в центрі сертифікації, який може знаходитися в іншому місті, або навіть

іншій країні, можуть перевірити докази ідентичності особистості? Для цієї мети в більшості масштабних Грід-проектів передбачено інститут реєстраторів (Registration Authority - RA), або довірених осіб, які засвідчують належність сертифікуємого об'єкту (користувача, ресурсу або сервісу) до певної організації і підтверджують персональні дані власника сертифікату. Тому, перш ніж підписати сертифікат, CA зв'язується з відповідним RA і отримує від нього підтвердження автентичності запиту на отримання сертифікату. В результаті, якщо рішення про видачу сертифікату прийнято, на вказану адресу електронної пошти приходить лист з подальшими інструкціями. Як правило, це спосіб отримання - або безпосередній імпорт сертифікату в браузер, тобто просто перехід за певним посиланням в браузері, або запуск надісланого в листі скрипта на комп'ютері з встановленим пакетом openssl. Залежно від способу створення сертифікату він може бути збережений в різних форматах (PKCS12 або PFX, DER, PEM). Для того щоб в подальшому використовувати отриманий сертифікат для роботи через Web, його необхідно експортувати в формат, що розпізнається браузером. Процедура експорту і тип файлу експортованого сертифікату залежить від браузеру (*.p12 - для Mozilla/Netscape/FireFox , *.pfx - для Internet Explorer).

Після отримання цифрового сертифікату користувачеві необхідно зареєструватися в віртуальної організації. Залежно від області роботи користувача це може бути міжнародна, національна або локальна віртуальна організація. Правила реєстрації в віртуальній організації необхідно дізнатися у відповідному Центрі реєстрації (не плутати з Центром сертифікації). Можлива реєстрація одного і того ж користувача (сертифіката) в декількох віртуальних організаціях. Ставши членом однієї з віртуальних організацій, користувач отримує можливість працювати в Грід. Але будь-який сеанс роботи в Грід повинен починатися зі створення тимчасового проксі-сертифікату. Цей сертифікат служить мірою безпеки для запобігання використання основного сертифікату зловмисниками. Проксі-сертифікат має невеликий термін дії (зазвичай не більше 24 годин) і використовується для підтвердження особи користувача при виконанні будь-яких операцій в Грід.

Далі розглянемо дії, які повинен виконати користувач для того, щоб отримати доступ до Грід і почати працювати в ньому.

1. Перш за все, потрібно бути зареєстрованим користувачем комп’ютера, на якій встановлено призначений для користувача інтерфейс Грід (User Interface).
2. Отримати призначений для користувача цифровий сертифікат в CA RDIG. Для цього:

• Ознайомитися з документом "RDIG Certification Authority Certificate Policy and Certification Practice Statement"

• Зайти на сторінку "Отримання нового користувацького сертифікату" і заповнити електронну форму, дотримуючись інструкції. Заповнивши всі поля форми, натисніть кнопку "Далі".

• Зберегти на своєму локальному комп'ютері отриманий файл сценарію new_cert.sh.

• Зайти на cluster через ssh (або putty з-під Windows), скопіювати в свій домашній каталог файл new_cert.sh і запустити його (chmod + x new_cert.sh; ./new_cert.sh).
• В процесі виконання сценарію Вам потрібно буде двічі ввести пароль, яким буде захищений сертифікат. Правила вимагають, щоб пароль мав довжину не менше 15 символів. Ви повинні добре запам'ятати цей пароль, відновити його в разі втрати буде неможливо.
• В результаті виконання сценарію в вашому домашньому каталозі буде створена директорія .globus, в якій будуть створені кілька файлів, в тім рахунку:

certreq.mail – файл з запитом для центру сертифікації. Цей файл відправляється в CA.
userkey.pem – закритий ключ.
usercert.pem – порожній файл. У ньому потрібно буде зберегти підписаний відкритий ключ після отримання його з CA.

Пам'ятайте, що якщо ви втратите будь-який з цих файлів, то відновити його буде неможливо, і потрібно буде отримувати сертифікат заново.

• Надішліть сформований сценарієм файл запиту certreq.mail за адресою rdig-ca. Файл запиту має бути відправлений в тілі листа в текстовому вигляді, а не прикріплений до листа.

• У разі правильно сформованого запиту на адресу електронної пошти, вказаної при заповненні форми, прийде повідомлення з порядковим номером вашого запиту. Цей номер знадобиться при подальшому спілкуванні з відповідальним за реєстрацію нових користувачів (RA - Registration Authority), тому його слід запам'ятати.
• Заповніть бланк запиту, після чого зв'яжіться з RA вашої організації.
• Після підтвердження ваших персональних даних та правильності Вашого запиту RA посилає підтвердження запиту в Центр сертифікації.
• Після отримання підтвердження Ваш сертифікат буде підписаний і висланий на вказану Вами електронну адресу протягом трьох робочих днів.
• При отриманні листа з підписаним сертифікатом збережіть його під ім'ям usercert.pem в підкаталозі .globus вашого домашнього каталогу. Сконвертувати цифровий сертифікат в формат PKCS12 для імпорту в браузер. Для цього, перебуваючи в підкаталозі .globus вашого домашнього каталогу, необхідно виконати команду:
openssl pkcs12 -export -inkey userkey.pem -in usercert.pem –out cert.p12 -name "My certificate"
Одержаний в результаті файл cert.p12 потрібно завантажити в браузер. Докладні інструкції (англійською мовою) доступні на сторінці https://lcg-registrar.cern.ch/load_certificates.html.

Доступ до Грід-ресурсів і сервісів

Доступ до ресурсів і сервісів для кінцевого користувача є одним з найбільш важливих компонентів Грід-системи. Ця компонента отримала назву User Interface (***UI***). Їй приділяється особлива увага, так як вона є ключовою сполучною ланкою між Грід і кінцевим користувачем. Основним інтерфейсом роботи з ***UI*** є інтерфейс командного рядка (Command Line Interface - CLI).

Розглянемо деякі команди інтерфейсу командного рядка, знання яких необхідно для початку роботи в Грід . Перед початком роботи користувачеві необхідно створити так званий проксі-сертифікат. Даний сертифікат обмежений за часом (за замовчуванням - 12 годинами) і передається в Грід-систему разом із завданням. Цей сертифікат дозволяє завданню виконувати операції від імені користувача, який запустив її (авторизація, робота з даними, запуск підзадач і т.п.):

grid-proxy-init

Буде необхідно ввести пароль, який був зазначений при отриманні персонального цифрового сертифікату, наприклад

[ole@ui ole]$ grid-proxy-init
Your identity: /C=RU/O=RDIG/OU=users/OU=pnpi.nw.xx/CN=Sergey Oleshko
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Fri Oct 6 02:48:13 2018

Для отримання інформації про проксі-сертифікат можна скористатися, наприклад, наступною командою:

grid-proxy-info –all

[ole@ui ole]$ grid-proxy-info
subject:/C=RU/O=RDIG/OU=users/OU=pnpi.nw.xx/CN=Sergey Oleshko/CN=proxy
issuer : /C=RU/O=RDIG/OU=users/OU=pnpi.nw.xx/CN=Sergey Oleshko

type : full legacy globus proxy
strength : 512 bits
path : /tmp/x509up_u10012
timeleft : 11:53:58

Після закінчення сеансу роботи проксі-сертифікат може бути видалений командою **grid-proxy-destroy**

Одним з важливих обмежень проксі-сертифікату є те, що завдання, запущені з даними проксі-сертифікатом, не може виконуватися довше, ніж час життя цього сертифіката. Дане обмеження введено з міркувань безпеки. Якщо користувачеві необхідно запустити завдання на тривалий час, але точно передбачити цей час в момент запуску завдання не представляється можливим, то проксі-сертифікат можна зареєструвати на сервері автоматичного

поновлення сертифікатів (myproxy). При цьому адреса myproxy server може бути вказана в команді, або береться з змінної оточення MYPROXY_SERVER.

Реєстрації проксі-сертифіката на сервері:

myproxy-init -s <сервер> -t <время регистрации>

За замовчуванням адреса myproxy server береться з змінної оточення MYPROXY_SERVER, час реєстрації за замовчуванням - 7 днів (168 годин). При реєстрації на myproxy server необхідно два рази ввести пароль, яким буде захищений проксі-сертифікат для автоматичного оновлення.

[ole@ui ole]$ myproxy-init
Your identity: /C=RU/O=RDIG/OU=users/OU=pnpi.nw.ru/CN=Petro Petrenko
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Thu Oct 12 15:57:25 2018
Enter MyProxy pass phrase:
Verifying password - Enter MyProxy pass phrase:
A proxy valid for 168 hours (7.0 days) for user ole now exists on myproxy.cern.ch.

Отримання автоматично оновленого проксі-сертифікату:
myproxy-get-delegation -s <сервер>

Отримання інформації про зареєстрований проксі-сертифікат:
myproxy-info -s <сервер>
[ole@ui ole]$ myproxy-info
username: ole
owner: /C=RU/O=RDIG/OU=users/OU=pnpi.nw.ru/CN= Petro Petrenko
timeleft: 167:36:07 (7.0 days)

Скасування реєстрації:
myproxy-destroy -s <сервер>

Після проведення всіх підготовчих операцій з проксі-сертифікатами можна приступати до запуску завдань в Грід-системі.

Робота в українському Грід

Ще раз звертаю Вашу увагу, що для того, щоб працювати в Грід потрібно бути членом віртуальної організації і мати цифровий сертифікат користувача, отриманий в CA RDIG. Для цього перш за все потрібно бути зареєстрованим користувачем комп'ютера, на який встановлений користувацький інтерфейс Грід (User Interface). Це означає,що Ви маєте бути зареєстрованим користувачем обчислювального кластеру (напр.,cluster.nw.xx.). Якщо у Вас нема такого права, то Ви маєте зв'язатися з адміністратором цього кластеру, або послати листа на globus@pnpi.nw.xx з інформацією про себе і типах задач, які будуть запускатися в Грід. Якщо Ви плануєте працювати по програмі

одного з експериментів LHC, то Ваш запит має бути підтвердженим керівником чи відповідальним за даний експеримент. Отже, потрібно мати справжній (“бойовий”) сертифікат користувача.

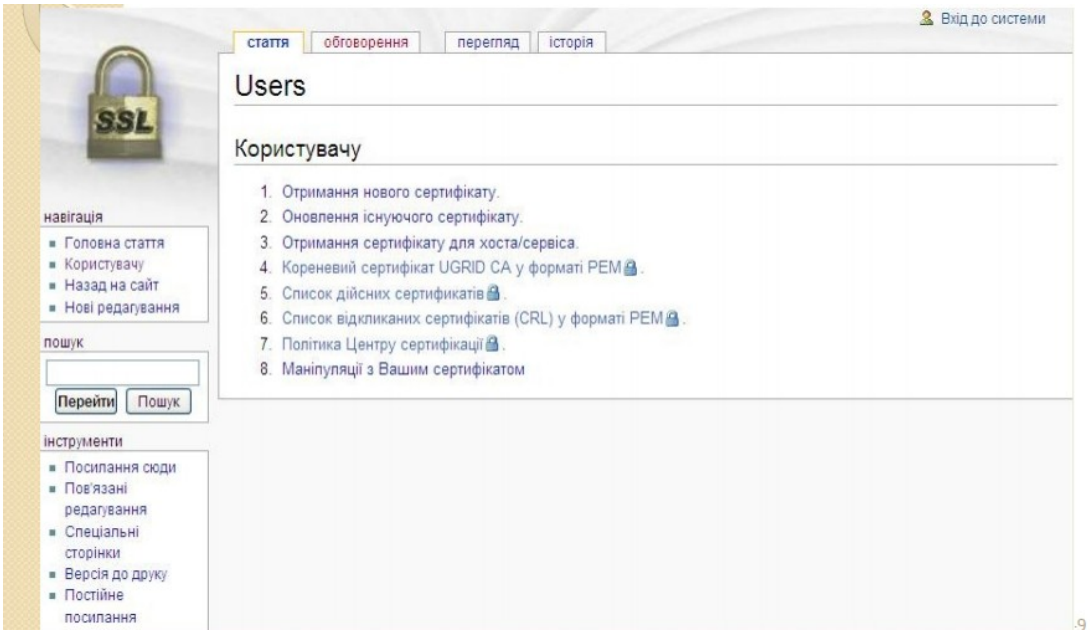
Як Розпочати роботу в українському Грід

1. Отримати цифровий *сертифікат користувача.*
2. Отримати доступ (*account*) до UI (U*ser* I*nterface*).

3. Зареєструватися в ВО *(Віртуальній Організації).*

Як отримати сертифікат?

•Ознайомитися з правилами отримання сертифікату на сайті https://ca.ugrid.org/wiki/index.php/Users (рис.2.9.2)



Отримання нового сертифікату

UGRID CA підписує сертифікати трьох типів: сертифікати користувачів, сертифікати вузлів та сертифікати сервісів. Для отримання нового сертифікату користувача необхідно зробити наступне:

1. Уважно ознайомитись з поточною політикою видачі сертифікатів.
2. Абонент (майбутній користувач) заповнює форму на сайті і отримує серійний код, під яким зберігається його запит. Необхідно обов'язково записати або запам'ятати цей код. Поля "First name"(Ім'я) та "Last name" (Прізвище) потрібно заповнювати за правилами транслітерації української мови латиницею або так, як записано у закордонному паспорті.
3. За допомогою інструкцій абонент:
    1. генерує запит на підписування сертифікату (**УВАГА ДОВЖИНА ПАРОЛЮ НЕ МЕНШЕ 15 СИМВОЛІВ**),
    2. відсилає запит на підписування сертифікату у формі https://ca.ugrid.org/process.php або електронною поштою ОБОВ'ЯЗКОВО з тієї електронної адреси, яку він вказав при реєстрації,
    3. надсилає у Центр Сертифікації лист електронною поштою ОБОВ'ЯЗКОВО з тієї електронної адреси, яку він вказав при реєстрації, у листі вказати серійний код, отриманий при заповнені форми на сайті (п.1).
4. Центр Сертифікації надішле лист про те, що запит прийнято до розгляду.
5. Абонент заповнює та підписує Запит на отримання сертифікату користувача.
6. Абонент ОСОБИСТО звертається до Центру Сертифікації (або до Реєстраційних Центрів), який підтвердить Ваші персональні дані, належність до Вашої організації та вірність Вашого запиту. При собі необхідно мати
    1. національний паспорт,
    2. закордонний паспорт (якщо є),
    3. копію першої сторінки та сторінки з останньою фотографією національного паспорту (у 2 екз.)
    4. документ, що засвідчує причетність особи до досліджень у галузі Grid,
    5. 2 копії заповненого та підписаного Запиту на отримання сертифікату користувача.
7. Після отримання листа від абонента (що є засвідченням приналежності йому вказаної електронної адреси) Центр Сертифікації підпише сертифікат протягом трьох днів та надішле подальші інструкції, як його отримати.

Рис.2.9.2

На сайті *https://ca.ugrid.org/* виконати всі інструкції по формуванню запиту на сертифікат (рис.2.9.3)

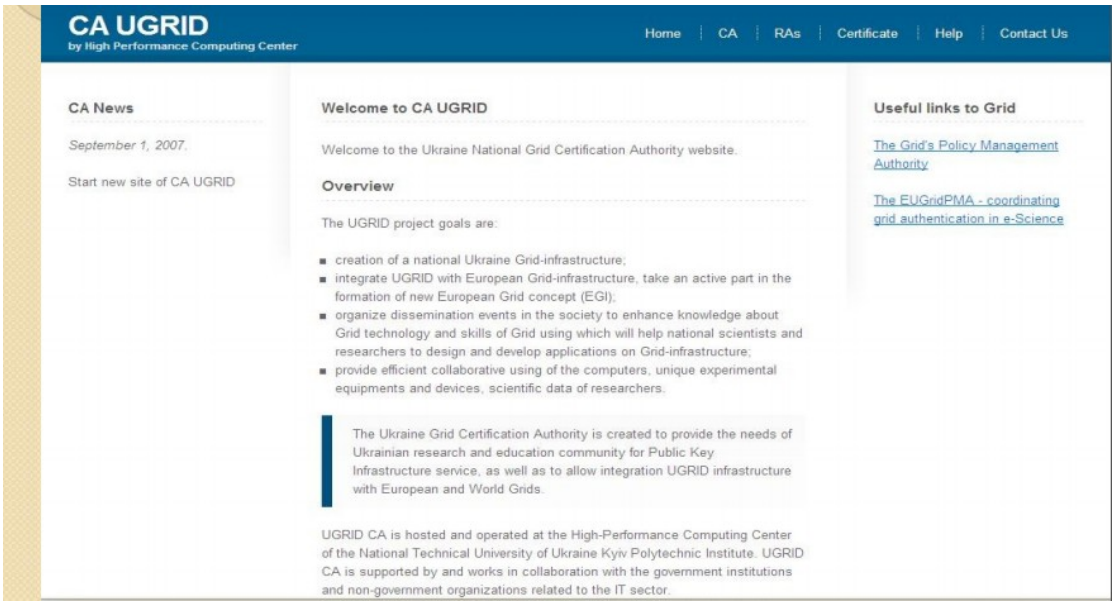


Рис.2.9.3 Формування запиту на сертифікат

Результатом успішної сертифікації являються два файли, поміщені в каталог *.globus,* створений автоматично при генерації запиту в Вашому домашньом каталозі:
-файл з відкритим ключем (*public key*), куди Ви скопіюєте підписаний сертифікат: *usercert.pem*
-файл, який містить закритий ключ (*private key*): *userkey.pem*

Далі необхідно зареєструватися в віртуальній організації.

В КПІ ім. І.Сікорського не існує навчального Грід і відповідно нема сертифікаціонного центру для для видачі учбових сертифікатів студентам, як і нема як такого окремого навчального Грід-центру в Україні. Окремі академічні і освітні установи України роблять цю роботу самотужки (наприклад, КДУ ім. Т. Г. Шевченка, інститут теоретичної фізики НАН України), тому скористаємось їх досвідом.

Необхідно зареєструватися в їхній учбовій віртуальній організації. Наприклад, в ІТФ НАНУ - це bitpedu. Для реєстрації необхідно в браузері з установленим *.p12 сертификатом зайти на ВОМС сервер
https://voms.bitp.kiev.ua:8443/voms/bitpedu/user/home.action

Для *конвертації* цифрового сертифікату в цей формат в підкаталозі *.globus* потрібно виконати команду (наприклад):

[Litov@Litov .globus]$ openssl pkcs12 -export -inkey userkey.pem -in usercert.pem -out cert.p12 -name "MyCertificate"

В вікні реєстрації потрібно обов'язково вказати адресу електронної пошти, на яку прийде повідомлення (рис.2.9.4)

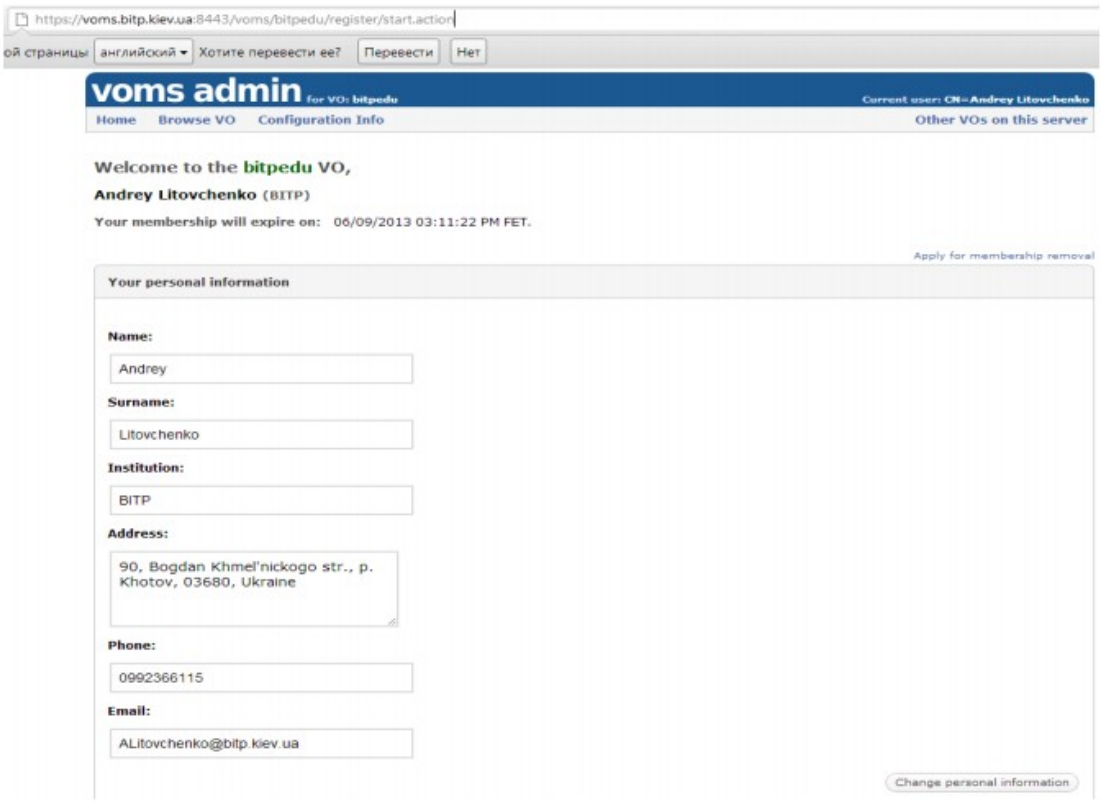


Рис.2.9.4 Вікно реєстрації

Після заповнення реєстраційної форми на вказану адресу поступить лист підтвердження реєстрації (потрібно виконати всі інструкції, вказані в листі). Після завершення процесу реєстрації в списку користувачів буде: (рис.2.9.5)

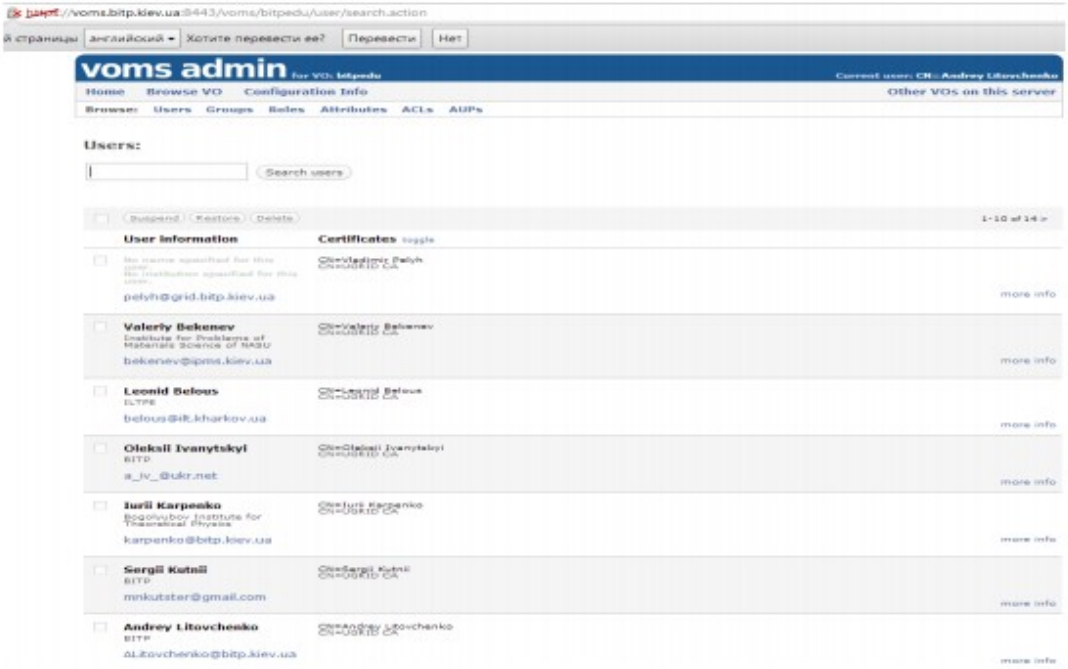


Рис.2.9.5 Список користувачів

Як запустити завдання?

Зайти на ***UI***.
Отримати proxy-сертифікат.
Створити ***JDL****-файл* з описом завдання.
Запустити задання за допомогою команди
*glite-wms-job-submit**

**тут і далі використовуються команди системи управління завданням* EMI-2.

Отримав account на UI (vobox-edu.bitp.kiev.ua),
виконати вхід (наприклад):
[litov@litov ~]$ ssh Litovchenko@vobox-edu.bitp.kiev.ua
*Litovchenko@vobox-edu.bitp.kiev.ua's password: ********
Last login: Thu Jun 18 15:39:46 2012 from 194.44.37.191
[Litov@vobox-edu ~]

Розглянемо попередні дії.

1. Ідентифікація для роботи в середовищі Грід:
-створити директорію *.globus* (при першому вході).
-скопіювати в туди два файли, згенеровані при
отриманні сертифікату: *usercert.pem, userkey.pem.*
[Litov@vobox-edu ~]$ ls -l ~/ .globus
total 12
-r--r--r-- 1 Litov Litov 5669 Oct 20 14:37 usercert.pem
-r-------- 1 Litov Litov 963 Oct 20 14:38 userkey.pem
Увага! Ці файли необхідно оновлювати після кожного

оновлення сертифікату (1 раз в рік).

2. Створення проксі-сертифікату для
отримання доступу до ресурсів Грід.
-передбачаєтьсят, що Ви вже член VO (наприклад, bitpedu).
-тоді проксі-сертифікат створюється командою
voms-proxy-init -voms bitpedu
По замовчуванню проксі-сертифікат створюється на 12 годин. Якщо необхідність в ньому відпала раніше, его потрібно анулювати:

voms-proxy-destroy

Приклад:

[Litov@vobox-edu ~]$ voms-proxy-init -voms bitpedu
Enter GRID pass phrase:

Your identity:
/DC=org/DC=ugrid/O=people/O=BITP/OU=BITP/CN=
Andrey Litovchenko
Creating temporary proxy ..... Done
Contacting voms.bitp.kiev.ua:15001
[/DC=org/DC=ugrid/O=hosts/O=BITP/OU=High Energy Physics
Department/CN=voms.bitp.kiev.ua] "bitpedu" Done
Creating proxy ..... Done
Your proxy is valid until Thu Dec 2 00:52:19 2012

"Стандартний" *proxy* предоставлє не більше 12 годин для виконання завдань. Можливе продовження *proxy* сервером автоматичного оновлення сертифікатів (*myproxy*). Для цього потрібно зареєструвати *proxy*-сертифікат на сервері командою *myproxy-init -s < сервер> -t < время реєстрації*>. По замовчуванню адреса *myproxy*-серверу береться із змінної оточення MYPROXY_S ERVER , а час реєстрації по замовчуванню складає 168 годин (7 днів).

3. Явне делегування повноважень (рекомендуєма, але не обов'язкова дія).
WMProxy-сервіс взаємодіє з WMS від імені користувача, тому отанній повинен делегувати йому свої повноваження.
Існують два способи делегування – автоматичне і явне. Перший спосіб реалізується в командах за допомогою опції ***-a***.
Другий – за допомогою опції ***--delegationid*** (***-d***) з вказуванням ідентифікатору делегування, який визначається командою

glite-wms-job-delegate-proxy -d <userdelegID>

4. Опис задання (***JDL-файл***)
Файл з описом задання – це текстовий файл на мові
JDL (Job Description Language), утримуючий рядки в вигляді пар:
attribute = "expression";
Атрибути в основному визначають:

-тип завдання;
-використовуємі файли (вхідні і вихідні);
-вимоги до обчислювальних ресурсів.

Перш ніж запускати завдання, корисно перевірити які *обчислювальні елементи* (CE) доступні для його виконання. Це реалізується командою:*glite-wms-job-list-match -a hello.jdl* з автоматичним делегуванням, чи
glite-wms-job-list-match -d $USER script.jdl з явним делегуванням. Крім того, ця команда позволяє перевірити синтаксис *JDL-файлу*. Однако, вона примінима тільки для простих завдань.

Запуск завдання

На прикладі простого задання оглянем команди CLI (Command Line Interface), доступні користувачу при його виконанні.

1. Команда запуску завдання:
glite-wms -job-s ubmit -a -o jobid hello .jdl
-a - автоматичне делегування повноважень *WMProxy*;
-o - направляє в файл *jobid* ідентифікатор завдання виду
https://wms-emi.bitp.kiev.ua:9000/In79SvkINSnoGMVaWlQzww
(ця опція позволяє надалі вказувати коротке ім'я файлу, а не сам громіздкий ідентифікатор).

2. Команда, яка показує поточний статус завдання:
- з файлом, який містить ідентифікатор завдання
glite-w ms -job-s tatus -i jobid
- з ідентифікатором завдання
glite-w ms -job-s tatus https ://w ms emi.bitp.kiev.ua:9000/In79S vkIN S noG M VaWlQzw w
Зручність першого варіанту очевидна.

3. Команда отримання результатів вконання завдання:
glite-w ms -job-output -i jobid

В цьому випадку всі результуючі файли поміщаються в каталог ***/tmp/username_<jobID>.*** Однако зручніше отримувати результат в каталозі, визначаємому за допомогою опції ***--dir***:

glite-w ms -job-output --dir path_name -i jobid

4. Команда зняття завдання з виконання:
glite-w ms -job-c anc el <jobID>

Ця команда перш, ніж зняти завдання, запитує вкористувача підтвердження на виконання цієї операції. При підтвердженні завдання знімається з відповідним повідомленням.

Приклад виконання команди
[Litov@ vobox-edu ~]$ glite-ce-job-cancel -i jobid
Are you sure you want to remove specified job(s) [y/n]y : y

Для работи в грід необхідно:
- перевірити, яка локальная система керування стоїть на віддаленому обчислювальному ресурсі;
- перевірити, яке програмне забезпечення встановлене і які значення змінних оточення прийняті на віддаленому ресурсі;
- підготувати тестову задачу і перевірити її роботу.

Далі познайомимось з європейським навчальним грід-проектом на прикладі італійського учбового проекту GILDA(Grid Infn Laboratory for Dissemination Activities)

Грід-портали GENIUS/GILDA

Грід-портали забезпечують доступ до ресурсів і прикладних програм Грід через зручний Web-інтерфейс. Фактично, Грід-портал - це середовище, яке дозволяє отримати доступ до Грід-ресурсів за допомогою простого Web браузера, який в наш час встановлено на кожному комп'ютері. Вони розглядаються спільнотою, як найбільш перспективний шлях подальшого використання Грід. Портальний інтерфейс також дуже важливий, так як за допомогою його користувач абсолютно будь-якої підготовки та рівня знань може без проблем почати працювати в Грід.

Початкова сторінка порталу проекту GILDA показана на рис.2.9.6.

Рис. 2.9.6. Початкова сторінка порталу проекту GILDA

- Повнофункціональний Грід-портал GENIUS (Grid Enabled web eNvironment for site Independent User job Submission), розроблений в італійському

інституті INFN (https://genius.ct.infn.it) використовується в рамках проекту GILDA (Grid Infn Laboratory for Dissemination Activities), що є віртуальною лабораторією для демонстрації можливостей технології Grid. Проект GILDA (https://gilda.ct.infn.it/) складається з декількох частин (Рис. 2.9.6.):

- **GILDA Testbed** – набір сайтів з встановленим ПЗ LCG.
- **Grid Demonstrator** – веб-інтерфейс GENIUS, який дозволяє працювати з певним набором додатків.
- **GILDA CA** – центр сертифікації, який видає 14-денні сертифікати для роботи з GILDA.
- **GILDA VO** – віртуальна організація, об'єднуюча всіх користувачів GILDA.
- **Grid Tutor** – веб-інтерфейс GENIUS, що використовується для демонстрації можливостей технології Grid.

Grid Demonstrator є демо-версією, яка дозволяє отримати перший досвід роботи в Грід-середовищі. При цьому від користувача не потрібно ні наявність сертифікату, ні попередня реєстрація на сайті - використовуються зумовлені ім'я / пароль. Правда через демо-режим можна користуватися тільки із заздалегідь підготовленими завданнями і сервісами неповної функціональності.

Можливості GILDA Grid Demonstrator (рис.2.9.7.):

• Перегляд файлів в каталозі демо-користувача .

• Перегляд змінних оточення .

• Навігація в межах каталога віртуальної організації GILDA (перегляд і скачування файлів). .

• Отримання інформації про проксі-сертифікат.

• Посилання на web інтерфейси деяких проектів, встановлених в GILDA: BLAST (біоінформатика) і Sonification (перетворення тексту в графічну або візуальну форму).

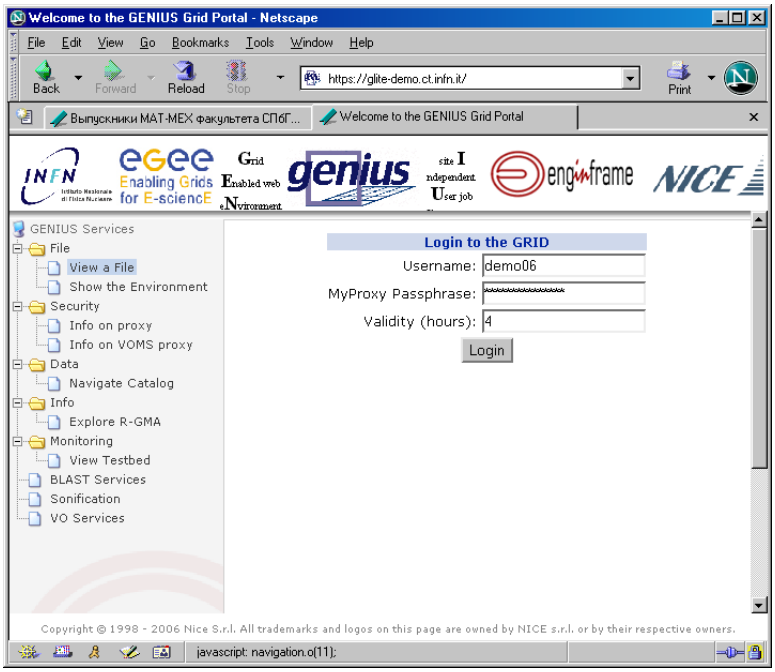


Рис.2.**9.7** Вхід в GILDA Grid Demonstrator

Основний набір додатків і засобів керування заданнями (рис. 2.9.8).

Рис.2.9.8. Відкрите меню VO Services

В пункті меню VO Services (рис.2.9.8.) можна вибрати один з додатків, встановлених в GILDA і, використовуючи Web інтерфейс для відповідного додатку, запустити завдання і отримати результати його виконання. В якості альтернативи можна відкрити пункт меню Job Services і вибрати одне завдання зі списку доступних. Зупинимося детальніше на другому варіанті. Виберіть в лівій панелі меню Job Services-> Job Submission. Справа відкриється список завдань, які можна запустити (рис.2.9.9.).

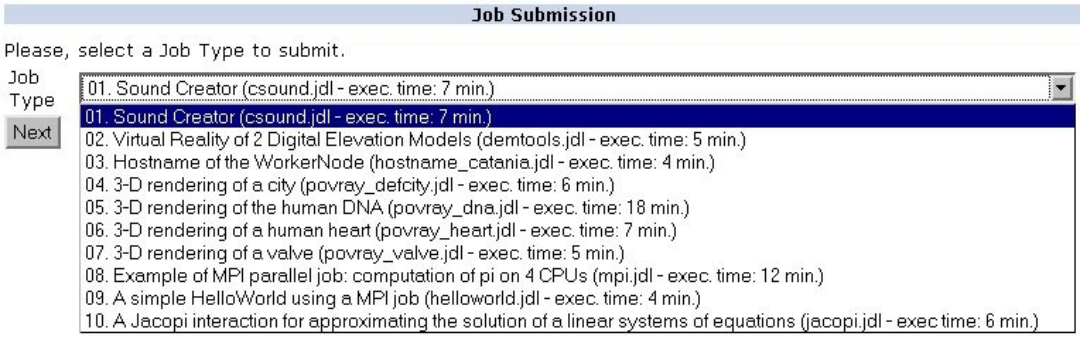


Рис.2.9.9.Список доступних завдань.

Виберіть одне з завдань. Для визначеності хай це буде 06. 3-D rendering of a human heart, результатом виконання якого повинно бути побудова тривимірної моделі людського серця. Після натискання кнопки Next з'явиться наступна сторінка, де можна вибрати обчислювальний елемент, на якому буде виконуватися завдання. Його можна вибрати явно зі списку або дозволити Брокеру завдань самому вирішити це за Вас. Залишимо вибраним другий варіант. (рис.2.9.10.)

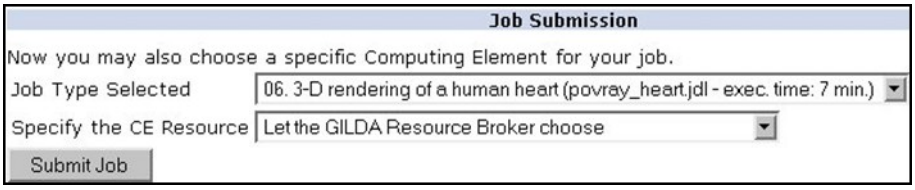


Рис. 2.9.10. Вибір обчислювального елемену для виконання завдання

Після натискання кнопки Submit Job на екрані з'явиться інформація про

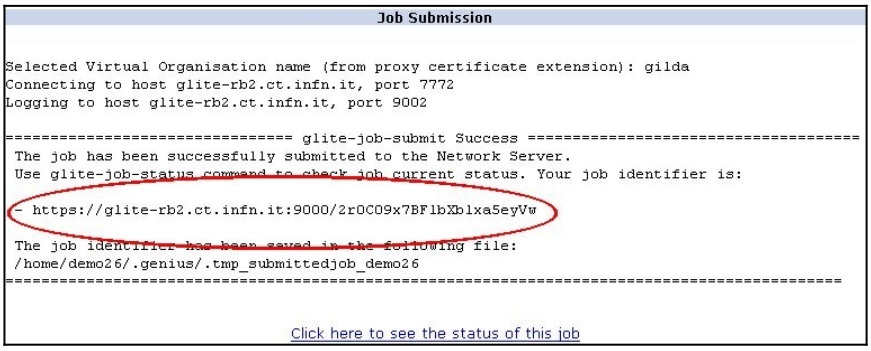


Рис. 2.9.11**.** Завдання створено і послано в Грід

успішне створення завдання і призначення йому ідентифікатора.(рис.2.9.11) За станом виконання завдання можна простежити, натиснувши посилання внизу або вибравши пункт меню Job Queue. (рис.2.9.12)

Рис. 2.9.12**.** Панель відображення статусу виконання завдання**.**

Показані ідентифікатор завдання, час запуску, статус виконання, ресурс, де виконується завдання. Коли завдання буде виконано, в полі Status з'явиться значення Done. Для того, щоб отримати результати виконання завдання, у випадаючому списку Status виберіть GET OUTPUT. Далі дивіться рис. 2.9.13.

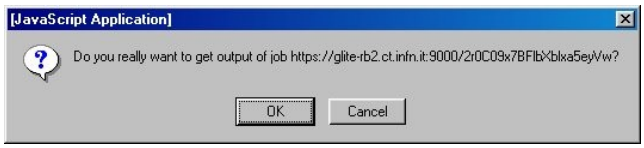


Рис**.** 2.9.13**.** Попередження про отримання результатів**.**

Після попередження з'явиться сторінка, на якій потрібно вибрати ідентифікатор завдання, результати виконання якого необхідно отримати (рис.2.9.14).

Рис. 2.9.14**.** Вибір вихідних файлів.

Після вибору певного завдання з'явиться список файлів, які створені в результаті виконання. В даному випадку: • povray_heart.out - вихідний текстовий файл; • povray_heart.err - повідомлення про помилки та попередження; • heart.png - графічний файл, який представляє тривимірну модель людського серця. Користуючись можливостями браузера можна прямо в браузері ознайомитись з результатами виконання завдання або зберегти вихідні файли на локальному комп'ютері. Таким чином, використовуючи тільки графічний інтерфейс GILDA Demonstrator, можна вибирати завдання для запуску, вибирати обчислювальний елемент, на якому буде виконуватися завдання, контролювати процес виконання, переглядати результати виконання завдання і зберігати вихідні файли на комп'ютері користувача. Інший сервіс проекту GILDA - Grid Tutor дозволяє запускати в GILDA Testbed довільні завдання користувача, але в цьому випадку потрібне отримання сертифікату від GILDA CA і включення в ВО GILDA. Найчастіше Grid Tutor використовується для проведення навчальних курсів EGEE по роботі в Грід. У цьому випадку всі необхідні попередні дії по отриманню сертифікатів, членства в ВО, отримання доступу до UI GILDA Testbed і т.п. виконуються службою підтримки GILDA. Можна використовувати GILDA Grid Tutor і для роботи в повнофункціональній Grid-інфраструктурі GILDA Tested. Інструкції можна знайти на сайті GILDA - https://gilda.ct.infn.it/users.html.

Питання для самоконтролю

1. Що таке координація розрізнених ресурсів?
2. Охарактеризуйте програмне забезпечення Грід.
3. Назвіть особливості програмування Грід.
4. Назвати обчислювальні ресурси Грід.
5. Що таке персональний сертифікат користувача?
6. Що таке Віртуальна організація?
7. Що означає Реєстрація в Віртуальній організації?
8. Дати визначення авторизації і аутентифікації.
9. Що таке Цифровий сертифікат?
10. Що таке конфіденційність, цілісність?
11. Охарактеризуйте особливості алгоритмів планування розміщення завдання на розподілених ресурсах.
12. Що являє собою цифровий підпис?
13. Охарактеризуйте X.509 формат.
14. Хто підписує CA сертифікати?

Література основна

1. Український Центр сертифікації RDIG - http://ca.ugrid.org

Література допоміжна

1. Overview of the Grid Security Infrastructure, http://www.globus.org/security/overview.html

2. EGEE (Enabling Grids for E-sciencE) - http://www.eu-egee.org/

3. gLite 3.0 Generic User Guide - https://edms.cern.ch/file/722398//gLite-3-UserGuide.pdf
4. GILDA (Grid Infn Laboratory for Dissemination Activities) - https://gilda.ct.infn.it/

Розділ 3. Хмарні технології

3.1 Технології віртуалізації та основи хмарних обчислень

Хмарні обчислення - це бізнес-модель, в якій обчислювальні ресурси, такі як обчислення та накопичення, упаковані як сервіси, які схожі на фізичні комунальні послуги.

Хмарні обчислення представляють собою динамічно масштабуємий спосіб доступу до зовнішних обчислювальних ресурсів в вигляді сервісу, який надається посередництвом Інтернету, при цьому користовачу не потрібно ніяких особливих знань про інфраструктуру «хмари» чи навиків управління цією «хмарною» технологією, або Cloud computing – це програмно-апаратне забезпечення, доступне через Інтернет в вигляді сервісу, позволяючого використовувати інтерфейс для віддаленого доступу до виділенх ресурсів (обчислювальних ресурсів , програм і даних). Комп'ютер користовача виступає при цьому рядовим терміналом, підключеним до мережі.

Появу хмарних обчислень обумовили дві ключові тенденції – консолідація і віртуалізація IT-інфраструктури

Консолідація – це об'єднання обчислювальних ресурсів або структур управління в єдиному центрі.

Саме вона здатна суттєво зменшити витрати на IT.

Зазвичай кажуть про консолідацію:

- **серверів** – переміщення децентралізованих додатків, розподілених на різних серверах компанії, в один кластер централізованих гомогенних серверів;
- **систем зберігання** – спільне використання централізованої системи зберігання даних декількома гетерогенними вузлами;
- **додатків** – розміщення декількох додатків на одному хості.

При цьому можна виділити два базових типи консолідації - фізичну і логічну. Фізична консолідація має на увазі географічне переміщення серверів на єдину площадку (в центр даних), а логічна - централізацію управління.

Переміщення комп'ютерів в єдиний центр обробки даних дозволяє забезпечити комфортні умови для обладнання та технічного персоналу, а також збільшити ступінь фізичного захисту серверів. Крім того, в центрі обробки даних можна використовувати більш продуктивне і високоякісне обладнання, яке економічно недоцільно встановлювати в рядових підрозділах.

Очевидна перевага цього рішення в тому, що спрощується робота з розгортання та керування системами, знижується ступінь дублювання.

Спрощується забезпечення фізичного захисту та поліпшується мережева безпека, оскільки сервери опиняються під захистом єдиного, централізовано керованого центру.

Логічний тип консолідації має на увазі перебудову системи управління IT-інфраструктури. Це необхідно як для збільшення масштабованості і керованості складної розподіленої обчислювальної системи, так і для об'єднання сегментів корпоративної мережі. Логічна консолідація додатків призводить до централізації управління критичними для бізнесу системами і додатками. Тут переваги очевидні: в першу чергу це вивільнення апаратних ресурсів, які можна використовувати на інших ділянках інформаційної системи. По-друге, більш проста і логічна структура управління IT-інфраструктурою робить її більш гнучкою і пристосованою для майбутніх змін.

Сценарій гомогенної консолідації передбачає перенесення одного масштабованого додатку, який раніше виконувався на декількох серверах, на один, більш потужний. Об'єднання даних і додатків на одному сервері помітно прискорює процеси обробки і пошуку, а також підвищує рівень цілісності.

Гетерогенна консолідація за змістом схожа з гомогенною, але в цьому випадку об'єднанню підлягають різні додатки. Наприклад, кілька примірників Exchange Server і SQL Server, раніше запускалися на окремих комп'ютерах, можуть бути зведені на єдиній машині. Переваги гетерогенної консолідації – зростаюча масштабованість сервісів і більш повне задіяння системних ресурсів.

Як відзначають фахівці з хмарних технологій - консолідація IT-інфраструктури стала першим кроком до "хмари".

Віртуалізація – це маскування від користувача складності апаратної та програмної реалізації системи та її складових, географічних відстаней між вузлами, належності вузлів різним організаціям, створення ілюзії роботи з реальним суперкомп’ютером. По статистиці средній рівень завантаження процессорных потужностей у серверів під управлінням Windows не превершує 10%, у Unix-систем цей показник кращий, але і там в средньому не превершує 20%. Низька ефективність використання серверів пояснюється широким використанням з початку 90-х років підходом “один додаток — один сервер”, т. ч. щоразу для розгортання нового додатку використовували новий сервер. Очевидно, що на практиці це призводить до швидкого збільшення серверного парку і як наслідок — до зростанню затрат на його адміністрування, енергоспоживання та охолодження, а також потреби в збільшенні приміщень для установки нових серверів та отриманні ліцензій на серверну ОС.

В основі віртуалізації лежить можливість одного комп'ютера виконувати роботу декількох комп'ютерів завдяки розподілу його ресурсів на декілька середовищ. За допомогою віртуальних серверів і віртуальних настільних комп'ютерів можна розмістити кілька ОС і кілька додатків в єдиному розташування. Таким чином, фізичні та географічні обмеження перестають мати якесь значення.

У комп'ютерних технологіях під терміном «віртуалізація» зазвичай розуміється абстракція обчислювальних ресурсів і надання користувачеві системи, яка «інкапсулює» (приховує в собі) власну реалізацію. Простіше кажучи, користувач працює зі зручним для себе представленням об'єкту і для

нього не має значення, як об'єкт влаштований в дійсності. Можливість запуску декількох віртуальних машин на одній фізичній викликає інтерес серед комп'ютерних фахівців не тільки тому, що це підвищує гнучкість IT – інфраструктури, але й тому, що це дає значну економічну вигоду.

Переваги технологій віртуалізації

1. Ефективне використання обчислювальних ресурсів.
2. Скорочення витрат на інфраструктуру. Віртуалізація дозволяє скоротити кількість серверів і пов'язаного з ними IT -обладнання в інформаційному центрі.
3. Зниження витрат на програмне забезпечення. Деякі виробники програмного забезпечення ввели окремі схеми ліцензування спеціально для віртуальних середовищ.
4. Підвищення гнучкості і швидкості реагування системи. Віртуалізація пропонує новий метод управління IT – інфраструктурою і допомагає IT – адміністраторам витрачати менше часу на виконання повторюваних завдань – наприклад, на ініціацію, налаштування, відстеження і технічне обслуговування.
5. **Несумісні додатки можуть працювати на одному комп'ютері**. При використанні віртуалізації на одному сервері можлива установка Linux і Windows серверів, шлюзів, баз даних і інших абсолютно несумісних в рамках однієї невіртуалізованої системи додатків.
6. Підвищення доступності додатків і забезпечення безперервності роботи підприємства. Завдяки надійній системі резервного копіювання та міграції віртуальних середовищ цілком без перерв в обслуговуванні Ви зможете скоротити періоди планового простою і забезпечити швидке відновлення системи в критичних ситуаціях. "Падіння» одного віртуального сервера не веде до втрати інших віртуальних серверів.
7. Можливості легкої архівації. Оскільки жорсткий диск віртуальної машини зазвичай представляється у вигляді файлу певного формату, розташованого на якому-небудь фізичному носії, віртуалізація дає можливість простого копіювання цього файлу на резервний носій як засіб архівування і резервного копіювання всієї віртуальної машини.
8. Підвищення керованості інфраструктури: використання централізованого управління віртуальною інфраструктурою дозволяє скоротити час на адміністрування серверів, забезпечує балансування навантаження і "живу" міграцію віртуальних машин.

Віртуальна машина – це повністю ізольований програмний контейнер, який працює з власною ОС і додатками, подібно фізичному комп'ютеру. Віртуальна машина діє так само, як фізичний комп'ютер, і містить власні віртуальні (тобто програмні) оперативну пам’ять, жорсткий диск і мережевий адаптер. ОС не може розрізнити віртуальну і фізичну машини. Те ж саме можна сказати про додатки та інших комп'ютерах в мережі. Але віртуальні машини складаються виключно з програмних компонентів і не включають обладнання. Це дає їм низку унікальних переваг.

Основні різновиди віртуалізації

Розглянемо основні різновиди віртуалізації, такі як:
• віртуалізація серверів (повна віртуалізація і паравіртуалізація)
• віртуалізація на рівні операційних систем
• віртуалізація додатків,
• віртуалізація уявлень.

Віртуалізація серверів

Віртуалізація серверів має на увазі запуск на одному фізичному сервері декількох віртуальних серверів. Віртуальні машини або сервери являють собою програми, запущені на хостовій операційній системі, які емулюють фізичні пристрої серверу. На кожній віртуальній машині може бути встановлена операційна система, на яку можуть бути встановлені додатки і служби. Типові представники - це продукти VmWare (ESX, Server, Workstation) і Microsoft (Hyper -V, Virtual Serer, Virtual PC).

Повна віртуалізація (Full, Native Virtualization). Використовуються немодифіковані екземпляри гостьових операційних систем, а для підтримки роботи цих ОС служить загальний шар емуляції їх виконання поверх хостової ОС, в ролі якої виступає звичайна операційна система. Така технологія застосовується, зокрема, в VMware Workstation, VMware Server (колишній GSX Server), Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server, Virtual Iron. До переваг даного підходу можна вважати відносну простоту реалізації, універсальність і надійність рішення; всі функції управління бере на себе хост-ОС. Недоліки – високі додаткові накладні витрати на використовувані апаратні ресурси, відсутність обліку особливостей гостьових ОС, менша, ніж потрібно гнучкість у використанні апаратних засобів.

Паравіртуалізації (paravirtualization). Модифікація ядра гостьової ОС виконується таким чином, що в неї включається новий набір API, через який вона може безпосередньо працювати з апаратурою, не конфліктуючи з іншими віртуальними машинами. При цьому немає необхідності задіяти повноцінну ОС в якості хостового ПЗ, функції якого в даному випадку виконує спеціальна система, що отримала назву гіпервізора (hypervisor). Саме цей варіант є сьогодні найбільш актуальним напрямком розвитку серверних технологій віртуалізації і застосовується в VMware ESX Server, Xen (і рішеннях інших постачальників на базі цієї технології), Microsoft Hyper -V. Переваги даної технології полягають у відсутності потреби в хостової ОС – ВМ встановлюються фактично на "голе залізо".

Віртуалізація на рівні ядра ОС (operating system – level virtualization). Цей варіант передбачає використання одного ядра хостової ОС для створення незалежних паралельно працюючих операційних середовищ. Для гостьового ПЗ створюється тільки власне мережеве та апаратне оточення. Такий варіант використовується в Virtuozzo (для Linux і Windows), OpenVZ (безкоштовний

варіант Virtuozzo) і Solaris Containers. Переваги – висока ефективність використання апаратних ресурсів, низькі накладні технічні витрати, відмінна керованість, мінімізація витрат на придбання ліцензій. Недоліки – реалізація тільки однорідних обчислювальних середовищ.

Віртуалізація додатків має на увазі застосування моделі сильної ізоляції прикладних програм з керованою взаємодією з ОС, при якій віртуалізується кожен екземпляр додатків, всі його основні компоненти: файли (включаючи системні), реєстр, шрифти, INI – файли, COM – об'єкти, служби. Додаток виповнюється без процедури інсталяції в традиційному її розумінні і може запускатися прямо з зовнішніх носіїв (наприклад, з флеш – карт або з мережевих папок). З точки зору IT – відділу, такий підхід має очевидні переваги: прискорення розгортання настільних систем і можливість управління ними, зведення до мінімуму не тільки конфліктів між додатками, а й потреби у тестуванні додатків на сумісність. Дана технологія дозволяє використовувати на одному комп'ютері, а точніше в одній і тій же операційній системі кілька несумісних між собою додатків одночасно.

Віртуалізація уявлень (робочих місць). Віртуалізація уявлень має на увазі емуляцію інтерфейсу користувача. Тобто користувач бачить додаток і працює з ним на своєму терміналі, хоча насправді додаток виконується на віддаленому сервері, а користувачеві передається лише зображення віддаленої програми. Залежно від режиму роботи користувач може побачити віддалений робочий стіл і запущений на ньому додаток, або тільки саме вікно програми.

Слід вказати, що історія розвитку технологій віртуалізації налічує 50 років. Компанія IBM була першою, хто задумався про створення віртуальних середовищ для різних користувальницьких завдань (тоді ще в мейнфреймах). У 60-х роках минулого століття віртуалізація представляла чисто науковий інтерес і була оригінальним рішенням для ізоляції комп'ютерних систем в рамках одного фізичного комп'ютера. Після появи персональних комп'ютерів інтерес до віртуалізації дещо послабився незважаючи бурхливий розвиткок операційних систем, які пред'являли адекватні вимоги до апаратного забезпечення того часу. Однак бурхливе зростання апаратних потужностей комп'ютерів наприкінці дев'яностих років минулого століття змусив IT-спільноту знову згадати про технології віртуалізації програмних платформ.

У 1999 р компанія VMware представила технологію віртуалізації систем на базі x86 в якості ефективного засобу, здатного перетворити системи на базі x86 в єдину апаратну інфраструктуру загального користування та призначення, що забезпечує повну ізоляцію, мобільність і широкий вибір ОС для прикладних середовищ. Компанія VMware була однією з перших, хто зробив серйозну ставку виключно на віртуалізацію. Як показав час, це виявилося абсолютно виправданим. Сьогодні WMware пропонує комплексну віртуалізаційну платформу четвертого покоління VMware vSphere 4, яка включає ресурси як для окремого ПК, так і для центру обробки даних. Ключовим компонентом цього програмного комплексу є гіпервізор VMware ESXServer. Пізніше в "битву" за місце в цьому модному напрямку розвитку інформаційних технологій включилися такі компанії як Parallels (раніше SWsoft), Oracle (Sun Microsystems), Citrix Systems (XenSourse).

Корпорація Microsoft вийшла на ринок засобів віртуалізації в 2003 р з придбанням компанії Connectix, випустивши свій перший продукт Virtual PC для настільних ПК. З тих пір вона послідовно нарощувала спектр пропозицій у цій галузі і на сьогодні майже завершила формування віртуалізаційних платформ , до складу якої входять такі рішення як Windows 2008Server R2 з компонентом Hyper-V, Server 2012 Hyper-V, Microsoft Application Virtualization (App-v), Microsoft Virtual Desktop Infrastructure (VDI),Remote Desktop Services, System Center Virtual MachineManager .

На сьогоднішній день постачальники технологій віртуалізації пропонують надійні і легкокеровані платформи, а ринок цих технологій переживає справжній бум. За оцінками провідних експертів, зараз віртуалізація входить до трійки найбільш перспективних комп'ютерних технологій.

Короткий огляд платформ віртуалізації.

VMware

Компанія VMware - одна з перших гравців на ринку платформ віртуалізації. В1998 році VMware запатентувала свої програмні техніки віртуалізації і з тих пір випустила чимало ефективних і професійних продуктів для віртуалізації різного рівня: від VMware Workstation, призначеного для настільних ПК, доVMware ESX Server, що дозволяє консолідувати фізичні сервери підприємства в віртуальній інфраструктурі. За результатами різних тестів продуктивності засоби віртуалізації VMware майже завжди за більшістю параметрів виграють у конкурентів. VMware має більше 100 000 клієнтів по всьому світу. Мережа партнерств охоплює понад 350 виробників обладнання та програмного забезпечення і понад 6000 реселерів.

Платформи віртуалізації VMware:

VMware Workstation - платформа, орієнтована на desktop-користувачів і призначена для використання розробниками ПЗ, а також професіоналами в сфері IT. Нова версія популярного продукту VMware Workstation 7 стала доступна в 2009 р, в якості хостових операційних систем підтримуються Windows і Linux. VMware Workstation 7 може використовуватися спільно з середовищем розробки, що робить її особливо популярною в середовищі розробників, користувачів і фахівців технічної підтримки. Вихід VMware Workstation 7 означав офіційну підтримку Windows 7 як в якості гостьової, так і хостової операційної системи. Продукт включає підтримку Aero Peek і Flip 3D, що робить можливим спостерігати за роботою віртуальної машини, наближаючи курсор до панелі завдань VMware, або до відповідної вкладки на робочому столі хоста. Версія може працювати на будь-якій версії Windows 7, а також як і будь-які версії Windows можуть бути запущені в віртуальних машинах. Крім того, віртуальні машини в VMware Workstation 7 повністю підтримують Windows Display Driver Model (WDDM), що дозволяє використовувати інтерфейс Windows Aero в гостьових машинах.

VMware Player - безкоштовний «програвач» віртуальних машин на основі віртуальної машини VMware Workstation, призначений для запуску вже готових зразків віртуальних машин, створених в інших продуктах VMware, а також в

Microsoft VirtualPC і Symantec LiveState Recovery. Починаючи з версії 3.0 VMware Player дозволяє також створювати зразки віртуальних машин. Обмеження функціональності стосується в основному функцій, призначених для IT-фахівців і розробників ПЗ.

VMware Fusion - настільний продукт для віртуалізації на платформі Mac від компанії Apple.

VMware Server, Безкоштовний продукт VMware Server є досить потужною платформою віртуалізації, яка може бути запущена на серверах під управлінням хостових операційних систем Windows і Linux. Основне призначення VMware Server - підтримка малих і середніх віртуальних інфраструктур невеликих підприємств. У зв'язку з невеликою складністю його освоєння і установки, VMware Server може бути розгорнутий в найкоротші терміни як на серверах організацій, так і на комп'ютерах домашніх користувачів.

VMware Ace - продукт для створення захищених політиками безпеки віртуальних машин, які потім можна поширювати по моделі SaaS (Software-as-aService).

VMware vSphere - комплекс продуктів, що представляє надійну платформу для віртуалізації ЦОД. Компанія позиціонує даний комплекс також як потужну платформу віртуалізації для створення і розгортання приватної «хмари». VMware vSphere поставляється в декількох випусках з можливостями, призначеними спеціально для малих підприємств і середніх компаній і корпорацій. VMware vSphere включає ряд компонентів, що перетворюють стандартне обладнання в загальне стійке середовище, яке нагадує мейнфрейм і включає вбудовані елементи управління рівнями обслуговування для всіх додатків:

• Служби інфраструктури - це компоненти, які забезпечують всебічну віртуалізацію ресурсів серверів, сховищ і мереж, їх об'єднання та точне виділення додатків на вимогу і відповідно до пріоритетів бізнесу.

• Служби додатків - це компоненти, що надають вбудовані елементи управління рівнями обслуговування для всіх додатків на платформі платформи vSphere незалежно від їх типу чи ОС.

• **VMware vCenter Server** надає центральну консоль для управління віртуалізацією, що забезпечує адміністрування служб інфраструктури і додатків. Ця консоль підтримує всебічну візуалізацію всіх аспектів віртуальної інфраструктури, автоматизацію повсякденної експлуатації і масштабованість для керування великими середовищами ЦОД (рис.3.1.1).

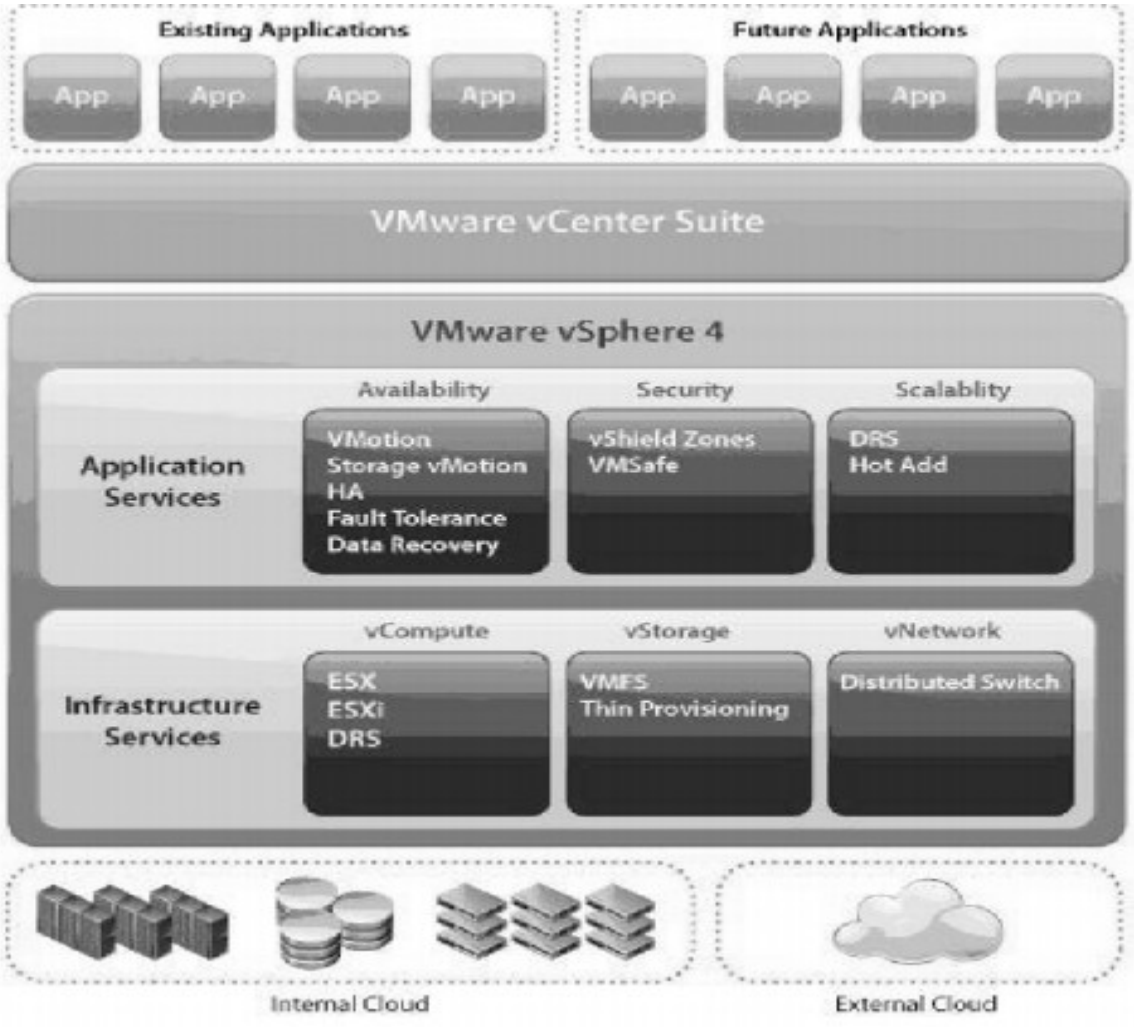


Рис.3.1.1 Структура платформи **vSphere**

VMware ESX Server - це гипервизор, який розбиває фізичні сервери на безліч віртуальних машин. VMware ESX є основою пакету VMware vSphere і входить в усі випуски VMware vSphere (рис.3.1.2)

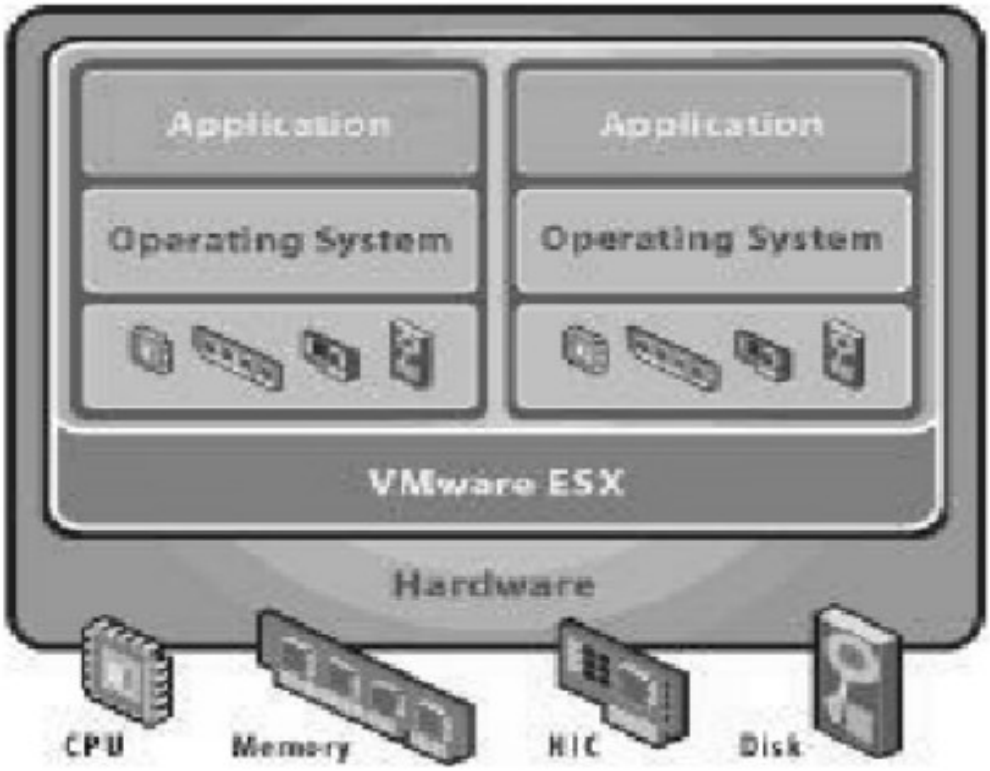


Рис.3.1.2 Гіпервізор VMware ESX

VMware vSphere Hypervisor (раніше VMware ESXi) - "полегшена" платформа віртуалізації корпоративного рівня, заснована на технологіях ESX. Продукт є безкоштовним і доступний для завантаження з сайту VMware. VSphere VMwareHypervisor є найпростішим способом для початку роботи з віртуалізацією.

VMware vCenter- надає розширюєму і масштабовану платформу для опереджувального управління віртуальною інфраструктурою і забезпечує отримання про неї всеосяжної інформації. VMware vCenter Server забезпечує централізоване управління середовищами vSphere і спрощує виконання повсякденних завдань, значно покращуючи адміністративне управління середовищем. Продукт володіє широкими можливостями по консолідації серверів, їх налаштування і управління. VMware vCenter Server агрегує в собі всі аспекти управління віртуальним середовищем: від віртуальних машин до збору інформації про фізичні сервери для подальшої їх міграції в віртуальну інфраструктуру. Крім центрального продукту управління віртуальною інфраструктурою vCenter Server існує також ряд доповнень, що реалізують різні аспекти планування, управління та інтеграції розподіленої віртуальної інфраструктури (VMware vCenter Server Heartbeat, VMware vCenter Orchestrator, VMware vCenter Capacity IQ, VMware vCenter Site Recovery Manager, VMware vCenter Lab Manager, VMware vCenter Configuration Manager, VMware vCenter Converter). Зокрема, vCenter Converter призначений для перекладу в віртуальне середовище фізичних серверів, що дозволяє здійснювати «гарячу» (без зупину систем) і «холодну» міграцію.

vCenter Site Recovery Manager - це ПЗ для створення територіально-віддаленого резервного сегменту віртуальної інфраструктури, який в разі відмови основного вузла, бере на себе функції по запуску віртуальних машин відповідно до плану відновлення після збоїв. **vCenter Lab Manager** - продукт для створення інфраструктури зберігання і доставки конфігурацій віртуальних машин, що дозволяє організувати ефективну схему тестування в компаніях-розробників ПЗ.

VMware ThinApp - колишній продукт Thinstall Virtualization Suite, ПЗ для віртуалізації додатків, що дозволяє поширювати встановлені додатки на клієнтські робочі станції, скорочуючи час на стандартні операції по встановленню і конфігурації.

VMware View - комплекс продуктів, який забезпечує централізацію призначених для користувача робочих станцій в віртуальних машинах на платформі vSphere. Це дозволяє скоротити витрати на стандартні IT-операції, пов'язані з розгортанням і обслуговуванням призначених для користувача десктопів.

VMware Capacity Planner - засіб централізованого збору та аналізу даних про апаратне і програмне забезпечення серверів, а також продуктивності обладнання. Ці дані використовуються авторизованими партнерами VMware для побудови планів консолідації віртуальних машин на платформі VMware ESX Server.

VMware VMmark - продукт, доступний тільки виробникам апаратного забезпечення, призначений для тестування продуктивності VMware ESX Server на серверних платформах.

Citrix (Xen)

Розробка некомерційного гипервизора Xen починалася як дослідницький проект комп'ютерної лабораторії Кембриджського університету. Засновником проекту і його лідером був Ian Пратт (Ian Pratt) співробітник університету, який створив згодом компанію XenSource, яка займається розробкою комерційних платформ віртуалізації на основі гіпервізора Xen, а також підтримкою Open Source співтовариства некомерційного продукту Xen. Спочатку Xen був найрозвинутішою платформою, яка підтримує технологію паравіртуалізації. Безкоштовна версія Xen включена в дистрибутиви декількох ОС, таких як Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. В середині серпня 2007 року компанія XenSource була поглинена компанією Citrix Systems. Сума проведеної операції близько 500 мільйонів доларів говорить про серйозні наміри Citrix щодо віртуалізації.

Безкоштовний Xen. В даний час Open Source версія платформи Xen застосовується в основному в освітніх і дослідницьких цілях. Деякі вдалі ідеї, реалізовані численними розробниками з усього світу, находять своє відображення в комерційних версіях продуктів віртуалізації компанії Citrix. Зараз безкоштовні версії Xen включаються в дистрибутиви багатьох Linux-систем, що дозволяє їх користувачам застосовувати віртуальні машини для ізоляції програмного забезпечення в гостьових ОС з метою його тестування і вивчення проблем безпеки, без необхідності установки платформи віртуалізації. До того ж багато незалежних розробники ПЗ можуть поширювати його за допомогою віртуальних шаблонів, в яких вже встановлена і налаштована гостьова система і пропонований продукт. Крім того, Xen ідеально підходить для підтримки старого програмного забезпечення у віртуальній машині. Для більш серйозних цілей у виробничому середовищі необхідно використовувати комерційні платформи компанії Citrix.

Citrix XenApp - призначений для віртуалізації і публікації додатків з метою оптимізації інфраструктури доставки сервісів у великих компаніях. XenApp має величезну кількість користувачів по всьому світу і в багатьох компаніях являється ключовим компонентом IT-інфраструктури.

Citrix XenServer - платформа для консолідації серверів підприємств середнього масштабу, яка включає основні можливості для підтримки віртуальної інфраструктури. Виробник позиціонує даний продукт як рішення Enterprise-рівня для віртуалізації серверів, яке підтримує роботу в «хмарному» оточенні.

Citrix XenDesktop - рішення по віртуалізації десктопів підприємства, яке дозволяє централізовано зберігати і доставляти робочі оточення в віртуальних машинах користувачам. Продукт підтримує кілька сценаріїв доставки додатків на настільні ПК, тонкі клієнти і мобільні ПК і сумісний з серверними віртуалізаційними рішеннями конкурентів.

Microsoft

Для Microsoft все почалося, коли в 2003 році вона придбала компанію Connectix, одну з небагатьох компаній, виробляючу програмне забезпечення для віртуалізації під Windows. Разом з Connectix, компанії Microsoft дістався продукт Virtual PC, який конкурував тоді з розробками компанії VMware щодо настільних систем віртуалізації. За великим рахунком, Virtual PC надавав тоді таку кількість функцій, як і VMware Workstation, і при належній увазі міг би бути в даний час повноцінним конкурентом цієї платформи. Однак з того часу компанія Microsoft випускала по мінорному релізу в рік, не приділяючи особливої уваги продукту Virtual PC, в той час як VMware стрімко розвивала свою систему віртуалізації, перетворивши її по-справжньому в професійний інструмент. Усвідомивши своє технологічне відставання в сфері віртуалізації серверних платформ, компанія Microsoft випустила продукт Virtual Server 2005, націлений на створення і консолідацію віртуальних серверів організацій. Однак було вже пізно. Компанія VMware вже захопила лідерство в цьому сегменті ринку, пропонуючи в той момент дві серверних платформи віртуалізації VMware GSX Server і VMware ESX Server, кожна з яких за багатьма параметрами перевершувала платформу Microsoft. Остаточний удар був нанесений в 2006 році, коли VMware фактично оголосила продукт VMware GSX Server безкоштовним, взявшись за розробку продукту VMware Server на його основі і сконцентрувавши всі зусилля на продажах потужної корпоративної платформи VMware ESX Server в складі віртуальної інфраструктури Virtual Infrastructure 3. У компанії Microsoft був тільки єдиний вихід у цій ситуації: в квітні 2006 року вона також оголосила про безкоштовність продукту Microsoft Virtual Server 2005. Також існуючі раніше два видання Standard Edition і Enterprise Edition були об'єднані в одне - Microsoft Virtual Server Enterprise Edition. З тих пір Microsoft істотно змінила стратегію щодо віртуалізації і влітку 2008 року був випущений фінальний реліз платформи віртуалізації Microsoft Hyper-V, інтегрованої в ОС Windows Server 2008. Тепер роль сервера віртуалізації доступна всім користувачам нової серверної операційної системи Microsoft. (свою гіпервізорну технологію Microsoft назвала Hyper-V)

Microsoft Virtual Server. Серверна платформа віртуалізації Microsoft Virtual Server може використовуватися на сервері під керуванням операційної системи Windows Server 2003 і призначена для одночасного запуску декількох віртуальних машин на одному фізичному хості. Платформа безкоштовна і надає тільки базові функції.

Microsoft Virtual PC. Продукт Virtual PC був куплений корпорацією Microsoft разом з компанією Connectix і вперше під маркою Microsoft був випущений як Microsoft Virtual PC 2004. Купуючи Virtual PC і компанію Connectix, компанія Microsoft будувала далекосяжні плани щодо забезпечення користувачів інструментом для полегшення міграції на наступну версію операційної системи Windows. Тепер Virtual PC 2007 безкоштовний і доступний для підтримки настільних ОС у віртуальних машинах.

Microsoft Hyper-V. Продукт Microsoft позиціонується як основний конкурент VMware ESX Server в області корпоративних платформ віртуалізації.

Microsoft HyperV є рішення для віртуалізації серверів на базі процесорів з архітектурою x64 в корпоративних середовищах. На відміну від продуктів Microsoft Virtual Server або Virtual PC, Hyper-V забезпечує віртуалізацію на апаратному рівні, з використанням технологій віртуалізації, вбудованих в процесори. Hyper-V забезпечує високу продуктивність, практично рівну продуктивності однієї операційної системи, яка працює на виділеному сервері. Hyper-V поширюється двома способами: як частина Windows Server 2008, або в складі незалежного безкоштовного продукту Microsoft Hyper-V Server.

У Windows Server 2008 технологія Hyper-V може бути розгорнута як в повній установці, так і в режимі Server Core. Hyper-V Server працює тільки в режимі Core. Це дозволяє в повній мірі реалізувати всі переваги «тонкої», економічної і керованої платформи віртуалізації.

Hyper-V є вбудованим компонентом 64-розрядних версій Windows Server 2008 Standard, Windows Server 2008 Enterprise і Windows Server 2008 Datacenter. Ця технологія недоступна в 32-розрядних версіях Windows Server 2008, в Windows Server 2008 Standard без Hyper-V, Windows Server 2008 Enterprise без Hyper-V, Windows Server 2008 Datacenter без Hyper-V, в Windows Web Server 2008 і Windows Server 2008 для систем на базі Itanium.

Розглянемо коротко особливості архітектури Hyper-v. Hyper-v є гіпервізор, тобто прошарок між обладнанням і віртуальними машинами рівнем нижче операційної системи. Ця архітектура була спочатку розроблена IBM в 1960-і роки для мейнфреймів і порівняно недавно стала доступною на платформах x86 / x64, як частина ряду рішень, включаючи Windows Server 2008 Hyper-V, Windows Server 2012 Hyper-V і Vmware ESX.

Віртуалізація на базі гіпервізора заснована на тому, що між обладнанням і віртуальними машинами з'являється прошарок, що перехоплює звернення операційних систем до процесора, пам'яті та інших пристроїв. При цьому доступ до периферійних пристроїв в різних реалізаціях гіпервізора може бути організований по різному. З точки зору існуючих рішень для реалізації менеджера віртуальних машин можна виділити два основних види архітектури гіпервізора: **мікроядерну і монолітну.**

Монолітний підхід розміщує гіпервізор в єдиному рівні, який також включає більшість необхідних компонентів, таких як ядро, драйвери пристроїв і стек введення / виведення. Це підхід використовується такими рішеннями, як VMware ESX і традиційними системами мейнфреймів.

Монолітний підхід має на увазі, що всі драйвери пристроїв поміщені в гіпервізор. У монолітної моделі гіпервізор для доступу до обладнання використовує власні драйвери. Гостьові ОС працюють на віртуальних машинах поверх гіпервізора. Коли гостьовій системі потрібен доступ до обладнання, вона повинна пройти через гіпервізор і його модель драйверів. Зазвичай одна з гостьових ОС грає роль адміністратора або консолі, в якій запускаються компоненти для надання ресурсів, управління і моніторингу всіх гостьових ОС, які працюють на сервері.

Модель монолітного гіпервізора забезпечує прекрасну продуктивність,але має ряд недоліків, таких як:

• Стійкість - якщо в оновлену версію драйвера вкралася помилка, то збої почнуться у всій системі, у всіх її віртуальних машинах.
• Проблеми оновлення драйверів - при необхідності поновлення драйверу будь-якого пристрою (наприклад мережевого адаптеру) оновити драйвер можливо тільки разом з виходом нової версії гіпервізору, в яку буде інтегрований новий драйвер для даного пристрою.
• Труднощі з використанням непідтримуваного обладнання. Наприклад, Ви зібралися використовувати обладнання «Сервер» досить потужний і надійний, але при цьому в гіпервізорі не виявилося потрібного драйвера для RAID-контролера або мережевого адаптера. Це унеможливить використання відповідного обладнання, а, значить, і серверу.

В мікроядерній реалізації можна говорити про «тонкий гіпервізор», в цьому випадку в ньому зовсім немає драйверів. Замість цього драйвери працюють в кожному індивідуальному розділі, щоб будь-яка гостьова ОС мала можливість отримати через гіпервизор доступ до обладнання. При цьому кожна віртуальна машина займає абсолютно окремий розділ, що позитивно позначається на захищеності і надійності. У мікроядерній моделі гіпервізора (в віртуалізації Windows Server 2008 R2 використовується саме вона) один розділ є кореневим, або батьківським (Parent), решта - дочірніми (child). Розділ - це найменша ізольована одиниця, підтримувана гіпервізором. Розмір гіпервизора Hyper-V менше 1,5 Мб.

Кожному розділу призначаються конкретні апаратні ресурси – частка процесорного часу, обсяг пам'яті, пристрої та ін. Батьківський розділ створює дочірні розділи і керує ними, а також містить стек віртуалізації (virtualization stack), який використовується для управління дочірніми розділами. Батьківський розділу створюється першим і володіє всіма ресурсами, які не належать Гіпервізору. Володіння всіма апаратними ресурсами означає, що саме кореневий (тобто, батьківський) розділ керує живленням, підключенням самоналагоджувальних пристроїв, відає питаннями апаратних збоїв і навіть керує завантаженням гіпервізора (рис.3.1.3).

У коренеому розділі міститься стек віртуалізації - набір програмних компонентів, розташованих поверх гипервизора, які спільно з ним забезпечують роботу віртуальних машин. Стек віртуалізації обмінюється даними з гіпервізором і виконує всі функції по віртуалізації, які не підтримуються безпосередньо гіпервізором. Велика частина цих функцій пов'язана зі створенням дочірніх розділів і управлінням ними і необхідними їм ресурсами (ЦП, пам'ять, пристрої).

Перевага мікроядерного підходу, застосованого в Windows Server 2008 R2, в порівнянні з монолітним підходом полягає в тому, що драйвери, які повинні розташовуватися між кореневим розділом і фізичним сервером, не вимагають внесення жодних змін до модель драйверів. Іншими словами, в системі можна просто застосовувати існуючі драйвери. У Microsoft цей підхід обрали, оскільки необхідність розробки нових драйверів сильно загальмувала б розвиток системи. Що ж стосується гостьових ОС, вони будуть працювати з емуляторами або синтетичними пристроями.

З іншого боку, мікроядерна модель може дещо програвати монолітній моделі в продуктивності. Однак в наші дні головним пріоритетом стала безпеку, тому для більшості компаній цілком прийнятна втрата кількох відсотків в продуктивності заради скорочення фронту нападу і підвищення стійкості. .

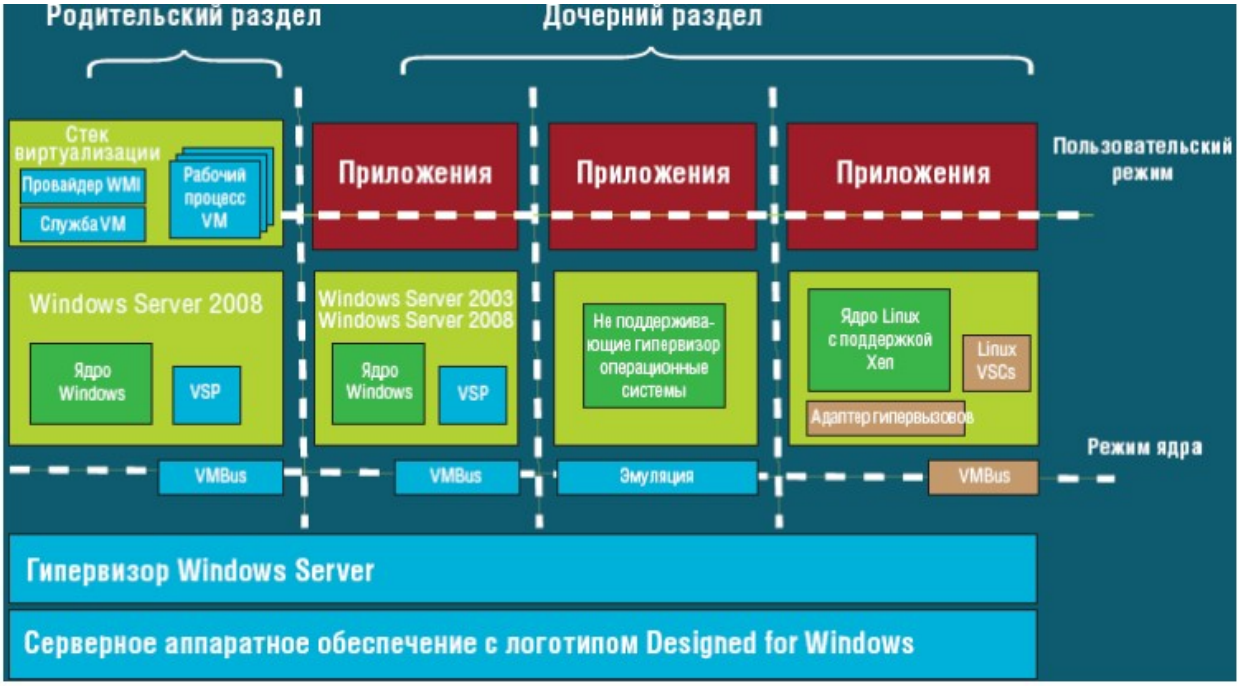


Рис.3.1.3. *Архітектура Hyper-v*

Усі версії Hyper-V мають один кореневий розділ. Цей розділ керує функціями Hyper-V. З батьківського розділу запускається консоль Windows Server Virtualization. Крім того, батьківський розділ використовується для запуску віртуальних машин (VM), що підтримують потокову емуляцію старих апаратних засобів. Такі VM, побудовані на готових шаблонах, емулюючих апаратні засоби, є аналогами VM, працюючими в продуктах з віртуалізацією на базі хоста, наприклад Virtual Server.

Гостьові VM запускаються з дочірніх розділів Hyper-V. Дочірні розділи підтримують два типи VM: високопродуктивні VM на основі архітектури VMBus і VM, керовані системою-хостом. До першої групи входять VM з системами Windows Server 2003, Windows Vista, Server 2008 і Linux (підтримують Xen).

Нову архітектуру VMBus відрізняє високопродуктивний конвеєр, функціонуючий в оперативній пам'яті, з'єднуючий клієнтів Virtualization Service Clients (VSC) на гостьових VM з провайдером Virtual Service Provider (VSP) хоста. VM, керовані хостом, запускають платформи, які не підтримують нову архітектуру VMBus: Windows NT, Windows 2000 і Linux (без підтримки технології Xen, наприклад SUSE Linux Server Enterprise 10).

Microsoft System Center Virtual Machine Manager (SCVMM) – окремий продукт сімейства System Center для управління віртуальною інфраструктурою, ефективного використанням ресурсів фізичних вузлів, а також спрощення підготовки та створення нових гостьових систем для адміністраторів і

користувачів. Продукт забезпечує всебічну підтримку консолідації фізичних серверів в віртуальній інфраструктурі, швидке і надійне перетворення фізичних машин в віртуальні, розумне розміщення віртуальних навантажень на підходящих фізичними вузлах, а також єдину консоль для управління ресурсами і їх оптимізації.

SCVMM забезпечує наступні можливості:

• Централізоване управління серверами віртуальних машин в масштабах організації. SCVMM підтримує управління серверами Microsoft Hyper-V, Microsoft Virtual Server, VMware ESX.

• Створення бібліотеки шаблонів віртуальних машин. Шаблони віртуальних машин є наборами образів встановлених операційних систем, які можуть бути розгорнуті за лічені хвилини.

• Моніторинг та розміщення віртуальних машин у відповідність із завантаженістю фізичних серверів.

• Міграція (конвертація) фізичних серверів у віртуальні машини -технологія P2V. Технологія P2V дозволяє зробити перенесення фізичного сервера на віртуальний без зупинки роботи. Таким чином, з'являється можливість онлайнового резервування цілого сервера, і в разі виходу його з ладу, можна протягом хвилини запустити віртуальний сервер і продовжити роботу.

• Міграція (конвертація) віртуальних машин інших форматів в віртуальні машини Hyper-V - технологія V2V. Дана технологія аналогічна P2V, але при цьому дозволяє переносити віртуальні машини Microsoft Virtual Server або VMware ESX в Hyper-V.

• Управління кластерами Hyper-V.

Хмарні обчислення

Хмарні обчислення (англ. cloud computing) - технологія розподіленої обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Надання користувачеві послуг як Інтернет-сервіс є ключовим. Проте під Інтернет-сервісом не варто розуміти доступ до сервісу тільки через Інтернет, він може здійснюватися також і через звичайну локальну мережу з використанням веб-технологій.

Основою для створення і швидкого розвитку хмарних обчислювальних систем послужили великі інтернет-сервіси, такі як Google, Amazon і ін., а так само технічний прогрес, що по суті говорить про те, що поява хмарних обчислень була усього лише справою часу.

Переваги хмарних обчислень: доступність - хмари доступні усім, з будь-якої точки, де є Інтернет, з будь-якого комп'ютера, де є браузер. Це дозволяє користувачам (підприємствам) економити на закупівлі високопродуктивних, дорогих комп'ютерів. Також співробітники компаній стають мобільнішими так, як можуть отримати доступ до свого робочого місця з будь-якої точки земної кулі, використовуючи ноутбук, нетбук, планшетник або смартфон. Немає необхідності в закупівлі ліцензійного ПЗ, його налаштуванні і оновленні, Ви просто заходите на сервіс і користуєтеся його послугами, заплативши за фактичне використання.

Низька вартість - основні чинники використання хмар, які знизили вартість, наступні:

- **зниження витрат на обслуговування віртуальної інфраструктури**, викликане розвитком технологій віртуалізації, за рахунок чого потрібно менший штат для обслуговування усієї IT-інфраструктури підприємства;
- **оплата фактичного використання ресурсів**- користувач хмари платить за фактичне використання обчислювальних потужностей хмари, що дозволяє йому ефективно розподіляти свої грошові кошти.

Це дозволяє користувачам (підприємствам) економити на купівлі ліцензій до ПЗ;

- **використання хмари на правах оренди** дозволяє користувачам понизити витрати на закупівлю дорогого устаткування і зробити акцент на вкладення грошових коштів на наладку бізнес- процесів підприємства, що у свою чергу дозволяє легше почати бізнес;
- **розвиток апаратної частини обчислювальних систем,** у зв'язку з чим знижується вартість устаткування.

Гнучкість - необмеженість обчислювальних ресурсів (пам'ять, процесор, диски), за рахунок використання систем віртуалізації, процес масштабування і адміністрування "хмар" стає досить легким завданням, оскільки "хмара" самостійно може надати Вам ресурси, які вам потрібні, а Ви платите тільки за фактичне їх використання.

Надійність - надійність "хмар", які особливо знаходяться в спеціально обладнаних ЦОД, дуже висока, так як такі ЦОД мають резервні джерела живлення, охорону.

Недоліки хмарних обчислень:

постійне з'єднання з мережею - для доступу до послуг "хмари" потрібне постійне з'єднання з мережею Інтернет.

конфіденційність - конфіденційність даних, які зберігаються на публічних "хмарах" в сьогодення викликає багато суперечок, але у більшості випадків експерти сходяться в тому, що не рекомендується зберігати найбільш цінні для компанії документи на публічній "хмарі", оскільки нині немає технології, яка б гарантувала 100% конфіденціальності даних, які зберігаються.

надійність - що стосується надійності інформації, яка зберігається, то з упевненістю можна сказати, що якщо Ви втратили інформацію, яка зберігається в "хмарі", то Ви її втратили назавжди.

безпека - "хмара" сама по собі є досить надійною системою, проте при проникненні на нюю зловмисник дістає доступ до величезного сховища даних. Ще один мінус - це використання систем віртуалізації, в яких як гіпервізор використовуються ядра стандартні ОС такі, як Linux, Windows та ін., що дозволяє використати віруси.

дорожнеча устаткування - для побудови власної хмари компанії необхідно виділити значні матеріальні ресурси, що не вигідно тільки що створеним і малим компаніям.

Види послуг, які надаються хмарними системами

Концепція хмарних обчислень припускає надання наступних типів послуг своїм користувачам:

- все як послуга (Everything as a Service);

При такому виді сервісу користувачеві буде надано все від програмно-апаратної частини і до управлінням бізнес-процесами, включаючи взаємодію між користувачами, від користувача потрібно тільки наявність доступу в мережу Інтернет.

- інфраструктура як сервіс (Infrastructure as a service);

Користувачеві надається комп'ютерна інфраструктура, зазвичай віртуальні платформи (комп'ютери), пов'язані в мережу.

- платформа як сервіс (Platform as a service);

Користувачеві надається комп'ютерна платформа зі встановленою операційною системою.

- програмне забезпечення як сервіс (Software as a service);

Цей вид послуги зазвичай позиціонується як "програмне забезпечення на вимогу", це програмне забезпечення розгорнуте на віддалених серверах і користувач може діставати до нього доступ за допомогою Інтернету, причому всі питання оновлення і ліцензій на це програмне забезпечення регулюється постачальником цієї послуги. Оплата в даному випадку робитися за фактичне використання програмного забезпечення.

- апаратне забезпечення як сервіс (Hardware as a Service);

В даному випадку користувачеві послуги надається устаткування, на правах оренди, яке він може використати для власних цілей. Цей варіант дозволяє економити на обслуговуванні цього устаткування, хоча за своєю суттю мало чим відрізняється від виду послуги "Інфраструктура як сервіс" за винятком того, що Ви маєте голе устаткування, на основі якого розгортаєте свою власну інфраструктуру з використанням найбільш відповідного програмного забезпечення.

- робоче місце як сервіс (Workplace as a Service);

В даному випадку компанія використовує хмарні обчислення для організації робочих місць своїх співробітників, налаштувавши і встановивши усе необхідне програмне забезпечення, необхідне для роботи персоналу.

- дані як сервіс (Data as a Service);

Основна ідея цього виду послуги полягає в тому, що користувачеві надається дисковий простір, який він може використати для зберігання великих об'ємів інформації.

- безпека як сервіс (Security as a Service).

Цей вид послуги надає можливість користувачам швидко розгортати, продукти дозволяють забезпечити безпечне використання веб-технологій, безпеку електронного листування, а також безпеку локальної системи, що дозволяє користувачам цього сервісу економити на розгортанні і підтримці своєї власної системи безпеки.

Класифікація хмарних сервісів

Зараз виділяють три категорії "хмар" :
1. Публічні;
2. Приватні;
3. Гібридні.

Публічна хмара – це IT-інфраструктура, яка використовується одночасно безліччю компаній і сервісів. Користувачі цих хмар не мають можливості керувати і обслуговувати цю хмару, уся відповідальність з цих питань покладена на власника цієї хмари. Абонентом пропонованих сервісів може стати будь-яка компанія і індивідуальний користувач. Вони пропонують легкий і доступний за ціною спосіб розгортання веб-сайтів, або бізнес-систем з великими можливостями масштабування, які в інших рішеннях були б недоступні. Приклади: онлайн сервіси Amazon EC2 і Simple Storage Service (S3), Google Apps/Docs, Salesforce.com, Microsoft Office Web.

Приватна хмара – це безпечна IT-інфраструктура, контрольована і експлуатована в інтересах однієї-єдиної організації. Організація може керувати приватною хмарою самостійно, або доручити це завдання зовнішньому підрядникові. Інфраструктура може розміщуватися або в приміщеннях замовника, або у зовнішнього оператора, або частково у замовника і частково у оператора. Ідеальний варіант приватної хмари - це хмара, розгорнута на території організації, обслуговувана і контрольована її співробітниками.

Гібридна хмара - це IT-інфраструктура використовує кращі якості публічної і приватної хмари при вирішені поставленої задачі. Часто такий тип хмар використовується, коли організація має сезонні періоди активності, іншими словам, як тільки внутрішня IT-інфраструктура не справляється з поточними завданнями, частина потужностей перекидається на публічну хмару (наприклад великі об'єми статистичної інформації, які в необробленому вигляді не представляють цінності для підприємства), а також для надання доступу користувачам до ресурсів підприємства (до приватної хмари) через публічну хмару.

На даний момент більшість хмарних інфраструктур розгорнуто на серверах датацентрів, використовуючи технології віртуалізації, що фактично дозволяє будь-якому призначеному для користувача додатку використовувати обчислювальні потужності, абсолютно не замислюючись про технологічні аспекти. Тоді можна розуміти «хмару» як єдиний доступ до обчислень з боку користувача.

Види хмарних обчислень

З поняттям хмарних обчислень пов'язують сервіси, які мають (Everything as a service) технології, такі як:
• «**Інфраструктура як сервіс»** (" Infrastructure as a Service " або " IaaS ");
• **«Платформа як сервіс»** (" Plaatform as a Service ", " PaaS ");
• **«Програмне забезпечення як сервіс»** (" Software as a Service " або "SaaS ").

Інфраструктура як сервіс (IaaS)

IaaS – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

IaaS складається з трьох основних компонентів:

• Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання);

• Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами);

• Сполучне ПЗ (наприклад, для управління системами).

IaaS заснована на технології віртуалізації, що дозволяє користувачу обладнання ділити його на частини, які відповідають поточним потребам бізнесу, тим самим збільшуючи ефективність використання наявних обчислювальних потужностей. Користувач (компанія або розробник ПЗ) повинен буде оплачувати всього лише реально необхідні йому для роботи серверний час, дисковий простір, мережеву пропускну спроможність та інші ресурси. Крім того, IaaS надає в розпорядження клієнта весь набір функцій керування в одній інтегрованої платформі.

Платформа як сервіс (PaaS)

PaaS – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб–додатків як послуги.

Для розгортання веб -додатків розробнику не потрібно купувати обладнання та програмне забезпечення, немає необхідності організовувати їх підтримку. Доступ для клієнта може бути організований на умовах оренди.

Такий підхід має такі переваги:

• масштабованість ;

• відмовостійкість ;

• віртуалізація ;

• безпека.

Масштабованість PaaS передбачає автоматичне виділення і звільнення необхідних ресурсів залежно від кількості обслуговуваних додатком користувачів. PaaS як інтегрована платформа для розробки, тестування, розгортання і підтримки веб-додатків дозволить весь перелік операцій з розробки, тестування, та розгортанню веб-додатків виконувати в одному інтегрованому середовищі, виключаючи тим самим витрати на підтримку окремих середовищ для окремих етапів. Здатність створювати вихідний код і надавати його в загальний доступ всередині команди розробки значно підвищує продуктивність по створенню додатків на основі PaaS.

Найвідомішим прикладом такої платформи є AppEngine від Google, яка пропонує хостинг для веб-додатків з можливістю купувати додаткові обчислювальні ресурси (наприклад, для тестування високих навантажень). Для запуску додатків Google AppEngine на віртуальних кластерних системах була розроблена платформа AppScale, яка не має ніякого відношення до Google.

У системах веб-пошуку і контекстної реклами компанії Yahoo використовується платформа Hadoop, орієнтована на передачу великих обсягів даних між мережевими серверами. На базі Hadoop побудовані HBase (аналог бази даних Google BigTable), а також HDFS (Hadoop Distributed File System, аналог Google File System).

Ще одним яскравим представником PaaS є продукти компанії Mosso:
-Cloud Sites – веб-хостинг (Linux, Windows, Mail) для навантажувальних веб – проектів з можливістю розширювати базові безкоштовні – можливості за додаткову плату (трафік, сховище даних, обчислювальна потужність).
-Cloud Files – файловий cloud-хостинг з щомісячною погігабайтною оплатою за обсяг збережених файлів. Управління здійснюється через браузер, або за допомогою API (PHP, Python, Java, .NET, Ruby).
-Cloud Servers – погодинна оренда серверів (RAM на годину), з можливістю вибору серверної ОС. Можна змінювати характеристики сервера, але не в режимі реального часу.

Ну а в центрі всієї хмарної інфраструктури Microsoft – операційна система Windows Azure. Windows Azure створює єдине середовище, що включає хмарні аналоги серверних продуктів Microsoft (реляційна база даних SQL Azure, що є аналогом SQL Server, а також Exchange Online, SharePoint Online і Microsoft Dynamics CRM Online) і інструменти розробки (.NET Framework і Visual Studio, оснащена в версії 2010 набором Windows Azure Tools). Так, наприклад, програміст, який створює сайт в Visual Studio 2010, може не виходячи з програми розмістити свій сайт в Windows Azure.

Програмне забезпечення як сервіс (SaaS).

SaaS – модель розгортання програми, яка надає додаток кінцевому користувачеві як послугу на вимогу (on demand). Доступ до такого додатку здійснюється за допомогою мережі, а найчастіше за допомогою Інтернет-браузера. У даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання та програмного забезпечення, що працює на ньому. Цільова аудиторія – кінцеві споживачі.

У моделі SaaS:
•додаток пристосований для віддаленого використання;
•одним додатком можуть користуватися декілька клієнтів;
•оплата за послугу стягується або як щомісячна абонентська плата, або на основі сумарного обсягу транзакцій;
•підтримка програми входить вже до складу оплати;
•модернізація програми може проводитися обслуговуючим персоналом плавно і прозоро для клієнтів.

З точки зору розробників програмного забезпечення, модель SaaS дозволить ефективно боротися з неліцензійним використанням програмного забезпечення, завдяки тому, що клієнт не може зберігати, копіювати і встановлювати програмне забезпечення.

Розвитком логіки SaaS є концепція **WaaS (Workplace as a Service – робоче місце як послуга).** Тобто клієнт отримує в своє розпорядження повністю оснащене всім необхідним для роботи віртуальне робоче місце.

Питання для самоконтролю

1.Охарактеризуйте програмне забезпечення хмарних обчислень.
2.Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація IT-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Назвати основні різновиди віртуалізації.
7. Що таке віртуалізація серверів (повна віртуалізація і паравіртуалізація)?
8. Що таке Віртуалізація на рівні ядра ОС (operating system – level virtualization)?
9. Що таке Віртуалізація додатків і Віртуалізація уявлень?
10. Що являє собою гіпервізорна технологія?
11. Охарактеризуйте архітектуру Hyper-v.
12. Охарактеризуйте Microsoft System Center Virtual Machine Manager (SCVMM).
13. Назвіть достоїнства та недоліки хмарних обчислень.
14. Перерахуйте види послуг, які надаються хмарними системами.
15. Що таке «Інфраструктура як сервіс» (" Infrastructure as a Service " або " IaaS ")?
16. Що таке «Платформа як сервіс» (" Plaatform as a Service ", " PaaS ")?
17. Що таке «Програмне забезпечення як сервіс» ("Software as a Service ", "SaaS")?

Література

1. Риз Дж. Облачные вычисления: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 288 с. ил.
2. Adobe Flash runtimes. Benefits. Режим доступу: http://www.adobe.com/products/flashruntimes/benefits.html
3.Windows Server 2008 R2 and Windows Server 2008. Режим доступу: www.microsoft.com/windowsserver2008/en/us/2008-dc. Aspx.
4. Crandell M. Defogging cloud computing: A taxonomy, June 16, 2008. Режим доступу: http://refresh.gigaom.com/2008/06/16/defogging-cloud-computing-a-taxonomy/
5. Google App Engine. Режим доступу: http://code.google.com/appengine Microsoft Azure. Режим доступу: http://www.microsoft.com/azure
6. http://www.pcweek.ru/themes/detail.php?ID=107230
7. http://www.ixbt.com/cm/virtualization.shtml

3.2 **Організація обчислень в хмарних середовищах**

Вперше мова про ідею хмарних обчислень пішла на конференції LinuxWorld 2003. Тоді звернули увагу на систему Scyld, яка була призначена для реалізації сайтів та бізнес-застосувань. Її особливістю було те, що будь-

який потік запитів до сайту завжди обслуговувався за рахунок того, що призупинялись малопріоритетні процеси, а важливі процеси динамічно переміщувались на ресурси КК (комп'ют.класт.), які вивільняються. Відміною від ОС паралельних систем була та, що система Scyld могла виконувати балансування навантаження великої кількості різних СЕ (процесор.елемент.), які мають власні ресурси та ОС різного типу.

Як і багато нових технологій, хмарні обчислення розвивались поетапно. На першому етапі хмарні обчислення розвивались як нова послуга - Software as a Service (SaaS), тобто програмне забезпечення як послуга в рамках Інтернет - стандарту Web 2.0. Почало з'являтись багато хостинг-сайтів, які були призначені для зберігання сайтів, веб-застосунків осіб та компаній. З'явились системи управління контентом (content management systems, CMS) і відповідно, традиційні сайти почали масово замінюватись на сайти на базі CMS. Тобто, тут CMS є типовим програмним забезпеченням як послуга на віддаленому сервері. Крім того, швидке зростання вартості електроенергії змусло багато фірм та організацій мінімізувати витрати, в тім рахунку використовуючи віддалений доступ до серверу.

На другому етапі відбулась віртуалізація хостингу. Віддалений доступ до серверів використовується вже не одне десятиліття і використовує, наприклад, протокол Microsoft Remote Desktop Protocol (RDP) та мережу VPN. Також давно застосовується віддалене тестування та налагодження серверів за допомогою запуску віртуального СЕ. При розвитку хмарних обчислень ця ідея набула подальшого розповсюдження у вигляді віртуалізації хостингу. При цьому віртуальний СЕ налаштовується разом з ОС та програмним забезпеченням у користувача, а потім цей СЕ разом з програмним забезпеченням перевстановлюється у віддаленому сервері.

Додатковий поштовх для розвитку віддаленого хостингу дали сплески цін на енергоносії у середині нульових років, через що тримати власний датацентр стало досить витратно. На цю тенденцію звернули увагу такі фірми, як Microsoft і Amazon, які почали будувати центри обробки даних в районах з джерелами недорогої енергії та з прохолодним кліматом, щоб мінімізувати витрати на живлення та охолодження центрів.

На третьому етапі були впроваджені дійсно хмарні обчислення. Хмарні середовища були результатом досвіду експлуатації Грід-систем та уроків віртуалізації хостингу. Цей досвід показав необхідність розробки галузевих стандартів для організації хмарних обчислень.

Хмарне середовище схоже як на Грід-систему, так і на КК (комп'ютерний кластер) у тому, що хмарні обчислення можуть бути організовані як у Грід-системі, так і у КК, якщо віртуальному Web-серверу користувача надати відповідні ресурси.

Тиск ринку змушує розвиток конкуруючих технологій хмарних обчислень стандартизувати ці технології. 27 жовтня 2008 р. фірма Microsoft першою представила свою версію хмарної ОС. Слід зауважити що хмарна ОС - це програмне забезпечення, яке дозволяє виконувати у хмарному середовищі один або декілька застосунків, які в свою чергу, дають змогу їх використовувати кінцевими користувачами.

Віртуальні процеси як основа хмарних обчислень

Реальні обчислювальні процеси у системі обробки даних, як правило, простоюють до 80-90% часу, очікуючи готовність даних. За рахунок віртуалізації можна запустити кілька обчислювальних процесів на одному реальному СЕ, завдяки чому підвищити ефективність використання системи. Крім того, якщо ця віртуалізація виконується в Грід-системі, або КК, то досягається економія енергоспоживання, оскільки у цих системах ефективність джерел живлення досягає 95% . Крім того кожен віртуальний процес можна зупинити, зберегти, переслати і поновити виконання у будь-якому місці хмарного середовища.

Одним з головних факторів, які ускладнюють організацію обчислень у хмарному середовищі, є відсутність ефективних засобів для переміщення застосування у хмарне середовище. Досі процес автоматичного завантаження завдання у хмару реалізується за допомогою набору утиліт від несумісних систем (Microsoft, Amazon, Google і т.д.). З боку користувача існують програми, які часто некоректно називаються хмарними ОС, які допомагають переслати дані та команди у хмарне середовище. Це, наприклад, системи Dropbox і Microsoft Mesh. Але на стороні сервера все набагато складніше через проблеми, які виникають при віртуалізації процесів.

Віртуалізація процесу пов’язана з його абстрагуванням. Тут під абстрагуванням мається на увазі визначення місцезнаходження процесу серед рівнів обчислювальної системи (від рівнів апаратури, драйверів, ОС до рівня користувацької програми) та визначення механізмів взаємодії з нижчерозташованим та вищерозташованим рівнями. Наприклад, рівень .NET, який додано у ОС Windows, успішно сприяє розробці мережевих застосунків, бо він приховує від розробника усі рівні, що лежать нижче, тобто абстрагується від них. Такий самий рівень необхідний для реалізації хмарних обчислень. Тоді відпадає необхідність для користувачів і адміністраторів явно визначати СЕ, на якому буде виконуватись конкретний застосунок.

Для віртуалізації у хмарному середовищі використовується рівень гіпервізора. Коротко ми це розглянули в попередній лекції, але вона була досить об’ємна і тому там я нічого не сказав про Windows Server 2012, плануючи присвятити цьому питанню окрему лекцію. Але так як кількість лекцій обмежена (їх має бути 18), коротко зупинюсь на цьому питанню в цій лекції.

Гіпервізор (ще називається монітор віртуального СЕ, virtual machine monitor, VMM) - це програмна або апаратно-програмна платформа для віртуалізації програмного забезпечення, яка дає змогу кільком ОС одночасно виконуватись на одному фізичному СЕ. Наприклад, гіпервізор Windows Server 2008 R2 Datacenter дає права на запуск необмеженої кількості ОС Windows Server на сервері.[1]

Можливості Hyper-V в сімействі операційнх систем Windows Server 2008 вивели Microsoft на гідне місце на ринку віртуальних серверів, однако Hyper-V все ще був на крок позаду свого основного конкурента, VMware, в відношенні масштабованості, високої доступності та функціональності. В Windows Server

2012 Microsoft догнала VMware і в цих питаннях. Розробкою Hyper-V програмісти Microsoft заклали основу для інтегрованої технології віртуалізації серверів прямо в базовій операційній системі Windows Server. Базова технологія, відома як гіпервізор (hypervisor), по суті являє собою рівень в середині головної операційної системи, яким забезпечує кращу підтримку гостьових операційних систем. Як вже було сказано, свою гіпервізорну технологію компанія Microsoft назвала Hyper-V. До включення Hyper-V в Windows Server 2008 і Windows Server 2012 стек віртуалізації розміщувався поверх головної операційної системи і, по суті, вимагав, щоб всі гостьові операційні системи спільно використовували системні ресурси, такі як мережеві комунікаційні ресурси, можливості відеообробки та виділення памяті. При системному збою в головній операційній системі, наприклад, відмові драйверу мережевого адаптеру, всі гостьові сеанси втрачають можливості обміну даними по мережі.

Технології, подібні VMware ESX, Citrix XenServer і Hyper-V збільшують ефективність технологій, побудованих на основі гіпервізору, дозволяючи гостьовим операційним системам ефективно обходити головну операційну систему і безпосредньо обмінюватися даними з системними ресурсами. В одних випадках гіпервізор керує спільно використовуємими ресурсами гостьового сеансу, а в інших випадках гостьові сеанси оминають гіпервізор і відправляють запити безпосередньо на апаратний рівень системи. Забезпечуючи більшу незалежність обміну даними між системами, середовище з гіпервізором предоставляє організаціям більшу степінь масштабованості, більш високу продуктивність і більш високу надійність базового середовища віртуального хосту.

В Windows Server 2012 технологія Hyper-V підтримується тільки в 64-розрядних системах з апаратною підтримкою віртуалізації. ЦП має підтримувати опцію Intel VT, або AMD-V і режим захисту від виконання даних (Data Execution Protection — DEP). Крім того, ці функціональні можливості повинні бути включені в BIOS комп'ютера.[2]

Hyper-V в Windows Server 2012 містить значні покращення, які зробили Hyper-V не тільки достойним товаром на ринку віртуалізації серверів, але і лідером, який підняв планку вимог до технологій віртуалізації серверів в організаціях. Він містить багато довгоочікуваних компонентів і технологій, які можна розбити на три великі категорії: збільшена потужність хосту і гостьових сеансів, інтегровані технології високої доступності і покращена керованість. В Hyper-V є можливість розміщення гостьових операційних систем на серверах Windows, кліентських системах і системах з іншими ОС. А всякі інструменти і керування віртуальними хостами і гостьовими сеансами підсилюються використанням диспетчеру віртуальних машин (Virtual Machine Manager — VMM) із системного центру Microsoft System Center 2012. VMM позволяє створювати відображення фізичного серверу на віртуальний і копіювати віртуальний сервер на віртуальний та розширює поняття керування з віртуальних гостів і хостів на всю мережеву "фабрику". Фабрика мережі включає управління SAN, створення віртуальних локальних мереж (virtual

local-area network — VLAN) і автоматичне створення повних дво- і трьохрівневих зв'язок серверів.

Центр керування системою System Center 2012 доповнює можливості управління і адміністрування хостами Hyper-V. Весь пакет продуктів System Center 2012 описаний в книзі System Center 2012 Unleashed, яка присвячена не тільки використанню VMM для автоматизації процесів в ускладнюючихся середовищах гостьових сеансів, але і інших засобів: диспетчер конфігурації для виправлення і керування серверами-хостами і віртуальними гостьовими сеансами, диспетчер захисту даних для архівації хостів і гостьових серверів і інші компоненти їх семейства System Center.

Іншою прогресивною технологією, яка поширюється разом з віртуалізацією, є технологія мінімізації програмного забезпечення, яке виконується. Замість повної версії ОС для кожного віртуального СЕ, серед цих СЕ поширюється лише ядро ОС та необхідні додатки до нього, а загальні додатки розділяються серед кількох віртуальних СЕ. Завдяки цьому скоротились витрати на пересилки програм.

Іншим важливим аспектом є керування віртуальним СЕ. Можливість змонтувати образ диску, породжуючи новий екземпляр ОС з диску віртуального образу ОС, з'явилась майже одночасно в системах управління для блейд-серверів і у консольних застосунках віртуальних машин. Ця можливість дала змогу віддалено керувати віртуальним СЕ та виконувати автоматизоване балансування навантаження.

У той же час, можливість запуску віддаленої консолі для кожного віртуального СЕ з браузеру дала змогу продовжувати керування цим СЕ після того, як на нього встановлена ОС. Але зростаюча залежність від рівнів абстракції і розмноження екземплярів ОС призвело до наступної проблеми. Хоча рівень абстракції призначений для фізичного розділення нижнього рівня від верхнього, але ці рівні виявляються неспроможними бути цілком розділеними, бо система перестане функціонувати.

У той час, як віддалені консольні застосунки були вже розроблені і впроваджені, програми моніторингу мережі, такі як HP Open View, OpenNMS, Пакет Trap, Nagios, і UniCenter тільки почали використовуватись для аналізу віртуалізованого середовища. Така програма моніторингу може забезпечити точну інформацію в реальному часі, наприклад, про те, коли апаратні ресурси можуть бути перевантаженими, або коли тривалість ліцензії добігає до кінця.

Завдяки таким програмам та віртуалізаціїї процесів, у одній стійці сервера можна реалізувати сотні і тисячі віртуальних ОС. Помітною подією була поява блейд-серверів, або систем 234 (модульні, розроблені в 2001р, займають в 2 рази менше місця, споживання в три рази менше енергії, дешевші в 4 рази) серії С3000 фірми Hewlett-Packard. Такий сервер набирається з компактних процесорів та пакетів жорстких дисків у металевих корпусах (блейдах), які можуть вийматись та встромлятись у задню панель стійки без порушення роботи сервера. При цьому у блейдах, які щойно підключені до живлення та швидкодіючої локальної мережі, розміщується необхідна кількість нових віртуальних СЕ.

Поява та удосконалення хмарних обчислень

Отже, поява рівня абстракції віртуальної ОС, механізму віртуалізації та можливості переходу віртуального СЕ з одного фізичного СЕ на інший були передумовою для появи хмарних обчислень. Виникає питання, що відрізняє хмарні обчислення від віртуалізації? Спочатку вважалось, що відміна у розташуванні серверів та комп'ютерів користувача. Насправді, віртуалізація означає встановлення екземплярів віртуальних ОС на серверах, що належать організації користувача. А хмарні обчислення надаються на серверах, розташованих у "хмарі" Інтернету, в місцях, які навіть не відомі користувачу. Тобто, обчислювальні ресурси, які виділяються, не мають конкретного місця і є фізично недосяжними для користувача.

Отже, за визначенням, ***хмарні обчислення*** - це набір комп'ютерних послуг, які надаються виключно в якості служби, без відомостей про конкретний сервер, який замовляється споживачем. Провайдер хмарного сервісу (ПХС, CSP) має забезпечити встановлення віртуальних ОС будь-якого типу. Ефективність ПХС залежить від диспетчера завдань (scheduler), який складається з брокера та агента.[3]

Брокер надає інтерфейс користувача для зовнішніх процесів, щоб спілкуватись з хмарним застосунком, а агент – це програма, яка сканує хмарні сервіси, щоб знайти дані для застосунку користувача. Наприклад, брокер – це вікно браузера, а агент – засіб гортання сторінок Веб-магазину. Разом ці дві частини планувальника узгоджують рівні абстракції і компонентів, щоб організувати виконання застосунку у хмарі. Таке планування дозволяє підвищити рівень абстракції компонентів, тобто тепер застосунок можна створити з великої кількості розрізнених компонентів, навіть розміщених у різних ПХС.

Іншою ознакою хмарних обчислень є концепція групової ідентифікації (*identity federation*). При груповій ідентифікації користувачу досить зареєструватись у першої організації-постачальника послуг, що дає право не реєструватись у інших організацій, бо вони можуть перевірити права користувача, звернувшись до першої організації. Наприклад, зареєструвавшись в Facebook, користувач може не реєструватись на інших сайтах.

Ще однією особливістю хмарних обчислень є служба з'єднання (cloud dating service). Це засіб перевірки прав користувача третьою стороною. Наприклад, користувач починає підключення до веб-сайту, який в свою чергу починає процедуру створення захищеного з'єднання. Цей сайт має набір облікових даних, які надсилаються на веб-браузер користувача. Браузер, у свою чергу, посилає ці зашифровані облікові дані на сторонній сервер сертифікації. Після успішної сертифікації веб-сайт дає відповідний дозвіл для роботи користувача.

У хмарному середовищі часто застосовується також самоадміністрування. Концепція віртуальних машин, які самоадмініструються, основана на можливості автоматичного породження нових екземплярів віртуальних СЕ при необхідності збільшення продуктивності для виконання завдання. Такі можливості, наприклад, закладені в ОС VMotion фірми VMWare.

Користувачі хмарного середовища повинні мати можливість розробляти або налаштовувати додаток, протестувати його на локальних ресурсах, а потім через агента передати його постачальникам хмарних послуг. Великою

перевагою хмарного середовища є можливість "взяти в оренду" застосунки, розроблені іншими, для того, щоб швидше задовольнити свої нові потреби.
Новий вид послуг, коли береться в аренду стороннє програмне забезпечення, надається рядом постачальників, у тім рахунку Amazon, Google і IBM. Для розвитку цього напрямку необхідно створити стандартний користувацький інтерфейс та стандартну платформу (додатковий рівень абстракції) для виконання застосунків, яка не залежить від рівня ОС, який знаходиться нижче.
Одним з перших прикладів того, як має виглядати середовище для розробки хмарних застосунків, є середовище Adobe Integrated Runtime (AIR). AIR має частини, які необхідні для кожної системи розробки для хмарних обчислень, в тім рахунку:
• дескриптори запитів ресурсів у форматі XML;
• можливість підтримки декількох об'єктно-орієнтованих систем Веб-розробки (Java, Flash);
• можливість підтримки інтерфейсів Інтернету, баз даних і апаратних інтерфейсів, які не залежать від типу ОС (Windows чи Linux).

Класифікація хмарних систем

На сьогодні прийнято три різних класифікації видів хмарних обчислень, які з'явилися в різний час еволюції у цій комп'ютерній сфері.

1 SPI - класифікація

Протягом перших впроваджень хмарних обчислень вони зразу були розділені на три основних категорії: програмне забезпечення як послуга (Software as a Service, SaaS), платформа як послуга (Platform as a Service, PaaS), і інфраструктура як послуга (Infrastructure as a Service, IaaS). За першими буквами цих категорій ця класифікація й одержала назву SPI Розглянемо їх конкретніше.
SaaS - це підмножина хмарних систем, яка призначена для виконання застосунків, створених для хмари і розгорнутих у хмарі, які знаходиться в Інтернеті. Цільовий користувач SaaS є кінцевим користувачем. Ці застосунки - хмарні застосунки, як правило, побудовані на основі браузера і мають наперед визначені функціональність та можливості. Прикладом SaaS є застосунки Google, такі як Google Docs і Google Spreadsheets.

SaaS вважається привабливою альтернативою застосункам на ПК. Наприклад, маючи застосунок, розгорнутий на сервері провайдера, зменшуються вимоги до обладнання ПК, немає проблем з технічним обслуговуванням, чи з модернізацією програмного забезпечення.
В випадку PaaS провайдер забезпечує наявність надійної програмної платформи з інтерфейсом прикладного програмування (API), яка може бути використана в розробці хмарних застосунків. Одним із прикладів систем в цій категорії є платформа Google App Engine, яка забезпечує середовище і API-інтерфейси для виконання програм на мовах Python і Java, призначених для сфери Google.[4] Microsoft Azure також відноситься до PaaS.[5]
Розробка програми для хмарної платформи подібна до розробки веб-застосунку для веб-сервера в тому сенсі, що результуюча програма буде розгорнута на віддаленому сервері. Однак модель PaaS відрізняється тим, що платформа може надавати додаткові послуги, щоб спростити розробку програм, їх розгортання і

виконання, наприклад, автоматичного масштабування, моніторингу та балансування навантаження, спрощення програмування таких функцій, як служб аутентифікації, служб електронної пошти і компонентів інтерфейсу користувача.

Крім того, розробники програми можуть інтегрувати інші послуги, які надаються системою PaaS для їх застосування, наприклад, служб аутентифікації, служб електронної пошти і компонентів для користувача інтерфейсу, механізмів вимірювання завантаження, чи організації платежу за послуги. Все, що забезпечується за рахунок набору API, поставляється з платформи. В результаті, клас PaaS, як правило, призначений, щоб прискорити розробку програмного забезпечення і час розгортання. В свою чергу, програмне забезпечення, яке побудовано для хмарної платформи, зазвичай має більш короткий час виходу на ринок. Деякі наукові проекти з'явилися також на підтримку більш повного розуміння PaaS, таких як AppScale.

Клас систем IaaS забезпечує користувачів ресурсами інфраструктури, такими як обчислювальні ресурси, зберігання даних, послуги зв'язку. Найбільш відомий приклад – це Elastic Compute Cloud (EC2) фірми Amazon.[6]

Досягнення в області віртуалізації ОС сприяли реалізації IaaS на існуючих серверах. В результаті, віртуалізації ОС є можливість відноситись до усього програмного забезпечення та пов'язаних з ним ресурсів окремого користувача як до об’єкту, який можна призначити на фізичні ресурси, які знаходяться будь-де і виділити йому відповідну частку робочого часу. Тому віртуалізація ОС дає змогу провайдерам IaaS керувати ефективним використанням фізичних ресурсів при збереженні знайомого інтерфейсу до стандартних апаратних платформ і мереж з використанням існуючої практики створення і використання програмного забезпечення.

2 Класифікація UCSB-IBM

Ця класифікація була виконана при співробітництві між Каліфорнійським університетом в Санта-Барбарі (UCSB) та дослідним центром фірми IBM. У цій класифікації розглядаються взаємовідносини різних рівнів хмарного середовища за принципом можливості рівнів об’єднуватись для реалізації складеного сервісу (composability), що запозичено з класифікації сервісно-орієнтованих архітектур (SOA). Тоді хмарний сервіс – це композиція одного чи більше інших сервісів.

Модель UCSB-IBM складена з п’яти рівнів відповідно до цього принципу. Кожен рівень охоплює один або кілька хмарних сервісів. Хмарні сервіси відносяться до одного рівня, якщо вони мають однаковий рівень абстракції. Наприклад, всі хмарні середовища (хмарні платформи) орієнтовані на програмістів, а хмарні застосунки - на користувачів. Тому перші та другі належать до різних рівнів. У моделі UCSB-IBM рівні формують стек хмари, причому один рівень знаходиться нижче у стеку, якщо він забезпечує виконання послуг у рівні, який вище. Графічно ця модель показана на рис.3.2.1.

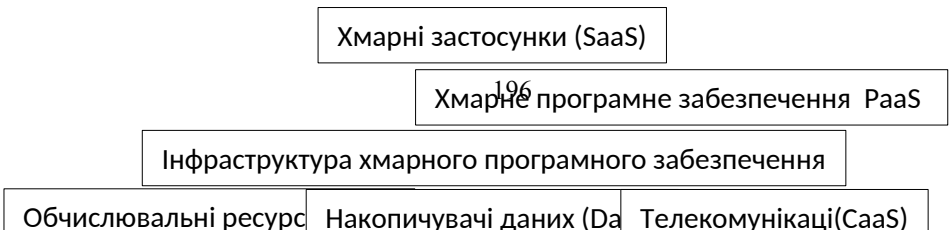


Рис.3.2.1. - Класифікація хмарних обчислень UCSB-IBM

Перші три рівні класифікації хмарних обчислень UCSB-IBM аналогічні класифікації SPI, за винятком того, що рівень інфраструктури розділено на три компоненти: обчислювальні ресурси, пам’ять та накопичувачі даних і зв'язок.
Перший – верхній рівень – це рівень застосунків (SaaS). Як правило, користувачі отримують доступ до послуг цього шару через браузер. Практикою доведено, що ця модель приваблива для багатьох користувачів, оскільки це полегшує обслуговування програмного забезпечення і зменшує допоміжні витрати. Крім того, вона забезпечує передачу обчислювальної роботи від терміналу до центрів обробки даних, що зменшує вимоги до обладнання, дозволяє отримати високу продуктивність без великих капіталовкладень.

Завдяки поширенню послуги SaaS відбувається зростання нового класу пристроїв користувача у вигляді нетбуків, тобто дешевих пристроїв кінцевих користувачів з невеликим розміром пам’яті та малопотужним процесором, робота яких основана на мережевих підключеннях і хмарних застосунках.
Що стосується постачальників хмарних послуг, ця модель спрощує їх роботу по відношенню до модернізації та тестування коду, захисту інтелектуальної власності.

Хмарні застосунки розробляються з використанням хмарного програмного забезпечення та компонентів інфраструктури. Крім того, хмарні застосунки можуть компонуватися як сервіс від інших служб з використанням понять SOA. В цьому відношенні, хмарні застосунки, призначені для вищих рівнів, стають простіше в розробці, в них менше помилок, так як всі їх взаємодії з хмарою виконуються через перевірений програмний інтерфейс. Але безпека і доступ до хмарних застосунків є основним питанням у цій моделі. Крім того, невирішеною проблемою залишається боротьба з простоями ресурсів у хмарі, включаючи простої від відмов мережі та СЕ. Крім того, інтеграція множини застосунків і переміщення даних користувача в хмару гальмують поширення моделі SaaS, що пов’язане з побоюваннями за ненадійність зберігання конфіденційних даних, аутентифікації і авторизації, безвідмовної роботи і продуктивності, резервного копіювання і відновлення після збоїв.

Другий рівень – це рівень програмного забезпечення або програмної платформи. Користувачами цього рівня є розробники хмарних застосунків. Провайдери забезпечують розробників відповідними засобами програмування та чітко визначеним інтерфейсом API. Таку послугу називають платформою як сервіс (PaaS). Прикладами такої платформи є система з мовою Apex фірми Salesforce, що дозволяє розробникам створювати крім власне програм макет

сторінки, об’єднання модулів і звіти клієнтів.[7] *PaaS* також забезпечує автоматичне масштабування і балансування навантаження, а також інтеграцію з іншими послугами (наприклад, послугами аутентифікації, електронної пошти та створення інтерфейсу користувача). Отже, більша частина накладних витрат розробки хмарних застосунків полегшується і оброблюється на рівні хмарного програмного забезпечення. Це спрощує і прискорює розробку хмарних застосунківння, і зводить до мінімуму кількість помилок. Для цього розроблюються спеціальні програмні комплекси, такі як Map Reduce [8], Hadoop [9].

Третій рівень - це рівень інфраструктури хмарного програмного забезпечення, який складається з трьох частин: обчислювальних ресурсів, системи зберігання даних та телекомунікаційної системи.

Віртуальні СЕ - це загальна форма надання обчислювальних ресурсів для користувачів в цьому шарі. Віртуалізація ОС дає користувачам гнучкість у налаштуванні їх параметрів, захищаючи фізичну інфраструктуру серверів провайдера. Користувачі зазвичай отримують доступ суперкористувача, щоб налаштувати програмний стек на їх віртуальних СЕ для підвищення продуктивності та ефективності. Тому такі послуги назвали інфраструктура як сервіс - IaaS.

Ресурс інфраструктури зберігання даних - DaaS - дає змогу користувачам зберігати свої дані на віддалених дисках і отримати до них доступ в будь-який час з будь-якого місця і це полегшує масштабування хмарних застосунків. Прикладом комерційної хмарної Daas системи є S3 фірми Amazon [10].

Телекомунікації – це життєво важливий компонент хмарної інфраструктури, якість якого забезпечує якість усього хмарного сервісу. Тому заради досягнення високого рівня цієї якості виділяється в окрему категорію телекомунікації як послуга (CaaS), які між іншим повинні відповідати вимогам мережевої безпеки, динамічного виділення віртуальних каналів для ізоляції трафіку, регулювання трафіку, гарантовану межу затримки повідомлення, шифрування зв'язку та моніторингу мережі.

Рівень ОС та проміжного програмного забезпечення забезпечує основні функції управління програмним забезпеченням для фізичних серверів, які формують хмару. Цей рівень може бути реалізований у вигляді ядра ОС, гіпервізору, віртуальних СЕ, монітору і / або кластеризації проміжного програмного забезпечення. Фактично – це рівень, на якому стоїть рівень застосунків Грід-систем. Системи Globus[11] і Condor[12] є двома успішними системами проміжного програмного забезпечення для Грід-систем. Зараз багато концепцій, які були розроблені для Грід-систем, використовуються у хмарних обчисленнях. У подальшому можливе запозичення нових розробок проміжного програмного забезпечення хмар для Грід-систем.

Апаратне забезпечення та вбудоване програмне забезпечення формують нижній рівень стека рівнів хмарних обчислень в класифікації UCSB-IBM.

У класифікації UCSB-IBM відсутній аспект безпеки хмарних обчислень. З акцентом на підтримку хмарних обчислень для урядових установ у роботі[13] представлена адаптована класифікація UCSB-IBM.

Рівень управління доступом додається над рівнем хмарних застосунків і призначений для забезпечення керування доступом до програм, що реалізують SaaS. У вигляді різних методів аутентифікації, цей рівень може забезпечити спрощену і уніфіковану, але ефективну форму захисту. В свою чергу, це може спростити розробку і використання додатків SaaS з вирішенням проблем безпеки для цих систем.

3 Модель хмари Хоффа

Під впливом перших двох класифікацій хмарних обчислень Крістофером Хоффом була розроблена більш розширена класифікація у вигляді моделі хмарних обчислень [14,15].

У цій моделі увага приділяється аналізу трьох основних хмарних сервісів: IaaS, PaaS і SaaS. У моделі рівень IaaS складається з кількох компонентів, перший з них характеризується продуктивністю і простором (ємністю пам'яті), дугий - це обладнання, яке складається з набору СЕ, сховища даних та мережевої системи. Третій компонент - засоби абстрагування, такі як монітор віртуальних СЕ, утиліти звернення до Грід-системи, кластеру, а також засоби мережевого з'єднання та постачання послуг, які забезпечують різні сервіси, наприклад, аутентифікація, DNS. Компонент API представляє послуги з управління, а також спрощений інтерфейс до наступного рівня в хмарі. Наступний рівень в моделі Хоффа, який є PaaS, складається з одного підрівня, який надає послуги з інтеграції в хмарі і забезпечує декілька послуг, таких як аутентифікація, організація баз даних і запитів до них.

Рівень SaaS також розбитий на кілька підрівнів і компонентів. Підрівень даних означає фактичні дані, метадані і їх зміст, який може бути у структурованій або неструктурованій формі. Прикладний компонент в рівні SaaS поділяється на три категорії: стандартні застосунки, веб-застосунки та вбудовані застосунки. Останні два підрівні у рівні SaaS - це API додатків та підрівень застосунків для відображення даних, який далі розкладається на відображення даних, аудіо- відеоданих, тобто різних форм представлення даних у хмарі.

Класифікація Хоффа відображає більш детальну інформацію про склад хмарного середовища, яка показує додаткові аспекти та особливості хмарних обчислень. Вона доповнює та уточнює дві попередні класифікації. Онтологічний підхід, задіяний в двох останніх класифікаціях дає змогу порівнювати існуючі та нові хмарні середовища, ототожнюючи більш ефективно їх нові аспекти та надихаючи на створення нових більш ефективних сценаріїв використання хмари, характеристики в світлі їх залежності від інших обчислювальних полів і моделей.

Розуміння компонентів хмарного середовища дає змогу знайти нові рішення загальних проблем хмарних обчислень, таких як доступність, міграція застосунків між хмарними середовищами, стійкість системи до відмов, хакерських атак, тощо.

Розробка застосунків для хмарного середовища

1. Переваги веб-браузеру як інтерфейсу користувача

Хмарні застосунки повинні мати функції, які розділені на частини: програми, які виконуються на боці користувача і програми, які виконуються на сервері. Але у випадку хмарних обчислень, частина користувацького інтерфейсу, як правило, ближче до користувача, і становить основну частину, яка потребує розробки.

Поширена помилка, що хмарні застосунки повинні по формі нагадувати традиційні веб-сайти. Однак, коли почали з'являтися нові середовища розробки, такі як Adobe AIR, стала розмиватись традиційна відміна між застосунками, які керуються з командного рядка та застосунками, які керуються з веб-браузера. Нова тенденція полягає в тому, що веб-застосунки не повинні бути обмежені межами веб-браузера.

Веб-браузер став зручним способом для виконання багатьох застосунків, оскільки він дозволяє просте розгортання у різних ОС і спрощене обслуговування застосунків. Крім того, сучасні мови програмування, які використані в браузері, дозволяють швидко розробити застосунки, які зв’язуються з віддаленим сервером. Так, середовище Adobe AIR доповнює браузер, надаючи ті самі переваги розробки застосунку, як і при розробці звичайної програми для РКС (розподіл. сист.), додаючи можливості багатофункціональних інтернет-застосунків (Rich Internet Applications, RIA) [16] .

Програмування функцій браузера з самого початку непросте. Наприклад, серйозною задачею є виділення контексту, на який вказує курсор на екрані. Але тепер функції браузера так тісно інтегровані в ОС, що стало можливим звертання до функцій браузера за допомогою операторів мови програмування. Тепер ключовими компонентами застосунків стають функції відеоконференцій та плагіни, такі як Enterprise Connect від Adobe, WebEx, або Live Meeting від Microsoft.

Регулярна архітектура платформи браузера несе ряд переваг і кілька суттєвих ризиків. Серед переваг є просте підключення функцій, плагінів, невеликий об’єм необхідної пам'яті та ресурсу СЕ для них, просте розширення функцій. Недолік, як правило, пов'язаний з безпекою, так як користувачі часто додають нові функції до браузера, не завжди усвідомлюючи наслідки цього.

2. Плагіни та веб-платформи

Поведінка і вплив плагінів ускладнюється тим, що деякі плагіни діють без втручання користувача і можуть сильно впливати на роботу застосунку, який розробляється. Наприклад, відомо, що при виборі плагінів певних панелей інструментів, змінюються домашні сторінки, вибір інших додатків і т.д.

Багато користувачів не знайомляться детально з призначенням плагінів. В деяких випадках використання плагінів неусвідомлено призводить до протизаконного поводження з інформацією. Хоча є правильно розроблені плагіни, які не виконують побічних дій, займають мало місця і самостійно видаляються з комп’ютеру, якщо в них немає необхідності.

При цьому слід відмітити революційний вплив появи таких платформ, як Ruby On Rails, Flex і Ajax, які служать для абстрагування на рівні програмування і які призвели до істотного зрушення в сутності програм, які називаються застосунками. Все, що виконують ці системи, також може бути зроблено на

мові нижчого рівня, тобто, застосовуючи PERL або PHP замість Ruby On Rails. Але прийдеться витратити значно більше часу для розробки та налагодження такої програми.

Ще однією перевагою програмування на рівні абстракції є те, що ці системи, як правило, стандартизовані, що також збільшує можливість повторного використання коду.

Недоліком застосування веб-платформ є те, що втрачається швидкість роботи. Для того, щоб відреагувати на будь-яку потенційну ситуацію, веб-платформа повинна мати велику бібліотеку функцій, щоб охопити більшість ситуацій. Так, видача повідомлення на екран може бути закодована одним рядком на такій мові, як Python, але він розширюється до декількох сотень або навіть тисяч рядків коду під час їх інтерпретації, коли викликаються відповідні бібліотечні функції. Отже, таке гальмування виконання веб-функцій є передумовою необхідності постійного зростання продуктивності комп'ютерів, в тому рахунку є виправдовуванням застосування закону Мура.

3. Низькорівневі та високорівневі мови програмування

Вважається, що найшвидкіші програми слід складати на мовах низького рівня, тому, що компілятор такої мови не здатен додавати надлишок команд, необхідних для запобігання усіх можливих випадків. Тому набагато легше налаштувати таку програму при жорстких часових обмеженнях, тому що в ній немає кодів, які необхідно мінімізувати.

Мови ассемблера та С і раніше були найпопулярнішими мовами програмування для систем керування і драйверів пристроїв через їх надзвичайно короткий код. Але недоліком є те, що такі короткі програми важко складати і праця такого програміста є високовартісною. Тому не дивно, що такі рівні абстракції, як Ruby, розроблялись на мовах Rails, Python, SPSS, SAS і т.д. Так що верхні рівні абстракції спрощують поєднання програмних блоків один з одним.

Прикладом ефективного програмного середовища для розробки застосунків, яке представляє високий рівень абстракції, є Adobe AIR (Adobe Integrated Runtime). AIR дозволяє розробникам писати програми, які абстрагується від основної ОС і апаратного забезпечення. Ця платформа забезпечує проміжний шар, який створює ілюзію у Java- чи Flash-програмістів, що вони працюють над ОС Microsoft Windows або Apple Macintosh. AIR також дає інструменти для розробки в галузях HTML / AJAX, Adobe Flash і Flex.

Аналогічним середовищем є Microsoft .NET, яке дає змогу суттєво зменшити роботу, необхідну для отримання доволі складних систем. Але .NET обмежене ОС сімейства Microsoft, ігноруючи той факт, що основна частина веб-серверів у світі побудована на Apache, велика частина яких встановлена на ОС Linux. Середовидще Microsoft Silverlight, хоча має гірші властивості, ніж AIR, але дає змогу маніпулювати Web-даними, такими, які забезпечуються послугами Netflix відео за запитом.

Отже, низькорівневі мови програмування – це не для Веб-застосунків. Вони використовуються для розробки середовищ, які підвищують рівень абстракції програмування користувача, завдяки чому таке програмування спрощується, а

самі застосунки можуть бути переміщувані і незалежні як від апаратури, так і ОС.

4. Бази даних в Інтернеті

Програмування веб-застосунків потребує стандартний спосіб доступу баз даних. Для цього був створений рівень абстракції, такий як ODBC (Open Database Connectivity), який став стандартним для кількох ОС. Іншим чинником, який дозволяє звертатись до хмарної бази даних, є загальний інтерфейс шлюзу (Common Gateway Interface, CGI), створений у 1993 р. і ставший основою для великої кількості інших засобів, призначених для створення веб-застосунків. Цей інтерфейс стандартизує методологію для зв'язку програм за межами веб-сервера, так що складні застосунки можуть бути пов'язані з базами даних у веб-середовищі.

5. Хмарне середовище

Хмарне середовище –це найвищий рівень абстракції, який приховує системні деталі від кінцевих користувачів. Існує тенденція, що хмарне середовище будується таким чином, що користувач не бачить ОС, але бачить велику бібліотеку застосунків, з якої вибирає для себе потрібні з оплатою їх використання.

Для Вас у хмарному середовищі найцікавішим є рівень SaaS. Але розробити, відлагодити та розмістити застосунки у хмарному середовищі – цього замало для успішної реалізації програмного проекту. Для досягнення успіху бажано дотримуватись наступних рекомендацій .[17]

Слід навести зв’язки з публічними мережами. Якщо буде посилання на SaaS в публічній мережі, такий як LinkedIn, Фейсбук, то більша ймовірність того, що застосунок побачать і знайдуть ті, кому він потрібен. При цьому його можна подати з різними ухилами: навчальний, науковий, технічний, соціальний. Головне, щоб його контент був: оригінальним, корисним, оптимізованим для серверів-пошуковиків.

Бажано на початку розповсюдження сервісу та частково при його експлуатації зробити його безкоштовним. Головне - зачепити користувача на гачок тим, що новий SaaS – це проста у використанні і ефективна розробка.

У хмарного сервісу зручно налаштувати електронну оплату. Це спрощує процес продажів і робить прозорим цей процес для покупця.

SaaS повинен мати властивості допомагати у навчанні користувача. Взагалі, SaaS розрахований на те, що користувач без сторонньої допомоги зможе запустити застосунок, розібратися в його роботі і отримає очікуваний результат. Тобто, користувач повинен сам себе навчати. Тому SaaS повинен мати навчальні можливості, а не бути розрахованим на експерта. Бажано, щоб SaaS був чимось на зразок квест-гри, щоб під час оволодіння застосунком користувач поступово знаходив його нові корисні можливості.

У застосунок повинні бути вбудовані функції обліку досвіду користувача – які моделі і якої складності він будує, який час займає їхня оптимізація тощо. Якщо SaaS враховує цей досвід, то він може бути під нього оптимізований. Такий досвід - це надбання, якого немає у конкурентів.

Слід намагатись тримати SaaS подовше затребуваним. Для цього треба постійно удосконалювати застосунок, нарощувати його функціональність, інтегрувати з іншими SaaS, залучати нові гілки у дереві його обходу.
Слід організувати службу зворотніх зв'язків з користувачами. Через відгуки, питання споживачів та відповіді на них можна організувати співтовариство любителів нового застосунку SaaS. Це дасть енергійний поштовх поширенню SaaS по мережі.

Слід організувати базу знань застосунку, яка виглядає як сайт вікіпедії. Це означає, що знання користувачі за їх бажанням можуть об'єднуватися в базу знань спільноти любителів застосунку SaaS.

Важливим аспектом SaaS є вбудовані інструменти підрахунку, наприклад, трафіку. Для вдалого маркетингу і поліпшення SaaS тут також потрібні різноманітні метрики: скільки кліків зробив користувач, які користувачі і як використовували нові можливості застосунку, які звичайні шляхи їх навігації, статистика трафіку сайту і т.п.

Інструментальна мова програмування хмарних застосунків

1. *Мова програмування Java*

Java - це об'єктно-орієнтована мова пограмування, яка розроблена і поширюється компанією Sun Microsystems як основний компонент платформи Java. Синтаксис мови багато в чому походить від C та C++. У офіційній реалізації платформи Java програми компілюються у байт-код, який при виконанні інтерпретується віртуальною Java-машиною.
Java пропонує своїм користувачам виключну кросплатформеність. Програма, написана на Java може працювати на будь-якій машині, куди можна встановити Java-машину. Це відкриває багатьом компаніям додаткові можливості, завдяки переходу на відкрите програмне забезпечення, де є підтримка платформи Java у повній мірі.
Основні переваги мови Java:

- Кросплатформеність. Програма може працювати на різних платформах без змін у коді або перекомпіляції.
- Звичний синтаксис. Мова значно запозичила синтаксис з C та C++. Знайомий синтаксис робить розробку проектів на Java більш інтенсивною.
- Java дозволяє викликати функції з сторонніх бібліотек. Якщо не вистачає функціонала, що пропонують стандартні бібліотеки – можна скористатися готовими вільними бібліотеками, які містять необхідний функціонал. Це значно розширює можливості інструменту програмування та дозволяє економити час на вже готових рішеннях. Завдяки цьому розробка проектів на Java більш інтенсивна.
- Платформа сервлетів. Сервлет – це стандартизований API, що дозволяє, використовуючи платформу Java, створювати динамічний контент до веб-сервера. Це дозволяє розвантажити клієнтські робочі термінали та перекласти обчислення на сервер, який має значно більші обчислювальні потужності. Таким чином користувач може отримувати результат обчислень швидше, виконувати на своїй машині більше

завдань без необхідності збільшення власної обчислювальної потужності. В свою чергу, потужний сервер стає дешевшим прямо пропорційно кількості клієнтів, які він обслуговує.

2 Додаткові засоби

XML - це запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Він є спрощеною підмножиною мови розмітки SGML. XML документ складається із текстових знаків, і придатний до читання людиною.

Стандарт XML (англ. *Recommendation*, перше видання від 10 лютого 1998, останнє) визначає набір базових лексичних та синтаксичних правил для побудови мови опису інформації шляхом застосування простих *тегів*. Цей формат достатньо гнучкий для того, щоб бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем (англ. *Schema*), таких як DTD, RELAX NG або XML Schema. Прикладами мов, основаних на XML є: RSS, MathML, GraphML, XHTML, Scalable Vector Graphics, і також XML Schema.[18]

Висновки до лекції

Хмарне середовище відрізняється тим, що воно організоване як рівень програмної абстракції, який є вищим за рівень операційних систем. Через це, з одного боку, досягається проста організація обчислювального процесу на будь-якій обчислювальній платформі, яка є віддаленою, а з іншого боку, через накладні витрати на реалізацію ще одного поміжного рівня абстракції на сьогодні хмарні обчислення не доцільно застосовувати для високопродуктивних обчислень та обробки великих об’ємів даних.

Найбільш поширеною схемою застосування хмарних обчислень є і буде клієнт-серверна схема з клієнтом, реалізованим у веб-браузері та сервері, який запущено у певному вузлі Грід-системи чи комп’ютерного кластеру, який забезпечує подачу html-сторінок до веб-браузеру. Тому хмарні обчислення можна успішно розвивати за допомогою існуючих засобів розробки веб-сторінок та сайтів.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення хмарних обчислень.
2. Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація ІТ-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Назвати етапи впровадження хмарних обчислень.
7. Що стало основою хмарних обчислень?
8. Що являє собою гіпервізорна технологія?

9. Охарактеризуйте архітектуру Hyper-v.
10. Охарактеризуйте особливості розробки застосунків для хмарного середовища.
11. Охарактеризуйте Microsoft System Center Virtual Machine Manager (SCVMM).
12. Що таке Плагіни та веб-платформи?
13. Перерахуйте види послуг, які надаються хмарними системами.
14. Що таке «Інфраструктура як сервіс» (Infrastructure as a Service , або IaaS)?
15. Що таке «Платформа як сервіс» (Plaatform as a Service , PaaS)?
16. Що таке «Програмне забезпечення як сервіс» (Software as a Service , SaaS)?

Література

1.Windows Server 2008 R2 and Windows Server 2008. Режим доступу: www.microsoft.com/windowsserver2008/en/us/2008-dc. aspx.
2. Microsoft" Windows Server' 2012 (Rand Morimoto, Ph.D., MCITP, MVP, Michael Noel, MVP, MCITP, Guy Yardeni, MCITP, CISSP, MVP, Omar Droubi, MCSE, MCTS, Andrew Abbate, MCITP, Chris Amaris, MCITP, MCTS, CISSP, Technical Edit by Tyson Kopczynski, CISSP, GCIH)// SAMS, 800 East 96th Street, Indianapolis, Indiana 46240 USA.
3. J. Schopf; Ten Actions When Super Scheduling // document of Scheduling Working Group, Global Grid Forum. Режим доступу: http://www.ggf.0rg/ d0cuments/GFD.4.pdf, July 2001.
4. Google App Engine. Режим доступу: http://code.google.com/appengine
5. Microsoft Azure. Режим доступу: http://www.microsoft.com/azure
6. Amazon Elastic Compute Cloud. Режим доступу: http://aws.amazon.com/ec2/
7. Apex: Salesforce on-demand programming language and framework. Режим доступу: http://developer. force.com/
8.Dean J. and Ghemawat S. MapReduce: Simplified data processing on large clusters // *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (*OSDI*), San Francisco, CA, - 2004. -p. 137–150.
9.Hadoop. Режим доступу: http://hadoop.apache.org/
10.Amazon Simple Storage Service. Режим доступу: http://aws.amazon.com/s3/
11. Foster I. , Kesselman C.. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl*., -1997. -V.11. - N2. -p.115–128.
12.D. Thain, T. Tannenbaum, Livny M.. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*. -2005. -V.17. N2–4. -p.323–356.
13.Jackson. K. L. An ontology for tactical cloud computing. Режим доступу: http:// kevinljackson.blogspot.com/
14. Hoff C. Christofer hoff blog: Rational survivability. Режим доступу: http://rationalsecurity.typepad.
15 Cloud Computing and Software Services. Theory and Techniques. / S. A. Ahson, M.Ilyas - ed-s.-CRC Press Taylor & Francis Group. -2011. -442.

16. Adobe Flash runtimes. Benefits. Режим доступу: http://www.adobe.com/products/flashruntimes/benefits.html
17. York. J. SaaS Growth Strategy. A Customer Lifecycle Approach. -2013. -16 P. Режим доступу: http://chaotic-flow.com/media/saas-growth-strategy.pdf
18. Harold E. R. XML 1.1 Bible. 3rd Ed. Indianapolis: Wiley Publishing, Inc. - 2004. - 1022 P.

3.3 **Інтерфейс програмування додатків Windows Azure SDK** .

Windows Azure SDK надає розробникам інтерфейс програмування додатків, необхідний для розробки, розгортання та управління масштабованих сервісів в Windows Azure.

В даній лекції ми розглянемо основні можливості Windows Azure SDK.

Мета даної лекції - ознайомитися з комплектом засобів розробки Windows Azure SDK.

Azure Cloud Fabric і служби Azure Storage не підтримують розробку або налагоджувальні операції в хмарі, тому Azure SDK дозволяє робити це локально у вигляді додатків *Development Fabric (DF)* і *Development Storage (DS)*, які встановлює Windows Azure SDK. Разом з SDK також встановлюються колекція додатків прикладів і бібліотеки упакованих класів для полегшення програмування додатків.

Повинні бути встановлені .NET Framework 3.5 SP1 і SQL Express 2005 or 2008, також необхідно включити ASP.NET і WCF HTTP Activation для IIS 7.0 для Windows Server 2008, Windows Vista SP2, or Windows 7 RC або пізніші для встановлення і запуску SDK.

Нотатки до випуску включають інструкції для налаштування цих опцій. Використання SDK не є обов'язковим, тому що є можливість користуватися будь-якими операційними системами і мовами програмування, які підтримують HTTP запити і відповіді. Однак, використання SDK інтерфейсів прикладного програмування .NET і бібліотек для додатків і сховищ дозволяє найбільш просто працювати з HTTP безпосередньо.

Після того, як Ви встановили Azure SDK, Ви повинні завантажити та встановити інструменти Windows Azure для Visual Studio для додавання шаблонів проектів *Web Cloud Sevice, Worker Cloud Service, Web and Worker Cloud Service Workflow Service*. Ви можете завантажити поточну версію Windows Azure SDK і Windows Azure Tools для Visual Studio з головної сторінки Windows Azure за посиланням *www.microsoft.com/azure/windowsazure.mspx*.

Після встановлення Windows Azure Tools в Visual Studio з'являються шаблони *Cloud Service* в діалозі створення нового проекту (рис.3.3.1.). При виборі вузла Cloud Service відкриваються New Cloud Service, який дозволяє додати *ASP. NET Web Roles, Worker Roles or CGI Web Roles* для нового проекту. Windows Azure SDK дозволяє додати більше, ніж одну роль для кожного типу

Cloud Service. Кожна роль використовує окремий екземпляр Windows Azure CPU.

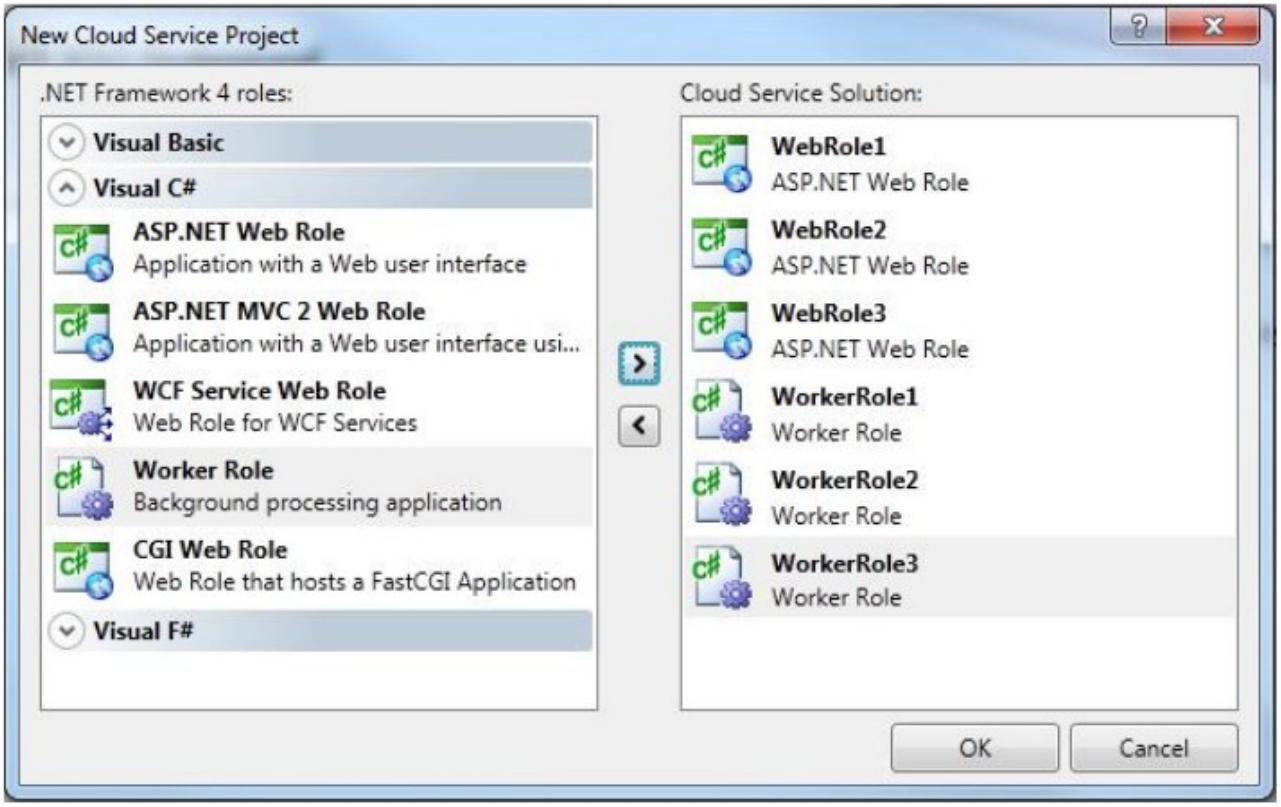


Рис.3.3.1. Створення нового проекту Cloud Service в Visual Studio

Додавши зазначені ролі і натиснувши ОК, відкриється нове рішення з проектами *WebRole і Worker-Role* в Solution Explorer як показано на рис.3.3.2.

Вузол *Roles* містить елементи *WebRole*, які вказують на кожну *WebRole*, що забезпечує користувацький інтерфейс ASP.NET для додатку і кожен *WorkerRole* для обчислювальних операцій, які не вимагають користувацького інтерфейсу, або використовують сторінку *ASP. NET WebRole* замість цього.

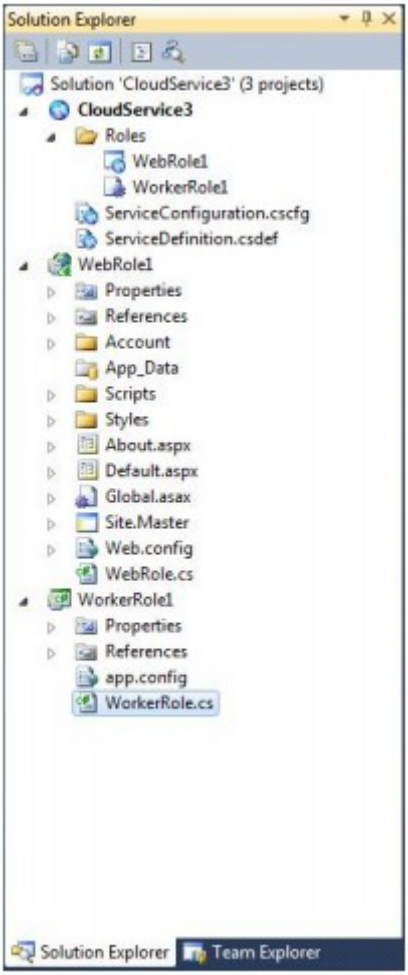


Рис.3.3.2 Solution Explorer Cloud Service

Залежно від типу Cloud Service проекти включають простір імен *Microsoft. ServiceHosting. ServiceRuntime*, який містить класи, зазначені в таблиці нижче.

Клас	Опис
RoleEntryPoint	Забезпечує методи для керування ініціалізацією, запуском і зупинкою методів сервісу і використовується для мониторингу стану сервісу
RoleException	Повідомляє про помилки, коли виникають недопустимі операції в середині ролі
RoleManager	Забезпечує методи журналювання повідомлень і надходячих застережень, витягує налаштування конфігурації сервісу та повертає місцезнаходження ресурсу
RoleStatus	Інформує про поточний статус ролі: Healthy, NonExistent, Started, Starting, Stopped, Stopping чи Unhealthy

Проекти, які використовують шаблон *WebRole* визначають веб сторінку ASP.NET Default.aspx як початкову точку для призначеного для користувача інтерфейсу додатку в хмарі.

Цей сервіс об'єднує бібліотеку класу *Common* з додатку-зразку *HelloFabric* для сприяння журналювання проблем додатку. Журнали додатку - це практичні засоби налагодження додатків, запущених в Cloud Fabric. Для

читання журналів Ви повинні скопіювати їх в Blob- масив, використовуючи інструментарій порталу. Зразок проекту *StorageClient* включає бібліотеку класу *StorageClient*, яка забезпечує в поєднанні з бібліотекою .NET Client для сервісу даних ADO.NET, інтерфейсний клас Microsoft .NET для HTTP операцій над *Azure Blob, Queue і Table Storage* сервісами. Цей проект також включає консольний додаток, який дозволяє тестувати можливості бібліотеки. Консольний додаток C # запускається в *Development Fabric* с *Development Storage*.

При встановленні Windows Azure SDK не встановлюються зразки додатків, які включені в *Program Files \ Microsoft Windows Azure SDK \ v1.0 \ samples.zip.* Встановіть зразки, розпакував samples.zip в директорію, де Ви маєте права на запис. Нижче можна знайти опис деяких зразків додатків.

Для запуску прикладу *CloudDrive* необхідний PowerShell.

Директорія, в яку було вилучено вміст архіву samples.zip, також містить наступні три пакетних файли (cmd), які можна запустити з командного рядка:

• *buildall.cmd* будує всі зразки проектів без використання Visual Studio:
• *createtables.cmd* викликає *buildall.cmd* і створює базу даних і таблиці, необхідні для зразків, які використовують Table Storage.
• *rundevstore.cmd* викликає *createtables cmd* і запускає розробку сховища, розміщуючи його в базі даних, створеної *createtables.cmd*.
До складу Development Fabric входять наступні виконувані файли: *DFAgent. exe*, *DFLoadBalancer. exe, DFMonitor. exe і DFService. exe*, які за замовчувнням встановлюються програмою установки Azure SDK в каталог *\ Program Files \ Windows Azure SDK \ v1.0 \ bin \ devfabric*. Після запуску Development Fabric в диспетчері завдань Ви можете побачити ці чотири процеси. Зробити це можна виконавши:
• Виберіть *Програми \ Windows Azure SDK \ Development Fabric* для запуску служби Development Fabric і його користувацького інтерфейсу *DFUI. exe*
• Правий клік мишею по значку Development Fabric в області повідомлень панелі задач і вибрати запуск служби Development Fabric (рис.3.3.3)
• Скомпілювати та запустити пдодаток Azure в Visual Studio.

Рис.3.3.3 Повідомлення, які відображаються при натисканні правою кнопкою миші по значку Development Fabric в області повідомлень панелі завдань.

Рис.3.3.4 показує користувацький інтерфейс DFUI. Коли Ви запускаєте або зупиняєте налагодження, відповідні додатки з'являються, або пропадають з користувацького інтерфейсу DFUI.

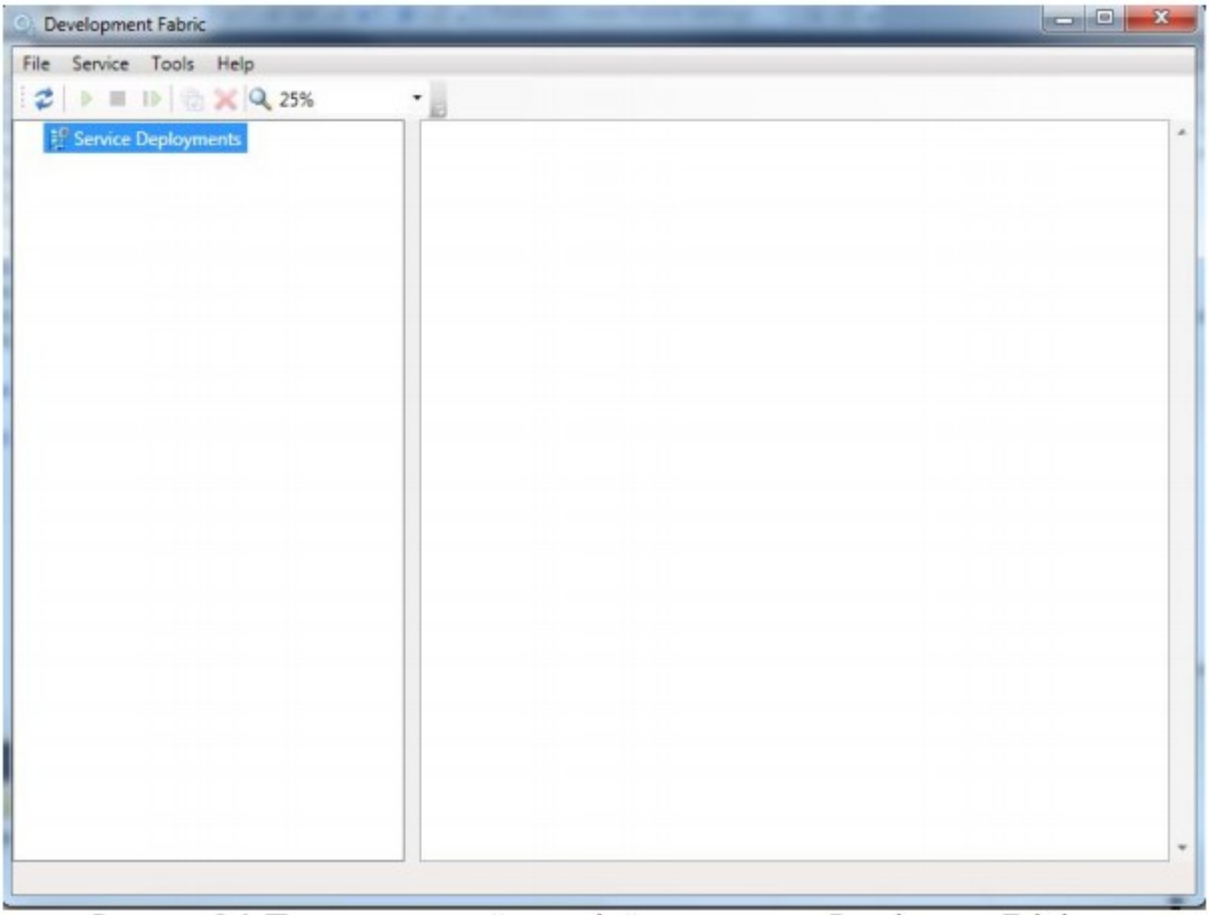


Рис.3.3.4. Користувацький інтерфейс додатку Development Fabric.

Платформа Windows Azure підтримує три типи масштабованих сховищ:

• Неструктуровані дані (blob)
• Структуровані дані (таблиці)
• Сполучення між додатками і сервісами (черги)

Запускаючи *rundevstore.exe,* або збираючи і запускаючи користувацький код Azure в Visual Studio, запускаються всі три сервіси, навіть якщо Ваш проект вимагає тільки один сервіс і відображається в користувацькому інтерфейсі Development Storage.

Для захисту від втрати даних, хмара Azure зберігає блоби, таблиці і черги в мінімум трьох роздільних контейнерах в одному центрі обробки даних. Інструмент геолокації Azure дозволяє дублювати дані в декількох центрах обробки даних Microsoft для зменшення наслідків відновлення після катастроф і для підвищення продуктивності в специфічних географічних регіонах.

Додатки Azure, які Ви запускаєте в Development Framework, можуть мати доступ до локальних даних в Development Storage, або до даних, завантажених в хмару Azure. Додаток звертається до певного порту і даних, розташованих в визначених місцях в файлі конфігурації проекту *ServiceConfiguration. cscfg*.

Файл конфігурації проекту *Azure ServiceDefinition.csdef* визначає стандартні точки входу і налаштування конфігурації, які зберігаються в файлі

ServiceConfiguration.cscfg. У роздруківці 1 показано вміст файлу *ServiceDefinition.csdef* за замовчуванням, коли Ви створюєте проект Azure, використовуючи один з стандартних шаблонів Windows Azure Tools для Visual Studio (відзначені важливі значення).

Роздруківка 1 Вміст файлу ServiceDefinition.csdef

```
<ServiceDefinition name="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
<WebRole name="WebRole">
<InputEndpoints>
<!-- Must use port 80 for http and port 443 for https
when running in the cloud -->
<InputEndpoint name="HttpIn" protocol="http" port="80" />
</InputEndpoints>
<ConfigurationSettings>
<Setting name="AccountName"/>
<Setting name="AccountSharedKey"/>
<Setting name="BlobStorageEndpoint"/>
<Setting name="QueueStorageEndpoint"/>
<Setting name="TableStorageEndpoint"/>
</ConfigurationSettings>
</WebRole>
</ServiceDefinition>
```

Значення *InputEndpoint* використовується тількио для сховищ в хмарі.

Роздруківка 2 показує вміст файлу *ServiceConfiguration. cscfg* для веб додатку *SampleWebCloudService* з конфігурацією по замовчуванню для Development Storage (виділено):

Роздруківка 2 Вміст файлу ServiceConfiguration.cscfg

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration
">
<Role name="WebRole">
<Instances count="1"/>
<ConfigurationSettings>
<Setting name="AccountName" value="devstoreaccount1"/>
<Setting name="AccountSharedKey"
value="Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ
1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==/>
<Setting name="BlobStorageEndpoint" value="http://127.0.0.1:10000/"/>
<Setting name="QueueStorageEndpoint" value="http://127.0.0.1:10001/"/>
<Setting name="TableStorageEndpoint" value="http://127.0.0.1:10002/"/>
<!--<Setting name="AccountName" value="oakleaf"/>
<Setting name="AccountSharedKey" value="3elV1ndd . . . Coc0AMQA==" />
```

```
<Setting name="BlobStorageEndpoint"
value="http://blob.core.windows.net" />
<Setting name="QueueStorageEndpoint"
value="http://queue.core.windows.net" />
<Setting name="TableStorageEndpoint"
value="http://table.core.windows.net" />
-->
</ConfigurationSettings>
</Role>
</ServiceConfiguration>
```

Описи елементів файлу конфігурації ServiceConfiguration.csfg:

• Instances count - кількість примірників Вашого застосування, яке буде створено в хмарі, коли Ви розгорнете його.
• AccountName- ім'я, асоційоване з Вашим Hosted Service, з яким Ви створювали обліковий запис, для Development Storage це devstoreaccount1.
• AccountSharedKey шифрує кілька елементів в HTTP запиті.
• BlobStorageEndpoint- це публічний постійний Universal Resource Identifier (URI). Для Developer Storage це адреса інтерфейсу комп'ютера loopback (localhost = 127.0.0.1) з TCP портом за замовчуванням 10000.
• QueueStorageEndpoint для сховища в хмарі це публічний постійний URI. Для Developer Storage це адреса інтерфейсу комп'ютера loopback з TCP портом за замовчуванням 10001.
• TableStorageEndpoint публічний постійний Universal Resource Identifier (URI). Для Developer Storage це адреса інтерфейсу комп'ютера loopback з TCP портом за замовчуванням 10002.

Ви можете налаштувати власні номери TCP портів, якщо при використанні стандартних налаштувань виникає конфлікт з поточною конфігурацією.

В даній лекції ми отримали початкові відомості про роботу з Windows Azure SDK. Розглянули процедуру створення Cloud Service, користувацький інтерфейс Development Fabric.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення хмарних обчислень.
2. Перерахуйте особливості програмування в «хмарах».
3. Що таке Windows Azure SDK?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Які елементи містить вузел Roles?
7. Які типи масштабованих сховищ підтримує Платформа Windows Azure?
8. Охарактеризуйте користувацький інтерфейс Development Fabric.
9. Наведіть процедуру створення Cloud Service.
10. Перерахуйте види послуг, які надаються хмарними системами.

Літеретура

1. http://www.microsoft.com/windowsazure/
2. http://www.microsoft.com/faculty

3.4 **Платформа** ***Windows Azure***

Платформа корпорації Microsoft Windows Azure (спочатку відома під назвою Azure Services Platform) - це група «хмарних» технологій, кожна з яких надає певний набір служб для розробників додатків. На рис.3.4.1 показано, що платформа Windows Azure може бути використана як додатками, які виконуються в «хмарі», так і додатками, які працюють на локальних комп'ютерах

Рис.3.4.1 Платформа Windows Azure підтримує додатки, дані та інфраструктуру, які знаходяться в «хмарі».

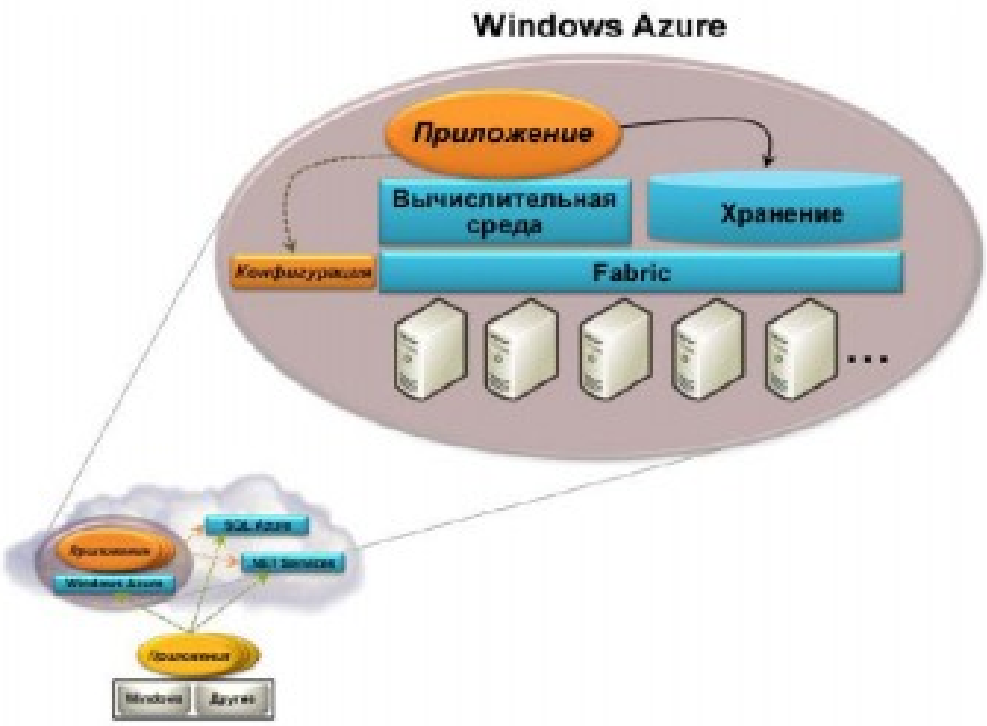
Платформа Windows Azure складається з наступних компонентів:

• Windows Azure. Забезпечує сердовище на базі Windows для виконання додатків і зберігання даних на серверах в центрах обробки даних корпорації Microsoft.

• SQL Azure. Забезпечує служби даних в «хмарі» на базі SQL Server.

• .NET Services. Забезпечують розподілену інфраструктуру для «хмарних» додатків і локальних додатків.

Кожен компонент платформи Windows Azure грає власну роль. На високому рівні зрозуміти Windows Azure досить легко. Це платформа для виконання додатків Windows і зберігання їх даних в Інтернеті («хмарі»). На рис.3.4.2 показані її основні компоненти.



Нис.3.4.2.. Windows Azure надає «хмарні» додатків служби для обчислення і зберігання на базі Windows

Як показано на рис.3.4.2, Windows Azure виконується на великій кількості комп'ютерів, розташованих в центрах обробки даних корпорації Microsoft, і доступна через Інтернет. Загальна структура підключення Fabric Windows Azure з'єднує безліч обчислювальних потужностей в єдине ціле. Служби Windows Azure для обчислення і зберігання побудовані на основі цієї структури. Спочатку корпорація Microsoft дозволила виконувати на Windows Azure тільки додатки, розроблені на платформі .NET Framework. Сьогодні Windows Azure також підтримує некерований код, дозволяючи розробникам виконувати додатки, які розроблені і не на базі .NET Framework. У будь-якому випадку такі додатки написані на звичайних мовах Windows - C #, Visual Basic, C ++ і інших - за допомогою Visual Studio 2008 або інших засобів розробки. Як додатки Windows Azure, так і локальні додатки можуть отримувати доступ до служби сховища Windows Azure за допомогою підходу RESTful. Однак Microsoft SQL Server не є базовим сховищем даних. Фактично сховище Windows Azure не відноситься до реляційних систем і мова його запитів не SQL. Оскільки воно призначено для підтримки додатків на базі Windows Azure, то забезпечує більш прості і масштабовані способи зберігання. Отже, воно дозволяє зберігати великі двійкові об'єкти (binary large object - blob), забезпечує створення черг для взаємодії між компонентами додатків і навіть щось на зразок таблиць з простою мовою запитів. (Для тих додатків Windows Azure, яким потрібно звичайне реляционное сховище, платформа Windows Azure надає базу даних SQL Azure. У Windows Azure кожен додаток має конфігураційний файл. Змінюючи інформацію в цьому файлі вручну або за допомогою програми, власник додатку може контролювати різні аспекти його поведінки , такі як налаштування кількості примірників, які повинні виконуватися на платформі Windows Azure. Структура Fabric платформи

Windows Azure спостерігає за тим, щоб додаток підтримувалося в необхідному стані.

Щоб дозволити своїм замовникам створювати, налаштовувати додатки та спостерігати за ними, Windows Azure надає портал, доступний за допомогою браузера. Замовник надає Windows Live ID, а потім вирішує, створювати йому обліковий запис розміщення для виконання додатків, обліковий запис зберігання для зберігання даних або і ту і іншу. Windows Azure - це загальна платформа, яку можна використовувати в різних сценаріях.

Виконання програм в «хмарі» - один з найважливіших аспектів «хмарних» обчислень. За допомогою Windows Azure корпорація Microsoft забезпечує як платформу для виконання додатків, так і спосіб зберігання даних. У міру того, як зростає інтерес до «хмарних» обчислень, очікується створення ще більшої кількості програм Windows для цієї нової області.

Один з найбільш привабливих способів використання серверів, доступних через Інтернет - це обробка даних. Мета SQL Azure - вирішити цю проблему, пропонуючи набір веб-служб для зберігання найрізноманітнішої інформації і роботи з нею.

База даних SQL Azure Database (раніше відома під назвою SQL DataServices) забезпечує систему управління базами даних (СКБД) в Інтернеті. Ця технологія дозволяє локальним і веб-додаткам зберігати реляційні та інші типи даних на серверах Microsoft в центрах обробки даних Microsoft. База даних SQL Azure розроблена на основі Microsoft SQL Server. Другий компонент SQL Azure був заявлений під назвою Huron Data Sync. Ця технологія, розроблена на основі Microsoft Sync Framework і SQL Azure Database, дозволяє синхронізувати реляційні дані в різних локальних СУБД. Власники даних можуть визначати, що саме має синхронізуватися, як повинні вирішуватися конфлікти і багато іншого.

Виконання програм і зберігання даних в Інтернеті відносяться до важливих аспектів обчислювального мережевого середовища. Однак вони далеко не вичерпують її можливості. Інша можливість полягає в забезпеченні інфраструктури служб на базі «хмари», які можуть використовуватися локальними додатками або веб-додатками. Прикладом є служби .NET Services. Спочатку відомі як BizTalk Services, служби .NET Services пропонують функції для вирішення спільних проблем інфраструктури при створенні розподілених додатків. На рис.3.4.3 показані їх основні компоненти.

Рис.3.4.3. Служби .NET Services забезпечують інфраструктуру в «хмарі», яка може бути використана для веб-додатків і локальних додатків.

Так само як у випадку Windows Azure надається портал, доступний з допомогою браузеру, щоб дати замовникам можливість використовувати служби .NET Services за допомогою Windows Live ID. Мета корпорації Microsoft, що досягається за допомогою .NET Services, абсолютно очевидна: забезпечити корисну «хмарну» інфраструктуру для розподілених додатків.

Архітектура Windows Azure Platform

Платформа Windows Azure – це модель Платформа як сервіс, яка передбачає запуск додатків на серверах і пов'язаної мережевої інфраструктури, розміщеної в центрах обробки даних Microsoft і має доступ в Інтернет. Платформа складається з матшабованої «хмарної" операційної системи, фабрики зберігання даних і пов'язаних сервісів доставки через фізичні або логічні (віртуалізація) екземпляри Windows Server 2008. Комплект засобів розробки Windows Azure (SDK) забезпечує розробку версії «хмарних сервісів», также добре, як інструменти і інтерфейси прикладного програмування (API), необхідні для розробки, розгортання та керування масштабованих сервісів в Windows Azure, включаючи шаблони додатків Azure для Visual Studio 2008 і 2010.

На рис.3.4.4 зображені компоненти «хмарної» платформи і компоненти розробника. Згідно Microsoft, при використанні Azure Ви отримуєте:

• Адаптацію існуючих додатків для роботи з веб-сервісами;

• Побудова, зміна і розподіл додатків в мережі з мінімальними локальними ресурсами;

• Виконання послуги, таких як зберігання великих обсягів даних, пакетна обробка даних, обчислення великих обсягів даних і так далі;

• Створення, тестування, налагодження і розподіл веб- сервісів швидко і недорого;
• Зниження вартості і ризиків побудови і поширення місцевих ресурсів;
• Зниження витрат і зусиль на IT управління;

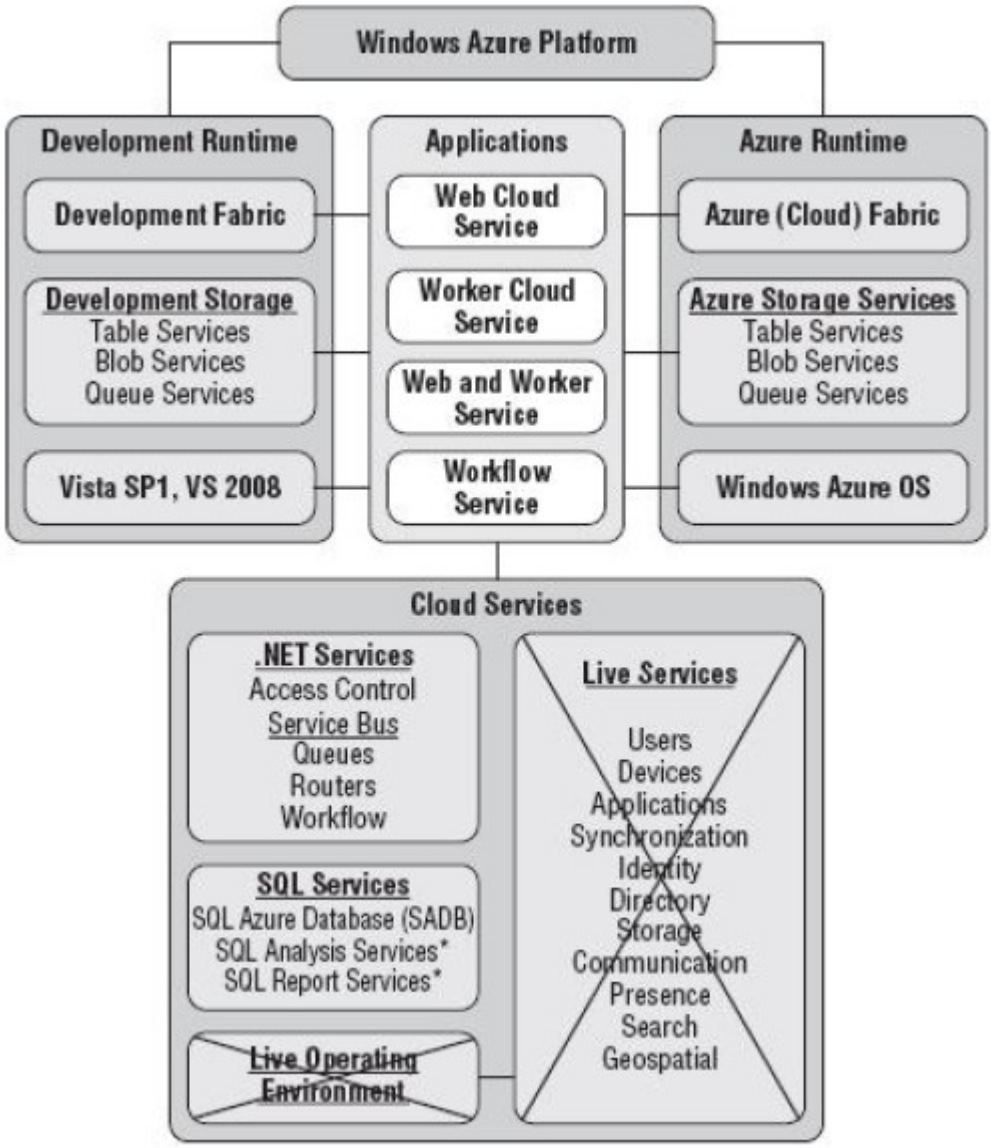


Рис.3.4.4. Компоненти платформи Windows Azure і комплекту засобів розробки.

Вихідною точкою входу для розробників Azure для розміщення ASP.NET додатків в хмару є портал Windows Azure за адресою https://windows.azure.com/ Cloud/Provisioning/Default.aspx. Портал вимагає входу з використанням Windows Live ID.

Можна вибрати центри обробки даних для розташування Hosted Services і Storage Accounts. Наприклад, USA-Northwest (Quincy, WA) і USA-Southeast (San Antonio, TX.). Ви можете додати набори Hosted Services і Storage Accounts в групу, щоб гарантувати, що сервіси та сховище розташовуються в одному і тому ж центрі обробки даних, для того, щоб збільшити продуктивність.

Якщо ви хочете розробляти веб-сайти з підтримкою Live Framework, або Mesh (програмний комплекс для синхронізації даних в кросплатформенних середовищах, розроблений компанією Microsoft) веб-додатків, необхідно запросити токен Live Framework по email meshctpe@microsoft.com. Після того, як ви отримаєте Live Framework токен, Ви можете завантажити і встановити поточні версії Live Framework SDK і Live Framework Tools для Visual Studio по посиланнях зазначених на сторінці http://dev.live.com/liveframework/sdk/. Ви повинні оплатити Live Framework токен для того, щоб завантажити Live SDK і додаткові інструменти. Немає необхідності використовувати обліковий запис Windows Azure для тестування Azure Hosted Services and Storage Services, тому що платформа розробки Azure емулює «хмарні» сервіси Azure на вашому комп'ютері.

Windows Azure Storage

Сховище Windows Azure Storage забезпечує розробникам можливість зберігання даних в хмарі. Додаток може виконувати доступ до своїх даних в будь-який момент часу з будь-якої точки планети, зберігати будь-який обсяг даних і як завгодно довго. При цьому дані гарантовано не будуть пошкоджені і втрачені. Windows Azure Storage пропонує багатий набір абстракцій даних:

• Windows Azure Table - забезпечує структуроване сховище станів сервісу.
• Windows Azure Blob - забезпечує сховище великих елементів даних.
• Windows Azure Queue - забезпечує диспетчеризацію асинхронних завдань для реалізації обміну даними між сервісами.

Azure Table Services

Windows Azure Table - структуроване сховище, котре підтримує високомасштабуємі таблиці в хмарі, які можуть містити мільярди сутностей і терабайти даних. У міру збільшення трафіку, система буде ефективно масштабуватися, автоматично підключаючи тисячі серверів. Структуроване сховище реалізовано у вигляді таблиць (Tables), в яких розташовуються сутності (Entities), що містять ряд іменованих властивостей (Properties).
Ось деякі з основних характеристик Windows Azure Table:

• Підтримка LINQ, ADO .NET Data Services і REST.
• Контроль типів під час компіляції при використанні клієнтської бібліотеки ADO .NET Data Services.
• Багатий набір типів даних для значень властивостей.
• Підтримка необмеженої кількості таблиць і сутностей без обмеження розмірів таблиць.
• Підтримка цілісності для кожної сутності.
• Нежорстке блокування при оновленнях і видаленнях.
• Для запитів, виконання яких вимагає тривалого періоду часу, або запитів, перерваних після завершення часу очікування, повертаються часткові результати і маркер продовження

Модель даних таблиці Windows Azure Table:

Облікова запись сховища (Storage Account) - для доступу до Windows Azure Storage додаток має використовувати дійсну облікову запись. Нову облікову запись можна створити через веб-інтерфейс порталу Windows Azure. Як тільки облікову запись створено, користувач отримує 256-розрядний секретний ключ, який згодом використовується для аутентифікації запитів цього користувача до системи зберігання. Зокрема, за допомогою цього секретного ключа створюється підпис HMAC SHA256 для запиту. Цей підпис передається з кожним запитом даного користувача для забезпечення аутентифікації. Ім'я облікового запису входить до складу імені хоста в URL.

Для доступу до таблиць використовується наступний формат імені хоста: <імяОбліковийЗапис> .table.core.windows.net.

• Таблиця (Table) - містить набір сутностей. Область дії імен таблиць обмежена обліковим записом. Додаток може створювати безліч таблиць в рамках облікового запису сховища.

• Сутність (рядок) (Entity (Row)) - Сутності (сутність є аналогом «Рядка») - це основні елементи даних, що зберігаються в таблиці. Сутність включає набір властивостей. У кожній таблиці є дві властивості, які утворюють унікальний ключ для сутності.

• Властивість (стовпець) (Property (Column)) - являє окреме значення сутності. Імена властивостей чутливі до регістру. Для значень властивостей підтримується багатий набір типів.

• Ключ секції (PartitionKey) - Перша властивість ключа кожної таблиці. Ця система використовує даний ключ для автоматичного розподілу сутностей таблиці по безлічі вузлів зберігання.

• Ключ рядки (RowKey) - Друга властивість ключа таблиці. Це унікальний ID сутності в рамках секції. PartitionKey в поєднанні з RowKey унікально ідентифікує сутність в таблиці.

• Тимчасова мітка (Timestamp) - Кожна сутність має версію, яка зберігається системою.

• Секція (Partition) - Набір сутностей в таблиці з однаковим значенням ключа секції.

• Порядок сортування (Sort Order) - Для CTP-версії надається всього один індекс, в якому всі сутності сортовані за PartitionKey і потім по RowKey.

Це означає, що запити із зазначенням цих ключів будуть більш ефективними, і всі повертаємі результати будуть сортовані за PartitionKey

Таблиця має гнучку схему. Windows Azure Table відстежує ім'я і типізоване значення кожної властивості кожної сутності. Додаток може моделювати фіксовану схему на стороні клієнта, забезпечуючи однаковий набір властивостей для всіх створюваних сутностей.

Деякі додаткові відомості про сутності:

• Сутність може мати до 255 властивостей, включаючи обов'язкові системні властивості: PartitionKey, RowKey і Timestamp. Імена всіх інших властивостей сутностей визначаються додатком.

• Властивості PartitionKey і RowKey строкового типу.

• Властивість Timestamp є доступною лише для читання обслуговуємою системою властивістю, яку слід розглядати як непрозору властивість.
• Відсутність фіксованої схеми - Windows Azure Table не зберігає ніякої схеми, тому всі властивості зберігаються як пари <ім'я, типізоване значення>. Це означає, що властивості сутностей однієї таблиці можуть сильно відрізнятися. В таблиці навіть може бути дві сутності, властивості яких мають однакові імена, але різні типи значень.
• Сумарний обсяг всіх даних сутності не може перевищувати 1 МБ. Сюди входить розмір імен властивостей, а також розмір значень властивостей або їх типів, включаючи і дві обов'язкові властивості ключів (PartitionKey і RowKey).
• Підтримуються типи Binary, Bool, DateTime, Double, GUID, Int, Int64, String.
Windows Azure Table забезпечує можливість масштабування таблиць до тисяч вузлів зберігання через розподіл сутностей в таблиці. При розподілі сутностей бажано забезпечити, щоб сутності, що входять в одну множину, розташовувалися в одному вузлі зберігання. Додаток формує ці множини відповідно значенням властивості PartitionKey сутностей. Додаткам повинне бути відоме робоче навантаження кожної окремо взятої секції. Для забезпечення бажаних результатів тестування має моделювати максимальне робоче навантаження.

На рис.3.4.5 представлена таблиця, яка містить множину версій документів. Кожна сутність даної таблиці відповідає певній версії певного документу. У цьому прикладі ключем секції таблиці є ім'я документу, ключем рядку - номер версії. Ім'я документа і версія унікально ідентифікують кожну сутність таблиці. В даному прикладі секцію утворюють всі версії одного документу.

Partition Key Document Name	Row Key Version	Property 3 Modification Time	.....	Property N Description	
Examples Doc	V1.0	8/2/2007	.....	Committed version	Partition 1
Examples Doc	V2.0.1	9/28/2007		Alice's working version	
FAQ Doc	V1.0	5/2/2007		Committed version	
FAQ Doc	V1.0.1	7/6/2007		Alice's working version	Partition 2
FAQ Doc	V1.0.2	8/1/2007		Sally's working version	

рис.3.4.5 Приклади секцій

Система відстежує характер використання секцій і автоматично рівномірно розподіляє ці секції по всіх вузлах зберігання. Це дозволяє системі і додатку масштабуватися відповідно до кількості запитів до таблиці. Тобто, якщо деякі секції запитуються більше інших, система автоматично рознесе їх на кілька вузлів зберігання, таким чином, розподіляючи трафік між множиною серверів. Однак секція, тобто всі сутності, що мають однаковий ключ секції, будуть обслуговуватися як один вузол. Але навіть незважаючи на це, обсяг даних в

рамках секції не обмежений ємністю сховища одного вузла зберігання. Сутності однієї секції зберігаються разом. Це забезпечує найбільш ефективну обробку запитів до секції. Більш того, в цьому випадку додаток може використовувати всі переваги ефективного кешування і інших оптимізацій продуктивності, які забезпечуються розташуванням даних в секції.
У наведених нижче прикладах описуються операції з таблицею «Blogs». У цій таблиці зберігаються блоги для додатку MicroBlogging. У додатку MicroBlogging є дві таблиці: Channels (Канали) і Blogs (Блоги). Є список каналів, блоги публікуються в певному каналі. Користувачі підписуються на канали і щодня отримують нові блоги цих каналів.
В даному прикладі розглянемо тільки таблицю Blogs і наведемо приклади наступних операцій з нею:

1. Опис схеми таблиці
2. Створення таблиці
3. Вставка блогу в таблицю
4. Отримання списку блогів з таблиці
5. Оновлення блогу в таблиці
6. Видалення блогу з таблиці

Схема таблиці описується як C # -клас. Таку модель використовує ADO.NET Data Services. Схема відома тільки клієнтському додатку і спрощує доступ до даних. Сервер схему не застосовує.

Розглянемо опис сутностей Blog, які зберігаються в таблиці Blogs. Кожна сутність блогу містить наступні дані:

1. Назва каналу (ChannelName) - канал, в якому розміщується блог.
2. Дата розміщення.
3. Текст (Text) - вміст тіла блогу.
4. Рейтинг (Rating) - популярність цього блогу.

Зверніть увагу, що для таблиці визначено PartitionKey, яка представляє ім'я каналу, частиною якого є блог, і в якості RowKey використовується дата розміщення блогу. PartitionKey і RowKey - ключі таблиці Blogs, вони оголошуються за допомогою атрибута класу DataServiceKey (Ключ сервісу даних). Тобто таблиця Blogs секціонована по іменах каналів (ChannelName). Це дозволяє додатку ефективно витягати самі недавні блоги каналу, на які підписаний користувач. Крім ключів, в якості властивостей оголошені характерні для користувача атрибути. Всі властивості мають відкриті (public) методи зчитування і присвоєння значення і зберігаються в таблиці Windows Azure Table. Отже, в прикладі нижче:

• Text і Rating зберігаються для екземпляра сутності в таблиці Azure.

• RatingAsString немає, тому що для нього не визначено метод присвоєння значення.

• Id не зберігається, тому що методи доступу не public.

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Blog
```

```
{
// ChannelName
public string PartitionKey { get; set; }
// PostedDate
public string RowKey { get; set; }
// визначаємі користувачем властивості
public string Text { get; set; }
public int Rating { get; set; }
public string RatingAsString { get; }
protected string Id { get; set; }
}
```

Далі розглянемо, як створити таблицю Blogs для облікового запису сховища.

Створення таблиці аналогічно створенню сутності в основній таблиці «Tables». Ця основна таблиця визначена для кожного облікового запису сховища, і ім'я кожної таблиці, використовуваної обліковим записом зберігання, повинно бути зареєстровано в основній таблиці. Опис класу основної таблиці наведено нижче, де властивість TableName (Ім'я таблиці) представляє ім'я створюваної таблиці.

```
[DataServiceKey("TableName")]
public class TableStorageTable
{
public string TableName { get; set; }
}
```

Фактично створення таблиці відбувається наступним чином:

```
// Uri сервиса: “http://<Account>.table.core.windows.net/”
DataServiceContext context = new DataServiceContext(serviceUri);
TableStorageTable table = new TableStorageTable("Blogs");
// Створюємо нову  таблицю, добавляючи нову  сущність
// в основну  таблицю "Tables"
context.AddObject("Tables", table);
// результатом визову SaveChanges являеться відгук сервера
DataServiceResponse response = context.SaveChanges();
```

serviceUri – це uri сервісу таблиці, http://<тут вказується ім’я облікового запису>.table.core.windows.net/. DataServiceContext (Контекст сервісу даних) – один із основних класів сервісу даних ADO.NET, представляє контекст часу виконання для сервісу. Він забезпечує API для вставки, оновлення, видалення та запиту сутностей за допомогою яких LINQ, або RESTful URI і зберігає стан на стороні клієнта.

Розглянемо вставку елемента Blog. Щоб вставити сутність, додаток має виконати наступне.

1. Створити новий C #-об'єкт і задати всі властивості.

2. Створити екземпляр DataServiceContext, який представляє підключення до серверу в сервісі даних ADO .NET для вашого профілю сховища.

3. Додати C #-об'єкт в контекст.

4. Викликати метод SaveChanges (Зберегти зміни) об'єкту DataServiceContext для відправки запиту серверу. Це забезпечує відправку на сервер HTTP- запиту з сутністю в XML-форматі ATOM.

Далі представлені приклади коду для перерахованих вище операцій:

```
Blog blog = new Blog {
PartitionKey = "Channel9", // ChannelName
RowKey = DateTime.UtcNow.ToString(), // PostedDate
Text = "Hello",
Rating = 3
};
serviceUri = new Uri("http://<account>.table.core.windows.net");
var context = new DataServiceContext(serviceUri);
context.AddObject("Blogs", blog);
DataServiceContext response = context.SaveChanges();
```

Запит сутностей виконується за допомогою вбудованої в C # мови запитів LINQ (Language Integrated Query). В даному прикладі отримаємо всі блоги, рейтинг яких дорівнює 3.

При обробці запиту (наприклад, за допомогою виразу foreach), він передається на сервер. Сервер відправляє результати в XML-форматі ATOM. Клієнтська бібліотека ADO .NET Data Services десеріалізує результати в C # -об'єкти, після чого вони можуть використовуватися додатком.

```
var serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext context = new DataServiceContext(serviceUri);
// LINQ-запит з використанням DataServiceContext для вибору
// із таблиці Blogs всіх сутностей блогів, для которих rating = 3
var blogs =
from blog in context.CreateQuery<Blog>("Blogs")
where blogs.Rating == 3
select blog;
// запит відправляється на сервер і виконується
foreach (Blog blog in blogs) { }
```

Оновлення суті виконується наступним чином.

1. Створюється DataContext (Контекст даних), властивості MergeOption (Варіант об'єднання) якого задається значення OverwriteChanges (Перезапис

змін) або PreserveChanges (Збереження змін) . Це забезпечує правильну обробку ETag для кожного витягнутого об'єкту.

2. За допомогою LINQ DataContext отримує сутність, яка буде оновлюватися. Витяг її з серверу гарантує оновлення ETag в сутностях, що відслідковуються контекстом, і те, що при подальших оновлення та видаленях в заголовку if-match буде використовуватися оновлений ETag. Міняємо C#-об'єкт, який представляє сутність.

3. Повертаємо C #-об'єкт в той же DataContext для поновлення. Використання того ж DataContext гарантує автоматичне повторне використання ETag, отриманого раніше для цього об'єкта.

4. Викликаємо метод SaveChanges для відправки запиту на сервер.

```
Blog blog =
(from blog in context.CreateQuery<Blog>("Blogs")
where blog.PartitionKey == "Channel9"
&& blog.RowKey == "Oct-29"
select blog).FirstOrDefault();
blog.Text = "Hi there";
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Видалення Blog

Видалення сутності аналогічно її оновленню. Для цього витягаємо сутність за допомогою DataServiceContext і викликаємо для вмісту замість методу UpdateObject метод DeleteObject (Видалити об'єкт).

```
// Отримуємо об'єкт Blog для ("Channel9", "Oct-29")
context.DeleteObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Розглянемо рекомендації по роботі з DataServiceContext:

• Об'єкт DataServiceContext не забезпечує безпеку потоків, тому він не може використовуватися спільно різними потоками, а також має нетривалий час існування.

• DataServiceContext не є об'єктом з тривалим часом життя. Замість того, щоб використовувати один DataServiceContext протягом всього життя потоку, рекомендується створювати об'єкт DataServiceContext кожен раз, коли виникає необхідність виконати ряд транзакцій з WindowsAzureTable, і потім видаляти цей об'єкт.

• Якщо для всіх вставок / оновлень / вилучень використовується один екземпляр DataServiceContext і виникає збій при виконанні SaveChanges, відомості про операції, що дала збій, зберігаються в DataServiceContext. При наступному виклику SaveChanges спроба виконати цю операцію повторюється.

• DataServiceContext має властивість MergeOption, яка використовується для управління тим, як DataServiceContext обробляє відслідковувані сутності. Можливі значення:

o AppendOnly (Тільки додавання): Це значення за замовчуванням, при використанні якого DataServiceContext не завантажує екземпляр сутності з сервера, якщо він вже є в його кеші.

o OverwriteChanges: DataServiceContext завжди завантажує екземпляр сутності з сервера і перезаписує попередній варіант сутності, тобто забезпечує відповідність екземпляра сутності її поточному стану.

o PreserveChanges: Якщо екземпляр сутності існує в DataServiceContext, він не завантажується з постійного сховища. Всі зміни, що вносяться до властивості об'єктів в DataServiceContext, зберігаються, але ETag оновлюється, тому дану опцію слід використовувати при необхідності відновлення після помилок спільного доступу з нежорстким блокуванням.

o NoTracking (Без відстеження): DataServiceContext не відслідковує екземпляри сутностей. Оновлення сутності в контексті без відстеження реалізується за допомогою Etag, який оновлюється за допомогою AttachTo. Цей варіант не рекомендується до застосування.

```
context.AttachTo("Blogs", blog, "etag to use");
context.UpdateObject(blog);
context.SaveChanges();
```

Коли MergeOption контексту задано значення AppendOnly і об'єкт DataServiceContext вже відстежує сутність в результаті попередньої операції витягнення або додавання, повторне витягнення сутності з сервера не приведе до відновлення відслідковуємої сутності в контексті. Таким чином, якщо сутність на сервері була змінена, подальші оновлення / видалення призведуть до збою необхідних умов (PreCondition). Результатом усіх розглянутих вище операцій є передача HTTP- повідомлень на і з сервера. Додаток може відмовитися від використання клієнтської бібліотеки .NET і працювати на рівні HTTP / REST.

Розглянемо паралельні поновлення. Для поновлення сутності необхідно виконати наступні операції.

1. Отримати сутність з сервера.
2. Відновити об'єкт локально і повернути його на сервер. Припустимо, два процеси, що виконуються паралельно, намагаються оновити одну і ту ж сутність. Оскільки кроки 1 і 2 не є неподільними, на будь-якому з них може виникнути ситуація внесення змін до вже застарілої версії сутності. Для вирішення цієї проблеми Windows Azure Table використовує нежорстке блокування.

1. Для кожної сутності система зберігає версію, яка змінюється сервером при кожному оновленні.
2. Під час витягу сутності, сервер відправляє цю версію клієнта у вигляді ETag HTTP.
3. Коли клієнт передає запит UPDATE (оновити) на сервер, він відправляє на нього цей ETag у вигляді заголовка If-Match.
4. Якщо версія сутності, яка зберігається на сервері, аналогічна ETag в заголовку If-Match, зміна приймається, і зберігаєма на сервері сутність отримує нову версію. Ця нова версія повертається клієнту як заголовок ETag.

5. Якщо версія сутності на сервері відрізняється від ETag в заголовку If- Match, зміна відхиляється, і клієнтові повертається HTTP-помилка «Precondition failed» (необхідна умова не виконана).

При отриманні помилки «precondition failed» типовою поведінкою клієнтського додатку буде повторення всієї операції, як показано в фрагменті коду нижче.

1. Додаток має отримати цей об'єкт знову, тобто отримати його останню версію.
2. Відновити об'єкт локально і повернути його на сервер.

При використанні клієнтської бібліотеки .NET додаток отримує HTTP-код помилки як виняток DataServiceRequestException.

У прикладі нижче два різних клієнта виконують один і той же код для зміни тексту. Ці два клієнти намагаються задати Text різні значення.

1. Вони витягають сутність. При цьому для кожної сутності витягується ETag, наприклад, «v1». Обидва клієнти вважають, що попередня версія суті - «v1».
2. Кожен клієнт локально оновлює властивість Text.
3. Кожен клієнт викликає методи UpdateObject і SaveChanges.
4. Кожен клієнт відправляє на сервер HTTP-запит з заголовком «If- Match: v1 ».
5. Запит одного з клієнтів потрапляє на сервер першим.
   a. Сервер порівнює заголовок If-Match з версією сутності. Вони збігаються.
   b. Сервер застосовує зміну.
   c. Версія сутності на сервері оновлюється і стає «v2».
   d. Як відповідь клієнту відправляється новий заголовок «ETag: v2» .__
6. Далі на сервер надходить запит іншого клієнта. На цей момент зміни першого клієнта вже застосовані.
   a. Сервер порівнює заголовок If-Match з версією сутності. Вони не збігаються, оскільки версія сутності вже змінена на «v2», тоді як в запиті вказується версія «v1».
   b. Сервер відхиляє запит.

```
// Задаємо такий варіант об'єднання, який забезпечує
// збереження оновлень, але дозволяє оновлення etag.
// За замовчуванням застосовується значення AppendOnly, при якому
// вже відстежувана сутність не буде перезаписана значеннями,
// отриманими з сервера, в результаті чого в разі зміни
// суті на сервері використовується недійсний etag.
 context.MergeOption = MergeOption.PreserveChanges;
Blog blog =
(from blog in context.CreateQuery<Blog>("Blogs")
where blog.PartitionKey == "Channel9"
&& blog.RowKey == "Oct-29"

select blog).FirstOrDefault();
blog.Text = "Hi there again";
try
{
context.UpdateObject(blog);
```

```
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
OperationResponse response = e.Response.First();
if (response.StatusCode == (int)HttpStatusCode.PreconditionFailed)
{
//виконуємо запит об'єкта повторно, щоб отримати
// останній etag, і проводимо оновлення}
}
}
```

Для безумовного поновлення сутності додаток виконує наступне:

1. Створює новий об'єкт DataServiceContext або, в разі використання існуючого контексту, від'єднує об'єкт, як демонструє приклад нижче.
2. Приєднуємо сутність до контексту і використовуємо «*» як нове значення ETag.
3. Оновлюємо сутність.
4. Викликаємо SaveChanges.

```
// задаємо опцію об'єднання, що дозволяє перезапис,
// щоб забезпечити можливість поновлення відслідковуємої сутності
context.Detach (blog);
// Приєднуємо сутність до контексту, використовуючи ім'я таблиці, сутність,
// яка повинна бути оновлена, і "*" як значення etag.
context.AttachTo("Blogs", blog, "*");
blog.Text = "Hi there again";
try
{
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
// Обробка помилки, але в даному випадку формування  помилки PreCondition
неможливо}
```

Для запитів, які можуть повертати велику кількість результатів, система забезпечує два механізми:

1. Можливість отримувати перші N сутностей, використовуючи LINQ-функцію Take (N).
2. Маркер продовження, який позначає місце початку наступної множини результатів.

Система підтримує функцію повернення перших N відповідних запиту сутностей. Наприклад, якщо програма розробляється на .NET, для витягу перших N сутностей (в даному прикладі це перші 100 сутностей) можна використовувати LINQ-функцію Take (N).

```
serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext svc = new DataServiceContext(serviceUri);
var allBlogs = context.CreateQuery<Blog>("Blogs");
foreach (Blog blog in allBlogs.Take(100))
{
//  виконуємо деякі операції з кожним блогом}
```

Аналогічна функціональність підтримується в інтерфейсі REST через опцію рядку запиту $ top =N. Наприклад, запит «GET http: // <UriСервіса> / Blogs? $ Top = 100» забезпечував би повернення перших 100 сутностей, що відповідають запиту. Фільтрація виконується на сервері, тому у відповіді клієнту може бути передано максимум 100 сутностей.

1. У запиті вказується максимальна кількість сутностей, яка повинна бути повернута.
2. Кількість сутностей перевищує максимально дозволене сервером число сутностей у відповіді (в даний час це 1000 сутностей).
3. Загальний розмір сутностей у відповіді перевищує максимально допустимий розмір відповіді (в даний час це 4МБ, включаючи імена властивостей, але виключаючи xml-теги, які використовуються для REST).
4. На виконання запиту потрібно більше часу, ніж заданий період очікування сервера (в даний час це 60 секунд)

У будь-якому з цих випадків відповідь буде включати маркер продовження у вигляді спеціального заголовка. Для запиту до ваших сутностей використовуються такі заголовки:

- x-ms-continuation-NextPartitionKey
- x-ms-continuation-NextRowKey

Якщо клієнт отримав ці значення, він повинен передати їх з наступним запитом у вигляді опцій HTTP-запиту; в усьому іншому запит залишається незмінним. Це забезпечить повернення наступного набору сутностей, що починається з місця, позначеного маркером продовження.

Наступний запит виглядає наступним чином:
http://<UriСервиса>/Blogs?<исходный запит>&NextPartitonKey=<деяке значення >&NextRowKey=<другеЗначення>

Це повторюється до тих пір, поки клієнтом не буде отриманий відповідь без маркера продовження, що свідчить про витягнення всіх відповідних запиту результатів. Маркер продовження повинен розглядатися як непрозоре значення. Воно вказує на точку початку наступного запиту і може не відповідати фактичній суті в таблиці. Якщо в таблицю додається нова сутність, так що Key (нова сутність)> Key (остання сутність, витягнута запитом), але Key (нова сутність) < «Маркера продовження», тоді ця нова сутність не буде повернута повторним запитом, використовуємим маркером продовження. Але нові сутності, додані так, що Key (нова сутність)> «Маркера продовження», увійдуть до результатів, які повертаються наступними використовуємимі маркер продовження запитами.

Тепер розглянемо модель узгодженості, забезпечуємую Windows Azure Table. В рамках однієї таблиці система забезпечує гарантії транзакції ACID для всіх операцій вставки / оновлення / видалення для однієї сутності.

Для запитів в рамках окремої секції виконується ізоляція моментального знімку. Запит забезпечується узгодженим поданням секції з моменту його початку і протягом всієї транзакції. Миттєвий знімок забезпечує наступне:

1. Відсутність «брудного зчитування». Транзакція не бачитиме незафіксовані зміни, внесені іншими транзакціями, які виконуються паралельно. Будуть представлені тільки ті зміни, які були завершені до початку виконання запиту на сервері .

2. Механізм ізоляції миттєвого знімку дозволяє виконувати читання паралельно з оновленням секції без блокування цього оновлення.

Ізоляція миттєвого знімку підтримується тільки в рамках секції і в рамках одного запиту. Система не підтримує ізоляцію миттєвого знімку для декількох секцій таблиці або інших фаз запиту. Додатки відповідають за збереження узгодженості між множиною таблиць.

У прикладі MicroBlogging використовувалося дві таблиці: Channels і Blogs. Додаток виконує узгодження таблиць Channels і Blogs. Наприклад, коли канал видаляється з таблиці Channels, додаток повинен видалити відповідні блоги з таблиці Blogs.

Збої можуть виникати в моменти синхронізації стану множини таблиць. Додаток має вміти обробляти такі збої і мати можливість відновлювати роботу з моменту, на якому вона була перервана.

У попередньому прикладі, коли канал видаляється з таблиці каналів, програма має також видалити всі блоги цього каналу з таблиці Blogs. В ході цього процесу можуть виникати збої програми. Для обробки таких збоїв додаток може зберігати транзакцію в Windows Azure Queues, що дозволяє користувачеві відновити операцію видалення каналу і всіх його блогів навіть в разі збою.

Повернемося до прикладу з таблицями Channels і Blogs. Channels має такі властивості: Name (Ім'я) як PartitionKey, порожній рядок як RowKey, Owner (Власник), CreatedOn (Дата створення). І Blogs має властивості Channel Name (Ім'я каналу) як PartitionKey, CreatedOn як RowKey, Title (Назва), Blog, UserId. Тепер, коли канал видалений, необхідно забезпечити, щоб всі асоційовані з ним блоги також були видалені. Для цього виконуємо наступні кроки:

1. Створюємо чергу для забезпечення узгодженості таблиць, назвемо її «DeleteChannelAndBlogs» (Видалення каналів і блогів).

2. Під час отримання запиту на видалення каналу від ролі веб-інтерфейсу, ставимо в створену вище чергу елемент, який визначає ім'я каналу.

3. Створюємо робочі ролі, які чекатимуть подію додавання елемента в чергу «DeleteChannelAndBlogs».

4. Робоча роль вилучає елемент з черги DeleteChannelAndBlogs, задаючи для витягнутого елемента черги час невидимості протягом N секунд. При цьому елемент, який визначає ім'я каналу, який повинен бути видалений, вилучається. Якщо роль працівника видаляє елемент черги протягом цих N секунд, даний елемент буде видалено з черги. Якщо ні, елемент стане знову видимим і

доступним для використання робочою роллю. Під час вилучення елементу робоча роль робить наступне:

a. У таблиці Channels позначає канал як недійсний, щоб з цього моменту ніхто не міг виконувати читання з нього.

b. Видаляє з таблиці Blogs всі записи, для яких PartitionKey = "Імені каналу", вказаного в елементі черги.

c. Видаляє канал з таблиці Channels.

d. Видаляє елемент з черги.

e. Повертається.

Якщо в ході виконання, наприклад, кроку 4, виникає якийсь збій, проводиться аварійне завершення робочого процесу, при цьому елемент черги не видаляється з неї. Таким чином, як тільки відповідний елемент черги стане знову видимим (тобто коли закінчиться час, заданий як час очікування видимості), це повідомлення буде знову вилучено з черги робочим процесом, і процес видалення відновиться з кроку 4. Більш докладно обробка черг розглядається в документації Windows Azure Queue.

Таким чином, ми ознайомилися з однією з абстракцій даних Windows Azure Storage - Windows Azure Table. Це технологія, яка забезпечує структуроване сховище станів сервісу.

Azure Blob Services

Для роботи з ***Windows Azure Storage*** користувач має створити обліковий запис сховища. Виконується це через веб-інтерфейс порталу Windows Azure Portal. При цьому користувач отримує 256-розрядний секретний ключ, який потім використовується для аутентифікації запитів користувача до системи сховища. Зокрема, цим ключем створюється підпис HMAC SHA256 для запиту. Цей підпис передається з кожним запитом даного користувача для забезпечення аутентифікації через перевірку достовірності підпису HMAC Завдячуючи Windows Azure Blob додаток отримує можливість зберігання в хмарі великих об’єктів, до 50 ГБ кожний. Він підтримує високомасштабуєму систему великих об’єктів (*blob*), в котрій найбільш часто використовуємі blob разподіляються серед великої кількості серверів для обслуговування необхідних об’ємів трафіка. Ця система характеризується високою надійністтю і тривалісттю зберігання. Дані доступні в любий момент часу з любої точки планети і продубльовані, по крайній мірі, тричі для підвищення надійності. Крім того, забезпечується строга відповідність, що гарантирує негайну доступність об’єкту при його добавлені чи обновлені: всі зміни, внесені в попередній операції записі, миттєво видні при наступному читанні.

Azure Blob

Далі розглянемо модель даних Azure Blob.

На рис.3.4.6 представлено простір імен Windows Azure Blob.

• Обліковий запис сховища - будь-який доступ до Windows Azure Storage здійснюється через обліковий запис сховища.

- Це найвищий рівень простору імен для доступу до об'єктів blob.
- Обліковий запис може мати безліч контейнерів Blob.

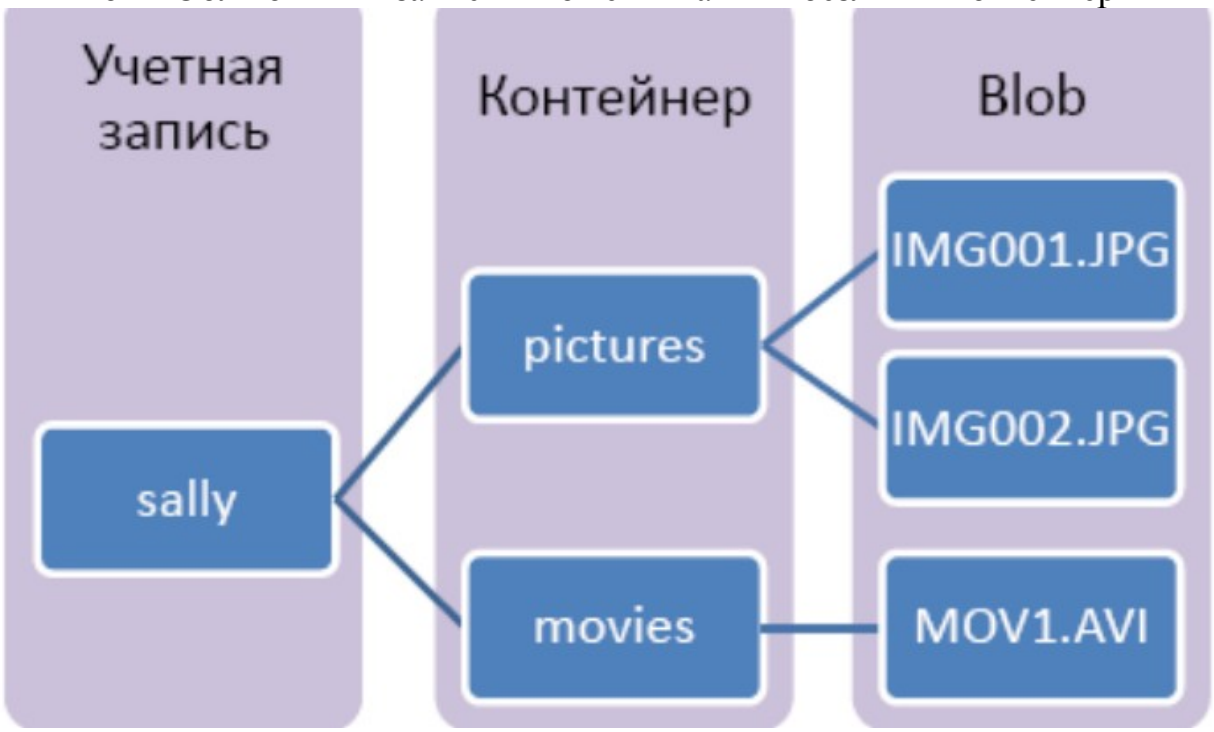


Рис.3.4.6 Загальне уявлення сховища Blob

• Контейнер Blob - контейнер забезпечує угрупування набору об'єктів blob. Область дії імені контейнера обмежена обліковим записом.

- Політики спільного використання задаються на рівні контейнера. В даний час підтримуються «Public READ» (Відкрите читання) і «Private» (Закритий). Якщо для контейнера визначена політика «Public READ», весь його вміст відкрито, доступно для читання без необхідності аутентифікації. Політика «Private» означає доступ з аутентифікацією, тобто тільки власник відповідного облікового запису має доступ до об'єктів blob цього контейнера.
- З контейнером можуть бути асоційовані метадані, які задаються у вигляді пар <ім'я, значення>. Максимальний розмір метаданих контейнера - 8Кб.
- Існує можливість отримання списку всіх об'єктів blob контейнера.

• Blob - Об'єкти blob зберігаються в контейнерах Blob Container та їх область дії обмежена цими контейнерами. Кожен blob може бути розміром до 50ГБ і має унікальне в рамках контейнера строкове ім'я. З blob можуть бути асоційовані метадані, які задаються у вигляді пар <ім'я, значення> і можуть досягати розміру 8Кб для blob. Метадані blob можуть бути отримані і задані окремо від даних blob.

Для доступу до Windows Azure Blob використовується наведені вище підходи. URI конкретного blob структурований таким чином: *http: // <обліковийзапис>.blob.core.windows.net/<контейнер><імяblob>*

Перша частина імені хоста утворена ім'ям облікового запису сховища, за яким слідує ключове слово «*blob». Це забезпечує направлення запиту в частину* ***Windows Azure Storage,*** *яка опрацьовує запити* ***blob.*** За іменем хосту іде ім’я

контейнера, «/» і потім ім'я blob. Існують обмеження іменування облікових записів і контейнерів . Наприклад, ім'я контейнера не може включати символ «/». Ще кілька зауважень з приводу контейнерів:

• Як говорилося вище, область дії контейнерів обмежена обліковим записом, якій вони належать. Контейнери зберігаються розподілено, що усуває ймовірність виникнення «вузьких місць» для трафіку при роботі з ними.

• Можлива затримка при відтворенні контейнера, який був недавно вилучений, особливо якщо в цьому контейнері знаходилася велика кількість об'єктів blob. Система повинна очистити об'єкти blob цього контейнера, перш ніж контейнер з таким же ім'ям зможе бути створений знову. Поки сервер видаляє всі об'єкти blob, спроби створення контейнера з таким же ім'ям будуть закінчуватися невдачею з формуванням помилки, що вказує на те, що контейнер знаходиться в процесі видалення.

• Команди на видалення, або створення абсолютно нового контейнера швидко передаються на сервер, і з додатком повертається підтвердження про їх виконання, навіть якщо операція видалення займає деякий час.

Розглянемо інтерфейс REST об'єктів Blob. Будь-який доступ до Windows Azure Blob виконується через стандартні HTTP-команди PUT / GET / DELETE інтерфейсу REST.

До команд HTTP / REST, підтримуваним для реалізації операцій з blob, відносяться:

• PUT Blob - Вставити новий або перезаписати існуючий blob з заданим ім'ям.

• GET Blob - Отримати весь blob або діапазон байтів blob, використовуючи стандартну HTTP-операцію для повернення діапазону GET.

• DELETE Blob - Видалити існуючий blob.

Всі ці операції з blob - Put, Get і Delete, - можуть бути виконані з використанням наступного URL:

http: // <обліковийзапис> .blob.core.windows.net / <контейнер> / <імяblob>

Один запит PUT забезпечує можливість завантаження в хмару blob розміром до 64 МБ. Для завантаження blob, розмір якого перевищує 64 МБ, використовується технологія завантаження блоками, описаними в наступному розділі.

Один з цільових сценаріїв Windows Azure Blob - ефективне завантаження об'єктів blob, розміром десятки гігабайт. Windows Azure Blob забезпечує це наступним чином:

Завантаження Blob (наприклад, Movie.avi) розбивається на послідовні блоки. Наприклад, ролік розміром 10ГБ може бути розбитий на 2500 блоків по 4 Мб, при цьому перший блок буде представляти діапазон байтів даних від 1 до 4194304, другий блок - від 4194305 до 8388608 і т.д.

• Кожному блоку присвоюється унікальний ID / ім'я. Область дії цього унікального ID обмежена ім'ям завантажуємого blob. Наприклад, перший блок може бути названий «Block 0001», другий - «Block 0002» і т.д.

• Кожен блок розміщується в хмарі. Для цього використовується операція PUT із зазначенням представленого вище URL із запитом, що визначає, що це

операція розміщення блоку, і ID блоку. Продовжуючи приклад, при розміщенні першого блоку буде вказано ім'я blob «Movie.avi» і ID блоку «Block 0001».

• Коли всі блоки розміщені в Windows Azure Storage, передається список завантажених блоків, щоб представити blob, з яким вони асоційовані. Виконується це за допомогою операції PUT із зазначенням представленого вище URL із запитом, що визначає, що це команда blocklist. Після цього HTTP-заголовок містить список блоків, які повинні використовуватися в цьому blob. У разі успішного завершення цієї операції отримуємо список блоків, що представляє придатну для читання версію blob. Тепер blob може бути зчитаний за допомогою описаної вище команди GET Blob.

На рис.3.4.7 представлено місце блоків в концепції даних Windows Azure Blob.

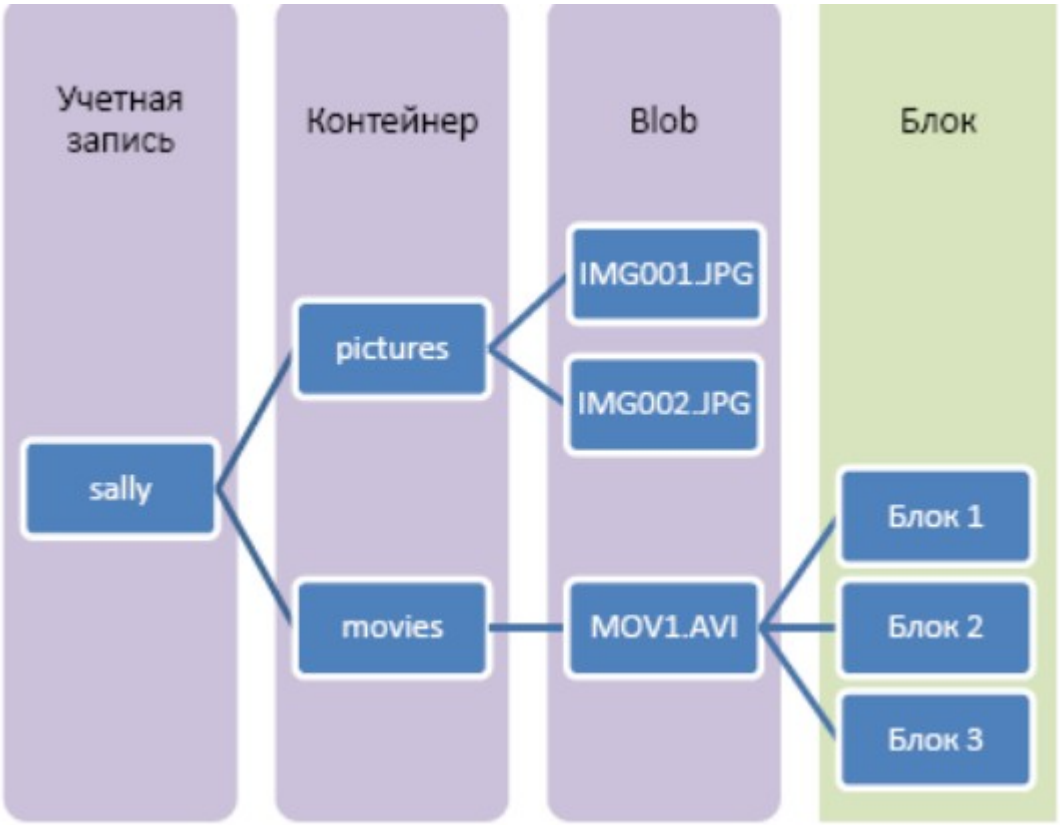


Рис.3.4.7 Загальне уявлення сховища Blob, додавання блоків.

Доступ до об'єктів blob може здійснюватися за допомогою операцій PUT і GET з використанням наступного URL:

http: // <обліковийзапис> .blob.core.windows.net / <контейнер> / <імяblob>

У прикладі, представленому на рис.3.4.7, зображення з наступними URL можуть бути розміщені однією операцією PUT:

http://sally.blob.core.windows.net/pictures/IMG001.JPG
http://sally.blob.core.windows.net/pictures/IMG002.JPG

Ті ж URL можуть використовуватися для повернення об'єктів blob. Одна операція PUT може забезпечити розміщення в сховищі об'єктів blob розміром до 64МБ. Для збереження об'єктів blob розміром більше 64МБ і аж до 50 ГБ

необхідно спочатку розмістити всі блоки за допомогою відповідної кількості операцій PUT і потім, за допомогою все тієї ж операції PUT, передати список блоків, щоб забезпечити придатну для читання версію blob. У прикладі, який ілюструє рис. 3.4.7, тільки після розміщення всіх блоків і підтвердження їх приналежності blob за допомогою списку блоків blob може бути зчитаний з використанням наступного URL:

http://sally.blob.core.windows.net/pictures/MOV1.AVI

Операції GET завжди виконуються на рівні blob і не припускають використання блоків. Розглянемо абстракції даних блоків. Кожен блок ідентифікує ID блоку розміром до 64 байт. Область дії ID блоку обмежена ім'ям blob, тому різні об'єкти blob можуть мати блоки з однаковими ID. Блоки незмінні. Кожен блок може бути розміром до 4 Мб, і один blob може включати блоки різного розміру. Windows Azure Blob забезпечує наступні операції рівня блоку:

• PUT block - завантажити блок blob. Зверніть увагу, що успішно завантажений за допомогою операції PUT block блок не є частиною blob до тих пір, поки це не буде підтверджено списком блоків, що завантажуються операцією PUT blocklist.

• PUT blocklist - підтвердити blob через надання списку ID блоків, його складових. Зазначені в цій операції блоки повинні бути вже успішно завантажені через виклики PUT. Порядок блоків в операції PUT blocklist забезпечить придатну для читання версію blob.

• GET blocklist - витягти список блоків, переданий раніше для blob операцією PUT blocklist. У повернутому списку блоків вказуються ID і розмір кожного блоку.

У всіх розглянутих далі прикладах використовується blob «MOV1.avi», що розташовується в контейнері «movies» (ролики) під обліковим записом «sally».

Нижче представлений приклад REST-запиту для розміщення блоку розміром 4 Мб за допомогою операції PUT block. Зверніть увагу, що використовується HTTP-команда PUT. «? Comp = block» вказує на те, що це операція PUT block. Потім задається BlockID. Параметр Content-MD5 може бути заданий для захисту від помилок передачі по мережі і забезпечення цілісності. В даному випадку, Content-MD5 - це контрольна сума MD5 даних блоку в запиті. Контрольна сума перевіряється на сервері, в разі розбіжності повертається помилка. Параметр Content-Length (Довжина вмісту) визначає розмір вмісту блоку. Також в заголовку HTTP-запиту є заголовок авторизації, як показано нижче.

PUT http://sally.blob.core.windows.net/movies/MOV1.avi
? Comp = block & blockid = BlockId1 & timeout = 60
HTTP / 1.1 Content-Length: 4194304
Content-MD5: HUXZLQLMuI / KZ5KDcJPcOA ==
Authorization: SharedKey sally: F5a + dUDvef +
PfMb4T8Rc2jHcwfK58KecSZY + l2naIao =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
..... Block Data Contents .....

Нижче представлений приклад REST-запиту для операції PUT blocklist. Зверніть увагу, що використовується HTTP-команда PUT. «? Comp = blocklist» вказує на те, що це операція PUT blocklist. Список блоків задається в тілі HTTP-запиту в форматі XML, як показано в прикладі нижче. Зверніть увагу, що значення поля Content-Length в заголовку запиту відповідає розміру тіла запиту, а не розміру створюваного blob. Також в заголовку HTTP-запиту є заголовок авторизації, як показано нижче.

PUT http://sally.blob.core.windows.net/movies/MOV1.avi
? Comp = blocklist & timeout = 120
HTTP / 1.1 Content-Length: 161213
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
<? Xml version = "1.0" encoding = "utf-8"?>
<BlockList>
<Block> BlockId1 </ Block>
<Block> BlockId2 </ Block>
.....
</ BlockList>

Нижче представлений приклад REST-запиту для операції GET blob. В даному випадку використовується HTTP-команда GET. Цей запит забезпечить витяг всього вмісту заданого blob. Якщо для контейнера, якому належить blob (в даному прикладі «Movies»), задана політика спільного використання «Private», для отримання blob необхідно пройти аутентифікацію. Якщо задана політика спільного використання «Public-Read», аутентифікація не вимагається і заголовок аутентифікації в заголовку запиту не потрібен.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP / 1.1
Authorization: SharedKey sally: RGllHMtzKMi4y / nedSk5Vn74IU6 / fRMwiPsL + uYSDjY =
X-ms-date: Mon, 27 Oct 2016 17:00:25 GMT

Як показано в прикладі нижче, також підтримується операція GET для витягу діапазону байта заданого blob.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP/1.1
Range: bytes=1024000-2048000

Завантаження blob у вигляді списку блоків має наступні переваги:

• Можливість продовження - для кожного блоку окремо можна перевірити успішність завантаження, в разі збою повторити спробу завантаження і продовжити виконання з цього моменту.

• Паралельне завантаження - завантаження блоків може виконуватися паралельно, що забезпечує скорочення часу завантаження дуже великих об'єктів blob.

• Завантаження не по порядку - Блоки можуть завантажуватися в довільному порядку. Значення має лише порядок блоків в списку операції PUT blocklist.

Список блоків в операції PUT blocklist визначає придатну для читання версію blob.

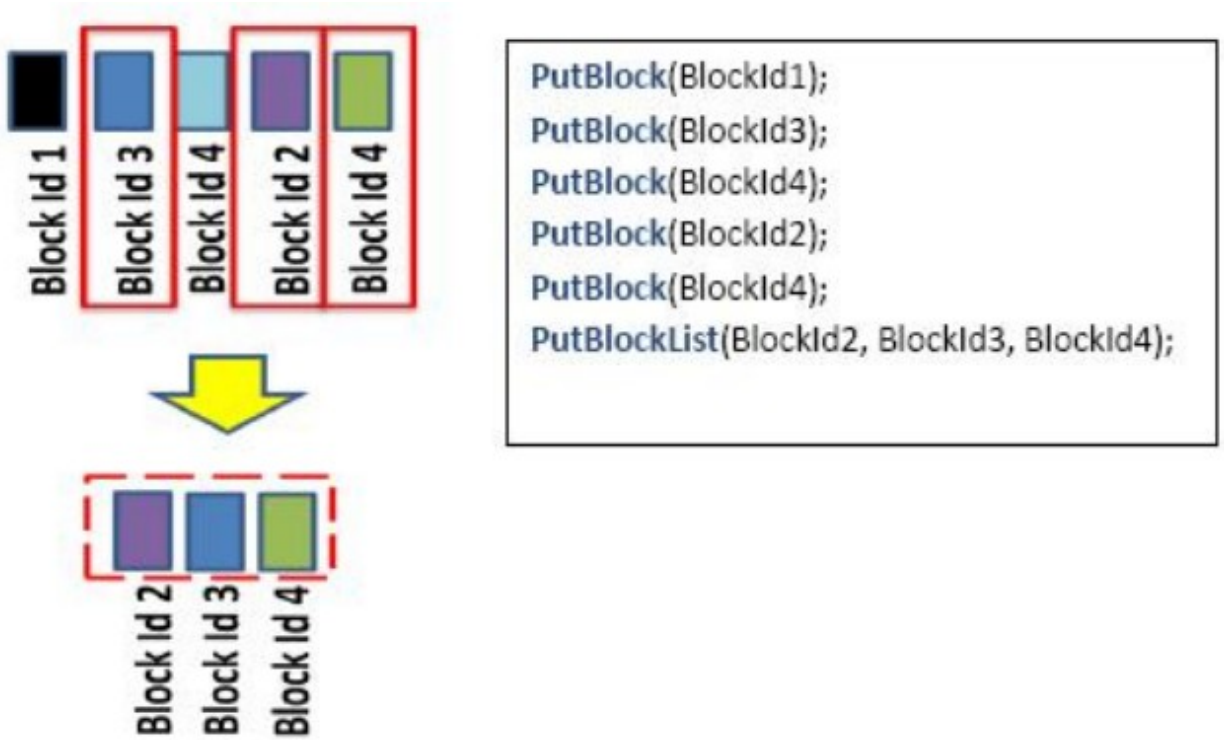


Рис.3.4.8 Сценарій завантаження блоків

Використовуючи представлений на рис.3.4.8 приклад, опишемо різні сценарії, можливі при використанні блоків для завантаження об'єктів blob:

• Завантаження блоків з однаковими ID блоку - коли для одного blob завантажуються блоки з однаковими ID блоку, при формуванні остаточно версії blob в операції PUT blocklist з усіх блоків з однаковим ID буде використовуватися тільки завантажений найостаннішим. В наведеному вище прикладі завантажуються два блоки з BlockId4 і тільки другий з них буде використовуватися в остаточному списку блоків blob.

• Завантаження блоків в довільному порядку - блоки можуть завантажуватися в порядку, відмінному від зазначеного в остаточному списку блоків blob. В наведеному вище прикладі в остаточному списку блоки розташовуються в порядку BlockId2, BlockId3 і BlockId4, але завантажувалися вони в іншій послідовності. Упорядкування даних blob (через операцію GET) в придатну для читання версію виконується відповідно до списку, зазначеного в операції PUT blocklist.

• Невикористані блоки - деякі блоки можуть ніколи не увійти в остаточний список блоків blob. Ці блоки будуть видалені системою в процесі збору «сміття». У розглянутому прикладі такими блоками є BlockId1 і перший блок з ID BlockId4. Точніше кажучи, як тільки blob створений за допомогою операції PUT blocklist, всі завантажені, але які не увійшли в список блоків операції PUT blocklist блоки будуть вилучені шляхом видалення «сміття».

Завантаження великого blob може займати досить тривалий час. При цьому завантажені, але не використані блоки займають місце в сховищі. Багато завантажених блоків можуть ніколи не увійти в список PUT blocklist. У разі

відсутності активності для даного blob протягом тривалого періоду часу (в даний час цей період становить тиждень), ці невикористані блоки будуть видалені системою як сміття.

Цікавим є сценарій паралельного завантаження блоків для одного blob. В цьому випадку заслуговують на увагу два питання:

• ID блоків - якщо для завантаження блоків в один blob додаток використовує множину клієнтських модулів записі, щоб уникнути конфліктів ID блоків повинні бути унікальними серед всіх цих модулів записі, або вони повинні представляти вміст записуємого блоку (таким чином, якщо один і той же блок записується декількома клієнтами, ID блоку у всіх клієнтів буде однаковим, оскільки він представляє одні й ті ж дані). Щоб уникнути помилок при потенційній можливості запису одного Blob декількома модулями записі в якості ID блоку рекомендується використовувати хеш (наприклад, MD5-хеш) вмісту блоку. Таким чином, ID блоку представлятиме його вміст.

• Пріоритет має перша фіксація - в ситуації, коли безліч клієнтів виконують завантаження блоків для одного blob паралельно, пріоритет має перша фіксація blob за операції PUT blocklist (або модуль запису, який викликав PUT blob першим). Всі інші невикористані блоки, завантажені іншими модулями записі для blob з цим ім'ям, будуть видалені в процесі видалення “сміття”. Отже, для ефективного паралельного поновлення blob необхідна координація всіх клієнтів, які ведуть запис паралельно.

Windows Azure Blob підтримує умовні PUT і GET, які забезпечують реалізацію ефективної обробки паралелізму і клієнтського кешування.

Умовний PUT може використовуватися в ситуаціях, коли один blob оновлюється кількома користувачами. Наприклад, завантаження blob може виконуватися підчас останньої зміни; це гарантує, що версія змінюваного blob аналогічна версії, яку змінює клієнт. Так може бути реалізований спільний доступ з нежорстким блокуванням. Скажімо, два клієнта, A і B, оновлюють один і той же blob. Вони паралельно виконують читання версії blob, вносять в неї якісь зміни і знову завантажують в сховище. У цьому сценарії кожен з клієнтів записує час останньої зміни витягнутого зі сховища blob (нехай час останньої зміни буде X). Коли вони готові завантажити оновлену версію blob назад в сховище, вони роблять це за допомогою умовного PUT на підставі збереженого при добуванні blob часу останньої зміни. В операції повинно бути визначено, що умовою виконання PUT є «якщо не змінювався з моменту X». Таким чином, якщо blob був змінений іншим клієнтом в проміжок часу з моменту X, операція оновлення дасть збій, і клієнт отримає повідомлення про це.

Умовний GET може використовуватися для ефективної обробки питань відповідності вмісту кешів. Наприклад, клієнт має локальний кеш об'єктів blob, в якому кешуються найчастіше викликаємі зі сховища blob. Для кожного кешованого blob записується час його останньої зміни. Коли клієнтський кеш приймає рішення оновити об'єкти blob зі сховища, він може використовувати умовний GET на підставі часу зміни (з умовою «якщо змінений з моменту X»). Таким чином, зі сховища будуть завантажуватися тільки ті об’єкти blob, які

були змінені в період часу з моменту X і відрізняються від своєї керованої копії.

Система Blob забезпечує інтерфейс для перечислення об'єктів *blob* контейнера. Підтримується ієрархічний перелік об'єктів *blob* контейнера і механизм продовження, що забезпечує виконання великої кількості об'єктів *blob*.

Интерфейс *ListBlobs* підтримує параметри *«prefix»* (префікс) і *«delimiter»* (розділювач), які забезпечують можливість побудови ієрархічного переліку об'єктів *blob*. Наприклад, хай в обліковому запису *«sally»* є контейнер *«movies»* объектів *blob* з такими іменами:

Action/Rocky1.wmv
Action/Rocky2.wmv
Action/Rocky3.wmv
Action/Rocky4.wmv
Action/Rocky5.wmv
Drama/Crime/GodFather1.wmv
Drama/Crime/GodFather2.wmv
Drama/Memento.wmv
Horror/TheBlob.wmv

Як бачите, «/» використовується як роздільник для створення подібної каталогу ієрархії імен blob. Щоб отримати список всіх папок , задаємо в запиті ListBlobs «delimiter = /». І ось як буде виглядати запит і частина відповіді:

запит:

GET http://sally.blob.windows.net/movies?comp=list&delimiter=/

Відповідь:

<BlobPrefix>Action</BlobPrefix>
<BlobPrefix>Drama</BlobPrefix>
<BlobPrefix>Horror</BlobPrefix>

Зверніть увагу, тег «BlobPrefix» вказує на те, що відповідний запис є префіксом імені blob, а не повним ім'ям blob. Також слід зауважити, що один і той же префікс повертається в результаті тільки один раз.

Наступним етапом можна поєднувати префікс і роздільник для отримання списку вмісту підпапки. Наприклад, задаючи «prefix = Drama /» і «delimiter = /», отримуємо список всіх підпапок і файлів каталогу «Drama»:

запит:

GET http://sally.blob.windows.net/movies?comp=list &prefix=Drama/ &delimiter=/

відповідь:

<BlobPrefix>Drama/Crime</BlobPrefix>
<Blob>Drama/Memento.wmv</Blob>

«Drama / Memento.wmv» - це повне ім'я blob, отже, воно так і позначено.

Інтерфейс ListBlobs забезпечує можливість задавати «maxresults», тобто максимальне число результатів, яке повинно бути повернуто в цьому виклику. Більш того, система визначає верхню межу для максимального числа результатів, які можуть бути повернуті одним викликом . Після досягнення меншого з цих двох граничних значень виклик повертається з відповідною кількістю результатів і непрозорим «NextMarker» (маркер наступного). Наявність цього маркеру свідчить про те, що даний запит не забезпечив повернення всіх можливих результатів. «NextMarker» може використовуватися для продовження складання списку для наступної сторінки результатів.

У попередньому прикладі припустимо, що потрібно скласти список всіх об'єктів blob каталогу «Action», повертаючи кожен раз максимум по 3 результату. У цьому випадку перший набір результатів був би таким:

запит:

*GET http://sally.blob.windows.net/movies?comp=list &prefix=Action &maxresults=3 *

віповідь:

<Blob>Action/Rocky1.wmv</Blob>
<Blob>Action/Rocky2.wmv</Blob>
<Blob>Action/Rocky3.wmv</Blob>
<NextMarker> OpaqueMarker1</NextMarker>

З першим набором об'єктів blob повертається і непрозорий маркер, який може бути переданий в другій виклик ListBlobs. Тоді цей виклик забезпечить повернення таких результатів:

запит:

GET http://sally.blob.windows.net/movies?comp=list &prefix=Action &maxresults=3
&marker=OpaqueMarker1

відповідь:

<Blob>Action/Rocky4.wmv</Blob>
<Blob>Action/Rocky5.wmv</Blob>
<NextMarker></NextMarker>

Як показано вище, повернуті залишені об'єкти blob каталогу; «NextMarker» порожній, це свідчить про те, що отримані всі результати.

Azure Queue Services

Windows Azure Queue надає надійний механізм доставки повідомлень. Вона пропонує простий алгоритм диспетчеризації асинхронних завдань, який забезпечує можливість підключення до різних компонентів додатку в хмарі. Черги Windows Azure Queue характеризуються високою надійністю, постійністю і продуктивністю. Поточна реалізація гарантує одноразову

обробку повідомлення. Більш того, Windows Azure Queue має REST-інтерфейс, таким чином, додатки можуть створюватися на будь-якій мові програмування і виконувати доступ до черги через веб в будь-який час з будь-якої точки Інтернету.

Розглянемо створення додатків в хмарі з використанням Azure Queue. Windows Azure Queue дозволяє розділити різні частини програми в хмарі, що робить можливим використання різних технологій для створення цих додатків і їх масштабування відповідно до потреб трафіку. Рис.3.4.9. ілюструє простий і поширений сценарій для додатків в хмарі.

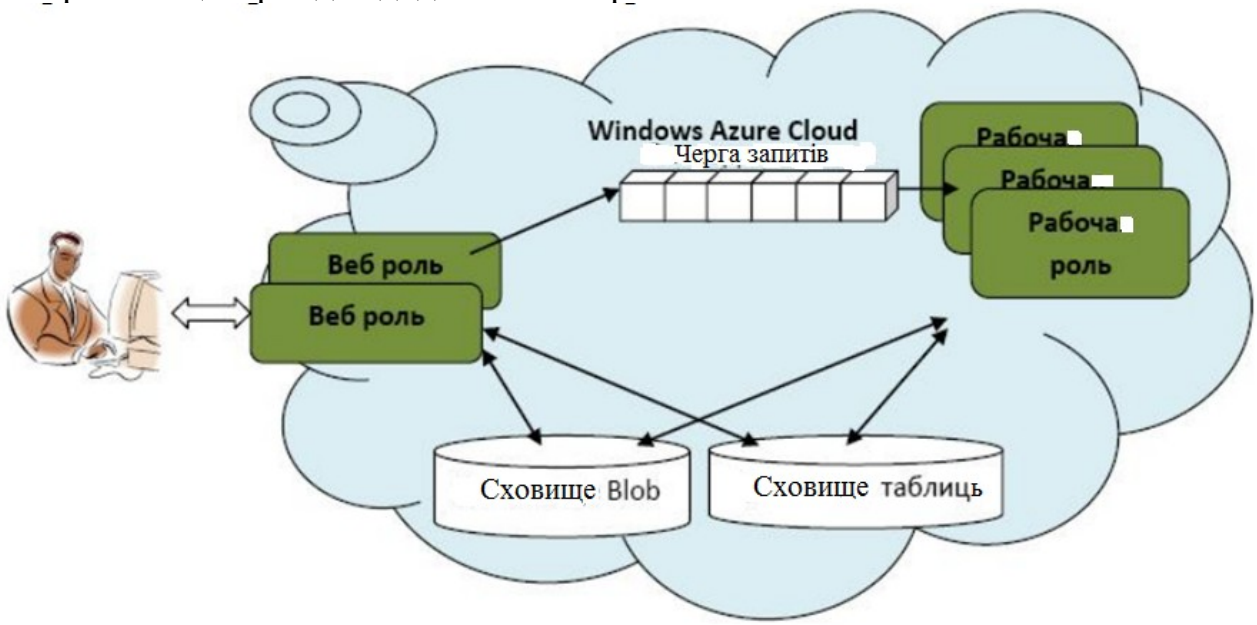


Рис.3.4.9. Побудова додатків для хмари з використанням Azure Queue

Є ряд веб ролей, на яких розміщується інтерфейсна логіка обробки веб-запитів. Також існує ряд робочих ролей, які реалізують бізнес-логіку програми. Веб ролі обмінюються інформацією з робочими ролями за допомогою наборів запитів.

Розглянемо, як приклад, додаток онлайн сервісу відеохостингу. Користувачі можуть завантажувати відео в цей додаток; після цього додаток може автоматично перетворювати і зберігати цей відеофайл в різних форматах мультимедіа; крім того, додаток автоматично індексує дані опису відео для спрощення пошуку (наприклад, за ключовими словами опису, іменами акторів, режисерів, назвою і т.д.).

Такий додаток може використовувати описану раніше архітектуру. Веб-ролі реалізують рівень представлення і обробляють веб-запити користувачів. Користувачі можуть завантажувати відео через веб-інтерфейси. Відео-файли можуть зберігатися як великі двійкові об'єкти в сховище Azure Blob. Додаток може також обслуговувати ряд таблиць для обліку наявних відеофайлів і зберігання індексів, використовуваних для пошуку. Робочі ролі виконують перетворення вхідних відеофайлів в різні формати і збереження їх в сховищі Azure Blob. Робочі ролі також відповідають за оновлення таблиць цього додатку в Azure Table. При отриманні запиту користувача (наприклад, запиту на завантаження відео) веб-ролі створюють робочий елемент і розміщують його

в чергу запитів. Робочі ролі витягають ці робочі елементи з черги і оброблюють відповідним чином. У разі успішної обробки робоча роль повинна видалити робочий елемент з черги, щоб уникнути повторної його обробки іншою робочою роллю.

Завдяки застосуванню Windows Azure Queue така архітектура має ряд переваг:

1. Масштабованість - Додаток може легше масштабуватися відповідно до потреб трафіку. Переваги, пов'язані з масштабуванням: По-перше, довжина черги безпосередньо відображає наскільки добре робочі ролі справляються з загальним робочим навантаженням. Збільшення черги свідчить про те, що робочі ролі не можуть обробляти дані з необхідною швидкістю. В цьому випадку додатку може знадобитися збільшити кількість робочих ролей, щоб забезпечити більш швидку обробку. Якщо довжина черги незмінно залишається близькою до нуля, це означає, що серверна частина програми має більші обчислювальні потужності, ніж потрібно. У цьому випадку програма може скоротити кількість робочих ролей для забезпечення раціонального витрачання ресурсів. Відстежуючи розміри черги і налаштовуючи кількість внутрішніх вузлів, додатки можуть ефективно масштабуватися відповідно до обсягів трафіку. По-друге, застосування черг дозволяє розділити частини програми і виконувати їх масштабування незалежно один від одного, що набагато спрощує це завдання. Додаток може вільно масштабувати критично важливі компоненти, додаючи для них більше ресурсів / комп'ютерів. По-третє, застосування черг забезпечує гнучкість для ефективного використання ресурсів в додатку, що підвищує ефективність масштабування. Тобто для робочих елементів з різними пріоритетами і / або різних розмірів можуть використовуватися різні черги, які будуть оброблятися різними пулами робочих ролей.
2. Поділ ролей - Використання черг дозволяє розділити різні частини програми, що забезпечує істотну гнучкість і розширюваність з точки зору побудови програми. Повідомлення в черзі можуть бути в стандартному або розширюваному форматі, такому як XML, що забезпечує незалежність компонентів на обох кінцях черги один від одного, оскільки вони можуть розуміти повідомлення в черзі. Різні частини системи можуть бути реалізовані з використанням різних технологій і мов програмування. Наприклад, компонент на стороні черги може бути написана на .NET Framework, а інший компонент – на Python. Більше того, зімни, проходящі в середині компонента, не мають ніякого впливу на остальну систему. Наприклад, компонент може бути зіменим з використанням зовсім іншої технології чи мови програмування і при цьому система буде продовжувати працювати і для цього не прийдеться змінювати інші компоненти так як використання черг забезпечує розділення компонентів. Крім того використання черг забезпечує співіснування в системі різних реалізацій одного і того ж компоненту, отже додаток може вільно переходити до нових технологій: стара і нова реалізації можуть виконуватися одночасно на різних серверах і

опрацьовувати робочі елементи однієї черги. При цьому інші компоненти додатку ніяк не будуть зачеплені.

3. Сплески трафіку - Черги забезпечують буферизацію, що компенсує сплески трафіку і скорочує вплив, який чиниться збоями окремих компонентів. У розглянутому раніше прикладі можливі випадки надходження великої кількості запитів в короткий проміжок часу. Робочі ролі не можуть швидко обробити всі запити. У цьому випадку запити не відхиляються, а буферизуются в чергу, і робочі ролі отримують можливість обробляти їх у власному нормальному темпі, поступово повертаючись до звичайного режиму роботи. Це дозволяє додатку обробляти нерівномірний трафік без зниження надійності.

Завдяки моделі черги додатки застраховані від втрати даних і зниження надійності навіть в умовах систематичних збоїв компонентів додатку.

Windows Azure Queue має наступну модель даних.

• Облікова запись сховища - Будь-який доступ до Windows Azure Storage здійснюється через обліковиу запись сховища.

• Це найвищий рівень простору імен для доступу до черг і їх повідомлень. Для використання Windows Azure Storage користувач має створити облікову запись сховища. Виконується це через веб-інтерфейс порталу Windows Azure Portal.

При створенні облікового запису користувач отримує 256-розрядний таємний ключ, котрий надалі використовується для аутентифікації запитів цього (за допомогою цього таємного ключа створюється підпис HMAC SHA256 для запиту). Цей підпис передається з кожним запитом для аутентифікації через перевірку достовірності підпису HMAC.

• Облікова запись може мати багато черг.

Черга – черга має багато сповіщень. Область дії імені черги обмежена обліковим записом.

1. Кількість сповіщень в черзі не обмежено.
2. Сповіщення зберігаються максимум тиждень. Система видаляє сповіщення, поступивші більше тиждня назад .
3. З чергами можуть бути асоційовані метадані. Метадані представляються в формі пар <*имя, значение*>,їх розмір може складати максимум 8КБ на чергу.

• Повідомлення

Повідомлення зберігаються в чергах. Кожне повідомлення може бути розміром не більше 8Кб. Для зберігання даних більшого розміру використовуються сховища Azure Blob або Azure Table, а в повідомленні вказується ім'я великого двійкового об'єкта / сутності. Зверніть увагу, що коли повідомлення поміщається в сховище, його дані можуть бути двійковими. Але при добуванні повідомлень зі сховища відповідь формується в форматі XML, і дані повідомлення повертаються base64-кодованими. Повідомлення можуть повертатися з черги в будь-якому порядку, і повідомлення може бути

повернуто кілька разів. Розглянемо деякі параметри, які використовуються Azure Queue Service:

1. MessageID: Значення GUID, яке ідентифікує повідомлення в черзі.

2. VisibilityTimeout: Ціле значення, яке визначає час очікування видимості повідомлення в секундах. Максимальне значення - 2 години. Значення за замовчуванням - 30 секунд.

3. PopReceipt: Рядок, який повертається для кожного витягнутого повідомлення. Цей рядок, разом з MessageID, необхідний для видалення повідомлення з черги (Queue). Даний параметр слід розглядати як непрозорий, оскільки в майбутньому його формат і вміст можуть бути змінені.

4. MessageTTL: Визначає термін життя (time-to-live, TTL) повідомлення в секундах. Максимально допустимий термін життя - 7 днів. Якщо цей параметр опущений, термін життя за замовчуванням - 7 днів. Якщо протягом терміну життя повідомлення не буде видалене з черги, воно буде видалено системою зберігання в процесі очищення.

URI конкретної черги структуровано таким чином:

http: // <облікова запись> .queue.core.windows.net / <Імя черги>

Перша частина імені хоста утворена ім'ям облікового запису сховища, за яким слідє ключове слово «queue». Це забезпечує направлення запиту до частини Windows Azure Storage, яка обробляє запити черги. За ім'ям хоста йде ім'я черги. Існують обмеження іменування облікових записів і черг . Наприклад, ім'я черги не може включати символ «/».

Будь-який доступ до Windows Azure Queue виконується через HTTP-інтерфейс REST. Підтримуються як HTTP, так і HTTPS протоколи.

До командам HTTP / REST на рівні облікового запису відносяться:

• List Queues - Представити список всіх черг цього облікового запису.

До командам HTTP / REST на рівні черги відносяться:

• Create Queue - Створити чергу під даним обліковим записом.

• Delete Queue - Видалити зазначену чергу і її вміст без можливості відновлення.

• Set Queue Metadata - Задати / оновити визначені користувачем метадані черги. Метадані асоційовані з чергою в вигляді пар ім'я / значення. Ця команда забезпечить перезапис всіх існуючих метаданих новими.

• Get Queue Metadata - Витягти визначені користувачем метадані черги, а також приблизну кількість повідомлень в заданій черзі.

Операції рівня черги можуть виконуватися з використанням наступного URL:

http: // <обліковий запис> .queue.core.windows.net / <ІмяЧерги>

До командам HTTP / REST, підтримуваним для реалізації операцій на рівні повідомлення, відносяться:

• PutMessage (QueueName, Message, MessageTTL) - Додати нове повідомлення в кінець черги. MessageTTL визначає термін життя даного повідомлення. Повідомлення може зберігатися в текстовому або двійковому форматі, але повертається base64-кодованим.

• GetMessages (QueueName, NumOfMessages N, VisibilityTimeout T) - Витягти N повідомлень з початку черги і зробити їх невидимими протягом заданого VisibilityTimeout проміжку часу T. Ця операція поверне ID повідомлення, повернутого разом з PopReceipt. Повідомлення можуть повертатися з черги в будь-якому порядку, і повідомлення може бути повернуто кілька разів.

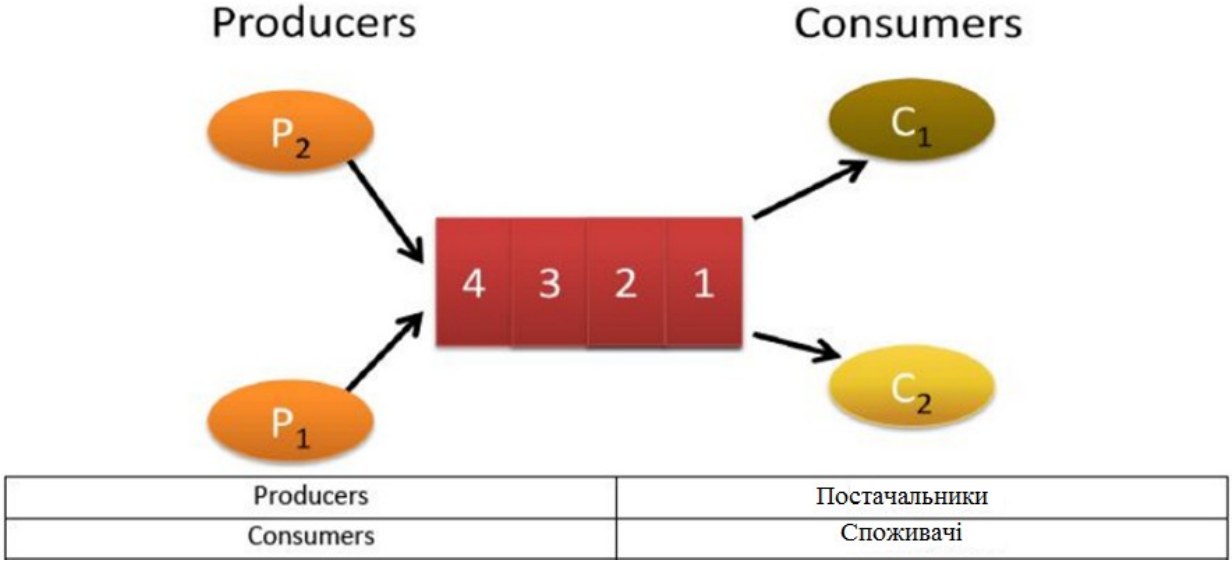
• DeleteMessage (QueueName, MessageID, PopReceipt) - Видалити повідомлення, асоційоване з даними PopReceipt, повернутим раніше викликом GetMessage. Зверніть увагу, що якщо повідомлення не буде видалене, воно повторно з'явиться в черзі після закінчення VisibilityTimeout.

• PeekMessage (QueueName, NumOfMessages N) - Витягнути N повідомлень з початку черги, не роблячи повідомлення невидимими. Ця операція поверне ID повідомлення для кожного повернутого повідомлення.

• ClearQueue - Видалити всі повідомлення з заданої черги. Зверніть увагу, що викликаюча сторона повинна повторювати цю операцію до тих пір, поки отримує повідомлення про успішне її виконання, це забезпечить видалення всіх повідомлень черги.

Операції рівня повідомлення можуть бути виконані з використанням наступного URL:

http: // <учетнаязапісь> .queue.core.windows.net / <ІмяОчереді> / messages На рис.3.4.10 наведено приклад, який ілюструє семантику Windows Azure Queue.



Producers	Постачальники
Consumers	Споживачі

рис.3.4.10 Приклад використання черги

У цьому прикладі постачальники (P1 і P2) і споживачі (C1 і C2) обмінюються інформацією через подану вище чергу. Постачальники формують робочі елементи і поміщають їх у вигляді повідомлень в чергу. Споживачі вилучають повідомлення / робочі елементи з черги і обробляють їх. Може існувати багато постачальників і багато споживачів. Розглянемо послідовність дій:

1. C1 витягує повідомлення з черги. Ця операція повертає повідомлення 1 і робить його невидимим в черзі на 30 секунд (приймаємо в даному прикладі, що використовується VisibilityTimeout за замовчуванням, що складає 30 секунд).

2. Потім з'являється C2 і витягує з черги ще одне повідомлення. Оскільки повідомлення 1 як і раніше невидиме, ця операція не побачить повідомлення 1 і поверне C2 повідомлення 2.

3. По завершенні обробки повідомлення 2 C2 викликає Delete, щоб видалити повідомлення 2 з черги.

4. Тепер уявімо, що C1 дає збій в ході обробки повідомлення 1, таким чином, C1 не видаляє це повідомлення з черги.

5. Після завершення часу VisibilityTimeout для повідомлення 1, воно знову з'являється в черзі.

6. Після того, як повідомлення 1 повторно з'явилося в черзі, воно може бути вилучено наступним викликом від C2. Тоді C2 повністю обробить повідомлення 1 і видалить його з черги.

Як показано в цьому прикладі, семантика API черги гарантує кожному повідомленні в черзі шанс бути обробленим повністю, в крайньому разі, один раз. Тобто, якщо виникає збій споживача в період після вилучення ним повідомлення з черги і до його видалення, повідомлення знову з'явиться в черзі після закінчення часу VisibilityTimeout. Це забезпечує можливість цьому повідомленню бути обробленим повністю іншим споживачем.

Розглянемо REST-запити, які використовуються Windows Azure Queue.

Нижче показаний приклад REST-запиту для операції постановки в чергу. Зверніть увагу, що використовується HTTP-команда PUT. Задається необов'язкова опція «messagettl», яка визначає термін життя повідомлення в секундах. Максимальний термін життя - 7 днів. Якщо цей параметр опущений, значення за замовчуванням - 7 днів. Після закінчення терміну життя повідомлення буде видалено системою . Параметр Content-MD5 може бути заданий для захисту від помилок передачі по мережі і забезпечення цілісності. В даному випадку, Content-MD5 - це контрольна сума MD5 даних повідомлення в запиті. Параметр Content-Length (Довжина вмісту) визначає розмір вмісту повідомлення. Також в заголовку HTTP-запиту є заголовок авторизації. Зверніть увагу, що дані повідомлення розташовуються в тілі HTTP-запиту. Повідомлення може зберігатися в текстовому або двійковому форматі, але при добуванні воно повертається base64-кодованим.

PUT http://sally.queue.core.windows.net/myqueue/messages
? messagettl = 3600
HTTP / 1.1 Content-Length 3900
Content-MD5: HUXZLQLMuI / KZ5KDcJPcOA ==
Authorization: SharedKey sally: F5a + dUDvef +
PfMb4T8Rc2jHcwfK58KecSZY + l2naIao =
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
Message Data Contents .....

Нижче показаний приклад REST-запиту для операції вилучення з черги. Зверніть увагу, що використовується HTTP-команда GET. Задані два необов'язкові параметри. «Numofmessages» визначає, скільки повідомлень

повинно бути вилучено з черги; максимальне число - 32. За замовчуванням витягується одне повідомлення. У прикладі нижче буде вилучатись по 2 повідомлення. Параметр «visibilitytimeout» визначає час очікування видимості; повідомлення буде залишатися невидимим в черзі, протягом цього проміжку часу, в секундах, і знову з'явиться в черзі, якщо не буде видалено до завершення періоду очікування видимості. Максимальне значення цього часу очікування - 2 години, і значення за замовчуванням - 30 секунд. У прикладі нижче час очікування видимості задано рівним 60 секундам. Також в заголовку HTTP-запиту є елемент авторизації. Зверніть увагу, що відповідь надходить в XML-форматі, і ці повідомлення у відповіді будуть base64-кодованими (в прикладі нижче розташовуються між тегами <MessageText> </ MessageText>).

GET http://sally.queue.core.windows.net/myqueue/messages
?numofmessages=2 &visibilitytimeout=60
HTTP/1.1
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw=
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT

Відповідь на цей вклик буде аналогічним, отримуємому в наступному прикладі:

HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 22fd6f9b-d638-4c30-b686-519af9c3d33d
Date: Thu, 13 Nov 2008 21:37:56 GMT
<?xml version="1.0" encoding="utf-8"?>
<QueueMessagesList>
<QueueMessage>
<MessageId>6012a834-f3cf-410f-bddd-dc29ee36de2a</MessageId>
<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD/////AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWUuU
2VydmljZ
S5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN1bHR1cmU9
bmV1dHJhbCwg
UHVibGljS2VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5DaXMuU2VydmljZXM
uTmVwaG9zLlF1
ZXVlLlNlcnZpY2UuQXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVldWVNYW5hZ2VyK
1JYVpcHQCAAAA
FjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWN
raW5nRmllbGQ
DAAtTeXN0ZW0uR3VpZA0CAAAABP3///8LU3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9j
Al9kAl9lAl9mAl
9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAIBwcCAgICAgICAjSoEmDP8w9Bvd3cK
e423ipfNapL7xPL

SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>
<MessageText>......</MessageText>
</QueueMessage>
<QueueMessage>
<MessageId>2ab3f8e5-b0f1-4641-be26-83514a4ef0a3</MessageId>
<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD/////AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWUuU
2VydmljZ
S5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN1bHR1cmU9
bmV1dHJhbCwg
UHVibGljS2VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5DaXMuU2VydmljZXM
uTmVwaG9zLlF1
ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVldWVNYW5hZ2VyK
1JYVpcHQCAAAA
FjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWN
raW5nRmllbGQ
DAAtTeXN0ZW0uR3VpZA0CAAAABP3///8LU3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9j
Al9kAl9lAl9mAl
9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAAIBwcCAgICAgICAuX4syrxsEFGviaDUU
pO8KNfNapL7xPL
SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>
<MessageText>.....</MessageText>
</QueueMessage>
</QueueMessagesList>

Нижче представлений приклад REST-запиту для операції видалення повідомлення. У цьому випадку використовується HTTP-команда DELETE. Параметр «popreceipt» визначає повідомлення, яке повинно бути видалено. «Popreceipt» виходить в результаті попередньої операції вилучення з черги, як показано в прикладі раніше.

DELETE /sally/myqueue/messages/6012a834-f3cf-410f-bddddc29ee36de2a?
popreceipt=AAEAAAD%2f%2f%2f%2f
%2fAQAAAAAAAAAMAgAAAFxOZXBob3Mu
UXVldWUuU2VydmljZS5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9uPTYuMC4
wLjAsIEN1bHR1c

mU9bmV1dHJhbCwgUHVibGljSV5VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5
DaXMuU2Vyd
mljZXMuTmVwaG9zLlF1ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWF
sUXVldWVNYW5
hZ2VyKJlY2VpcHQCAAAAFjxNc2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWx
pdHlTdGFydD5rX
19CYWNraW5nRmllbGQDAAtTeXN0ZW0uR3VpZA0CAAAABP3%2f%2f
%2f8LU3lzdGVtLkd1aWQL
AAAAAl9hAl9iAl9jAl9kAl9lAl9mAl9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAIBwc
CAgICAgICAjSoE
mDP8w9Bvd3cKe423ipfNapL7xPSAsAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA%3d&timeou
t=30
HTTP/1.1
Content-Type: binary/octet-stream
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT
Authorization: SharedKey sally:M/N65zg/
5hjEuUS1YGCbVDHfGnI7aCAudkuTHpCDvZY=

Питання для самоконтролю.

1. Охарактеризуйте програмне забезпечення хмарних обчислень.
2.Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація IT-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Що являє собою Платформа корпорації Майкрософт Windows Azure?
7. Що Ви отримуєте при використанні Azure?
8. Назвіть компоненти Платформи Windows Azure.
9. Назвіть інтерфейси прикладного програмування.
10. Що таке Windows Azure Storage?
11. Охарактеризуйте комплект засобів розробки Windows Azure.
12. Що таке Windows Azure Table?
13. Що таке Windows Azure Blob?
14. Що таке Windows Azure Queue?
15. Що таке «Інфраструктура як сервіс» (Infrastructure as a Service, або IaaS)?
16. Що таке «Платформа як сервіс» (Plaatform as a Service, або PaaS)?
17. Що таке «Програмне забезпечення як сервіс» (Software as a Service , SaaS)?

Література:

1.Windows Azure Platform Training Kit - January 2011 Update

2. http://microsoft.com/faculty

3.5 *Microsoft® .NET Services*

.NET Services надає основні стандартні блоки, які знадобляться при побудові додатків в хмарі і працюючих з хмарою для Azure ™ Services Platform. Сервіси, зібрані під ім'ям .NET Services, забезпечують інфраструктуру хмари, яка, в кінцевому рахунку, спрощує побудову працюючих в хмарі додатків. Сьогодні .NET Services забезпечують основну функціональність, пов'язану з можливостями підключення додатків, управління доступом і взаємодії за допомогою повідомлень на базі робочого процесу. Під іменем .NET Services об'єднані наступні основні блоки сервісів:

• *Microsoft® .NET Service Bus* - надає мережеву інфраструктуру для з'єднання додатків через Інтернет з використанням різноманітних шаблонів обміну повідомленнями способом, забезпечуючим можливість проходження міжмережевих екранів і NAT-пристроїв без порушення безпеки, яка надається цими пристроями.

• Microsoft® .NET Access Control Service - забезпечує управління доступом в хмарі на підставі тверджень. Він включає механізм перетворення тверджень, який об'єднується з постачальниками посвідчень, такими, наприклад, як Active Directory і Windows Live ID (WLID).

• Microsoft® .NET Workflow Services - надає інфраструктуру для розміщення та управління робочими процесами (WF), приділяючи основну увагу взаємодії через повідомлення за допомогою .NET Service Bus. Поставляється з новими діями WF і інструментами для розміщення та керування екземплярами робочого процесу.

Дані нові сервіси можна розглядати як .NET-інфраструктуру сервісів для хмари. Всі вони доступні через відкриті протоколи і стандарти, включаючи REST, SOAP, Atom / AtomPub і WS. Це означає, що розробники на будь-якій платформі можуть інтегруватися з цими сервісами. Однак, в спробі зробити все максимально звичним для .NET-розробників, Microsoft також надає .NET Services SDK, який забезпечує хороші умови для .NET-розробника і приховує багато складних моментів роботи з сервісами. .NET Services SDK дозволяє розробникам використовувати наявний досвід .NET- розробки, зокрема в областях WCF і WF, через застосування нових розширень інфраструктури SDK (наприклад, нових прив'язок, каналів і дії). SDK також включає підтримку інструментів Visual Studio для інтеграції з порталом Azure ™ Services Portal. Крім .NET Services SDK, сьогодні партнери Microsoft пропонують Java і Ruby SDK.

Щоб почати роботу з .NET Services, перейдіть на портал Azure ™ Services Platform за адресою http://azure.com і клацніть посилання «*Try It Now*». Ви перейдете на сторінку «*Register for Azure Services*» (Реєстрація для сервісів Azure), представлену на рис.3.5.1. На цій сторінці даються важливі посилання для скачування різних SDK, доступу до додаткових ресурсів і переходу на сайт Microsoft Connect, де можна зареєструватися для отримання коду запрошення.

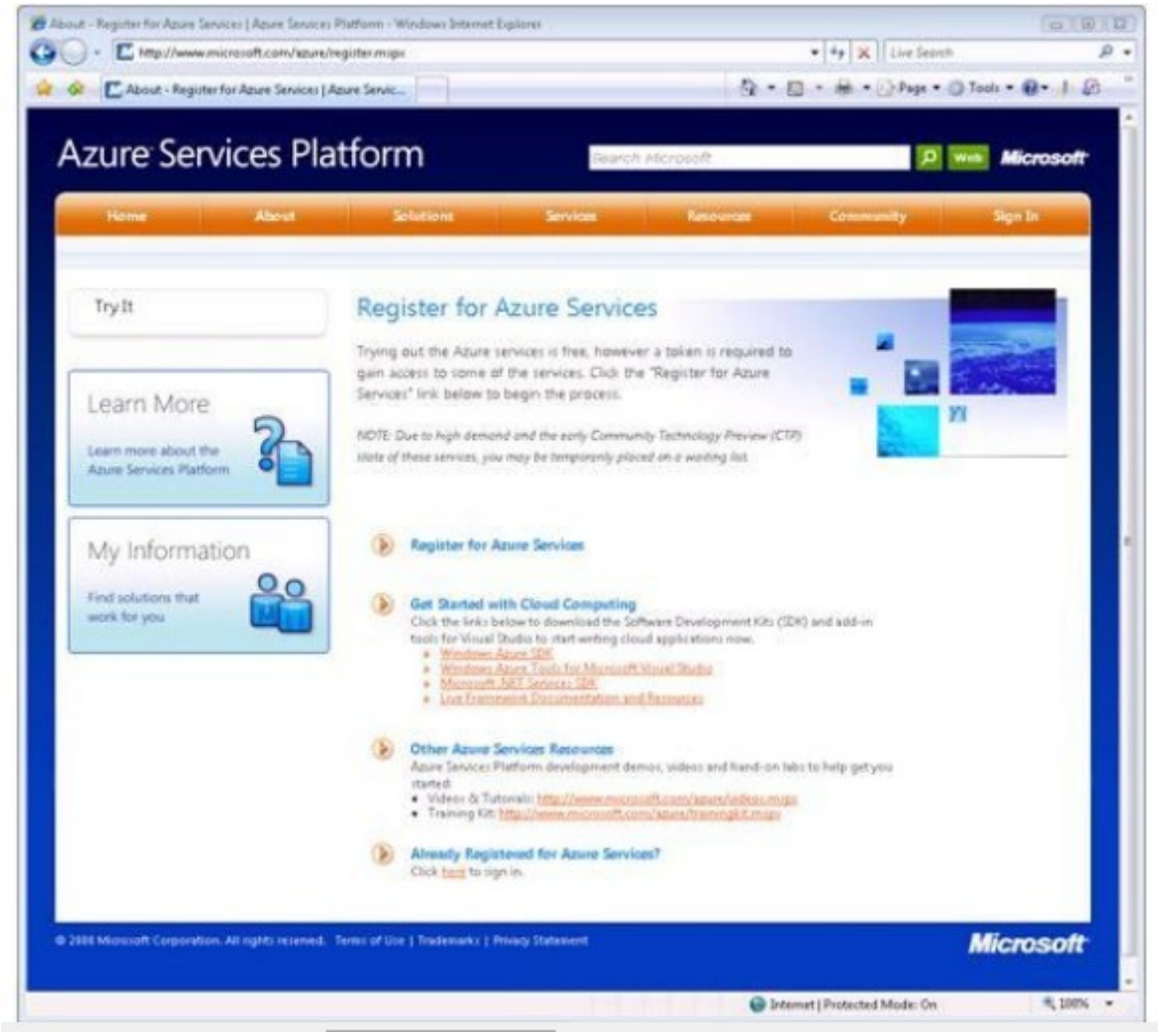


Рис.3.5.1 Портал Azure ™ Services Platform

Далі буде потрібно завантажити .NET Services SDK. Зверніть увагу, що є кілька SDK: один - спеціально призначений для розробки Windows® Azure ™; інший - для розробки .NET Services; і інші - для SQL Data Services і Live Framework. Для відтворення прикладів, пропонованих в даній серії документів, знадобиться завантажити і встановити .NET Services SDK.

Завантаживши .NET Services SDK, просто запустіть програму установки, як показано на рис.3.5.2. Тим самим Вам будуть доступні нові .NET-збірки, які разом з деякими надбудовами Visual Studio допоможуть розпочати використання різних функцій .NET Services. Приступаючи до роботи з .NET Services, обов'язково ознайомтеся з іншими ресурсами, доступними з цієї сторінки (демонстрації, відео, практичні лабораторні і т.д.), мета яких - зробити процес навчання більш насиченим і різноманітним. Завантажити SDK можна, не створюючи облікового запису, але, щоб використовувати сервіси, необхідно зареєструватися.

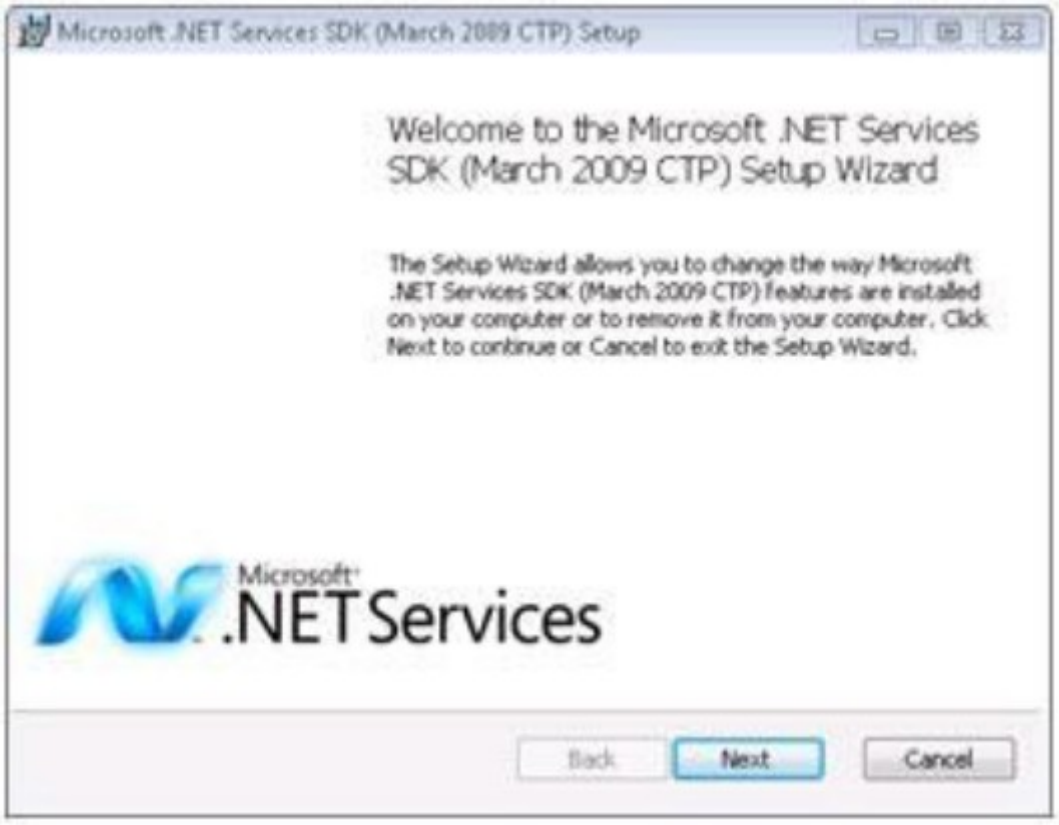


рис.3.5.2.Запуск установки .NET Services SDK

Щоб зареєструватися на отримання облікового запису Azure Services, клацніть посилання «*Register for Services*» (Реєстрація для сервісів). Вам буде запропоновано зареєструватися, використовуючи Windows Live ID (WLID). Після цього Ви перейдете на сайт Microsoft Connect, де потрібно заповнити реєстраційну форму Azure Services CTP. Після успішної реєстрації на Azure Services CTP, на екрані з'явиться сторінка, представлена на рис.3.5.3.

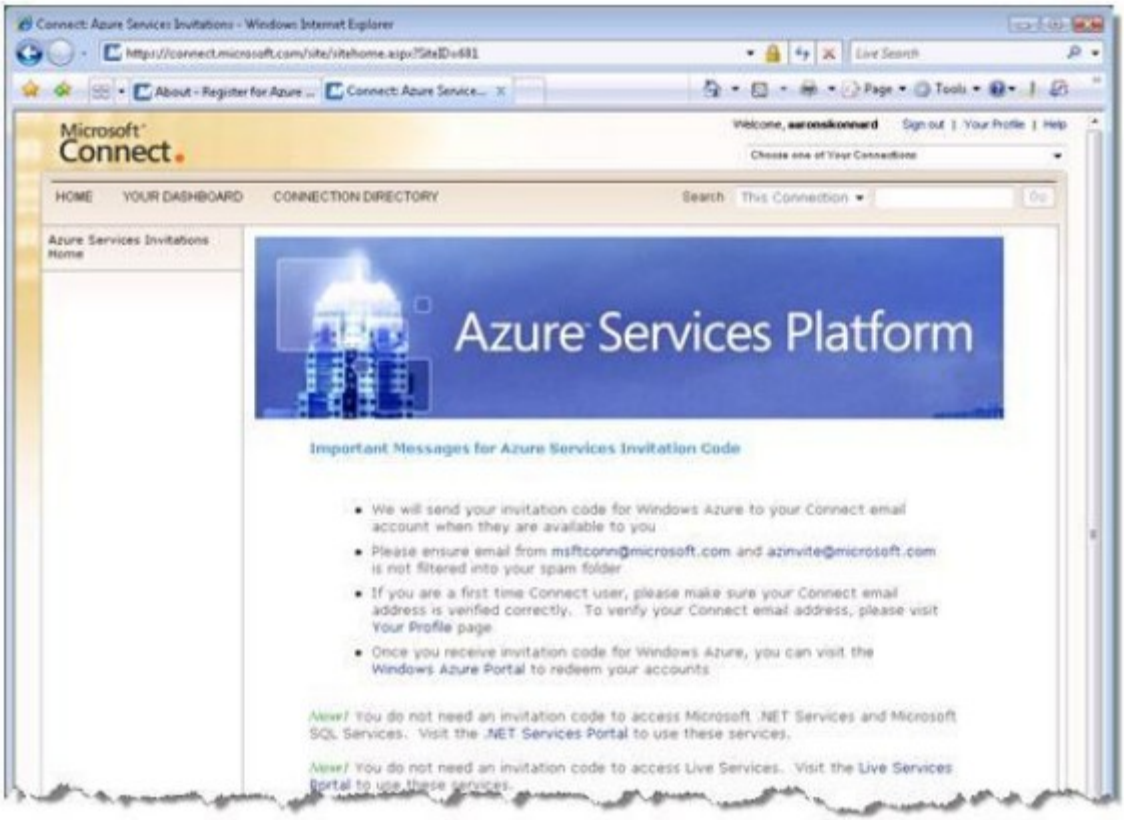


Рис.3.5.3 Реєстрація для Azure Services Platform на сайті Microsoft Connect

Тепер можна повернутися на сторінку входу .NET Services. Цю сторінку можна побачити на рис.3.5.4.

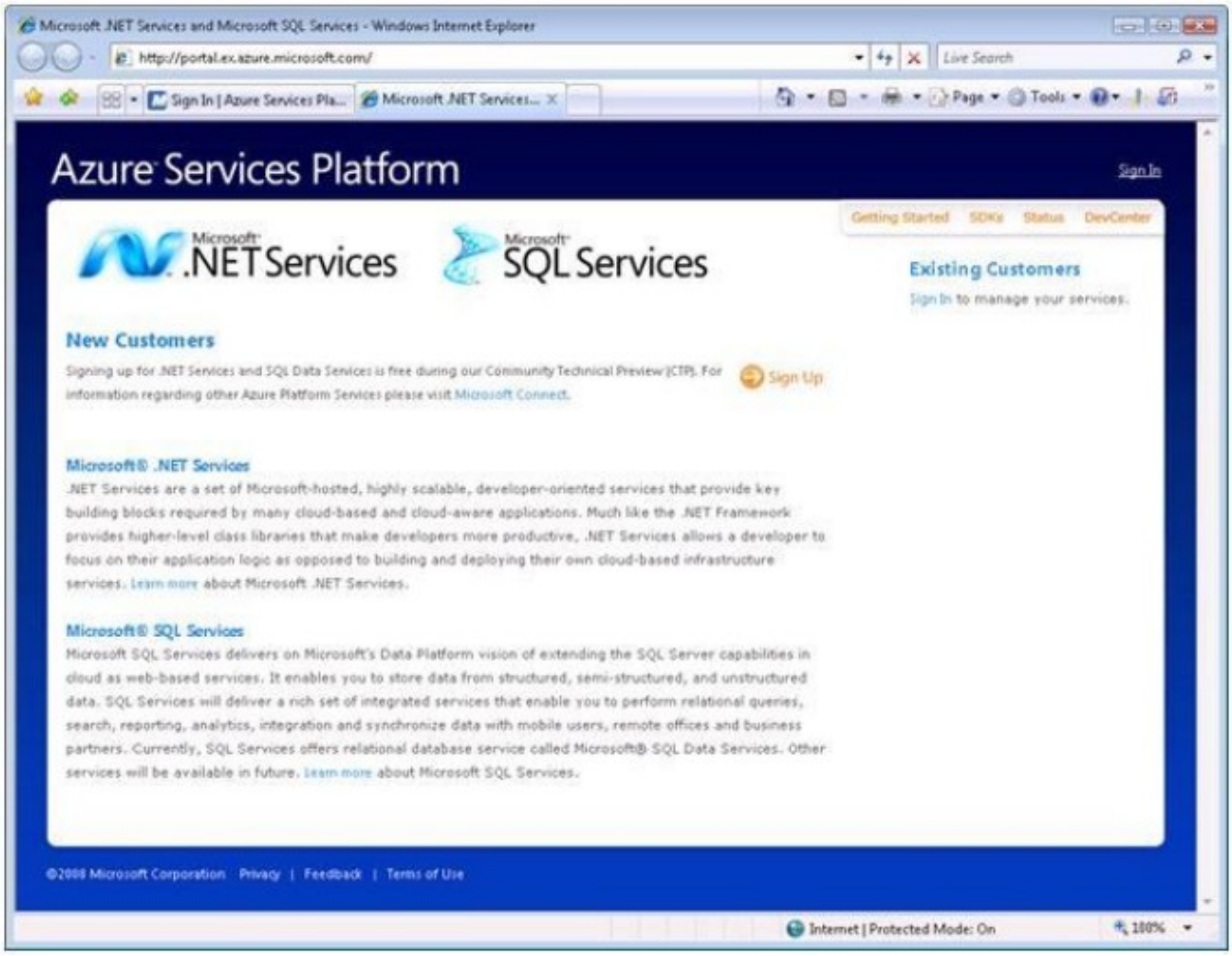


рис.3.5.4 Сторінка входу .NET Services

Тепер, клацнувши «*Sign Up*» (Ввійти), Ви отримуєте можливість створювати рішення .NET Services. Примітка: в CTP-версії для створення рішення .NET Services не потрібно код запрошення. Для створення рішення необхідно просто ввести унікальне ім'я рішення, прийняти умови використання і натиснути «*Create Solution*» (Створити рішення). Після цього нове рішення буде підготовлено і асоційоване з вашим WLID. Тепер в будь-який момент, зареєструвавшись на порталі Azure ™ Services Platform, Ви маєте можливість керувати рішеннями, асоційованими з вашим WLID. Ім'я рішення повинно бути не менше 6 символів довжиною і глобально унікальним серед всіх користувачів .NET-сервісів. Можливо, доведеться проявити кмітливість, щоб придумати таке ім'я для рішення, яке не використовується ніким іншим. Після того, як нове рішення створено, на екран виводиться сторінка (рис.3.5.5), яка пропонує пароль рішення, який бажано зберегти для використання в майбутньому. Ім'я рішення і пароль виступають в ролі облікових даних для доступу до різних сервісів .NET Services.

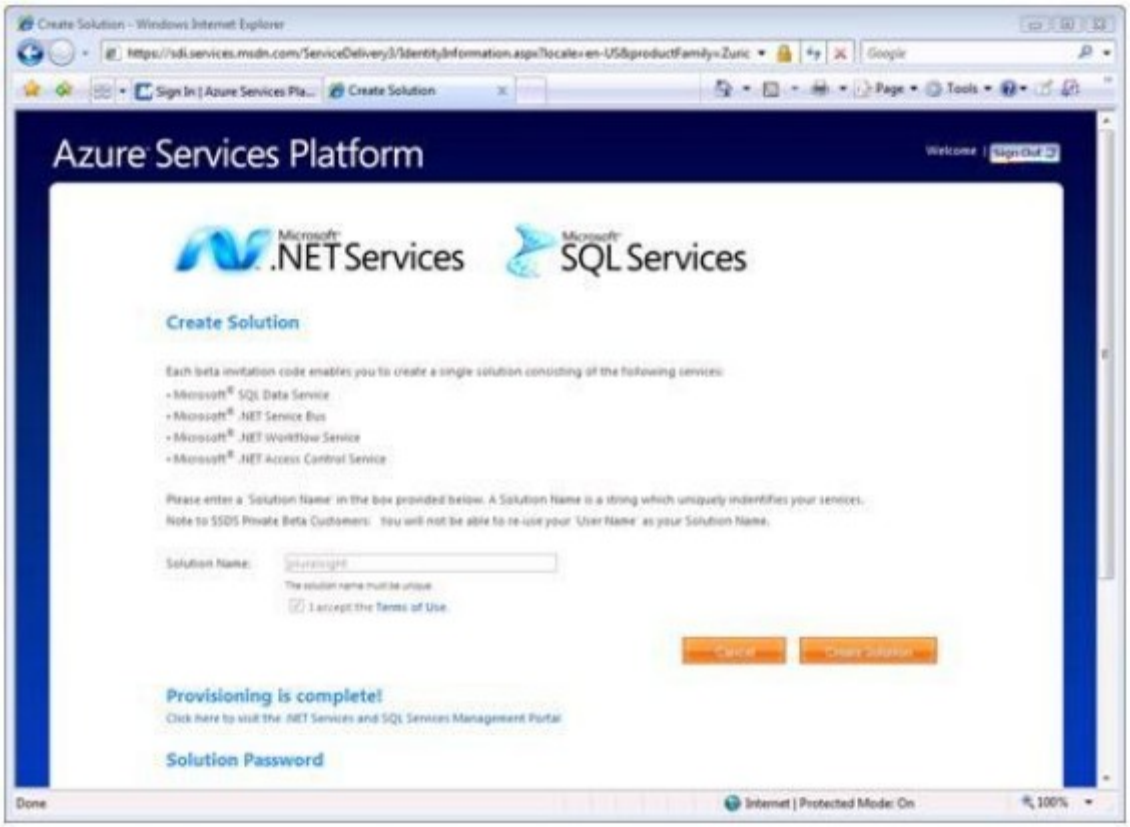


Рис.3.5.5 Завершення процесу підготовки рішення

Після успішного створення рішення можна приступати до роботи з ним на порталі Azure ™ Services Platform. Зареєструвавшись під власним WLID, на сторінці порталу справа Ви побачите меню «*My Solutions*» (Мої рішення) (рис.3.5.6). Для роботи з конкретним рішенням необхідно просто вибрати його в меню «*My Solutions*», після чого Вам буде представлена сторінка, показана на рис.3.5.7.

По суті, рішення - це контейнер верхнього рівня для організації різних ресурсів .NET Services. Наприклад, в ньому розміщуються кінцеві точки .NET Service Bus, типи і екземпляри робочих процесів .NET Workflow Service, а також ваші посвідчення .NET Access Control Service і правила перетворення тверджень. Але одним з найважливіших аспектів, яким Ви захочете керувати після створення власного рішення, є облікові дані рішення. Саме тому ім'я рішення повинно бути унікальним серед всіх користувачів.

Рис.3.5.6. Керування своїми рішеннями за допомогою меню «My Solutions»

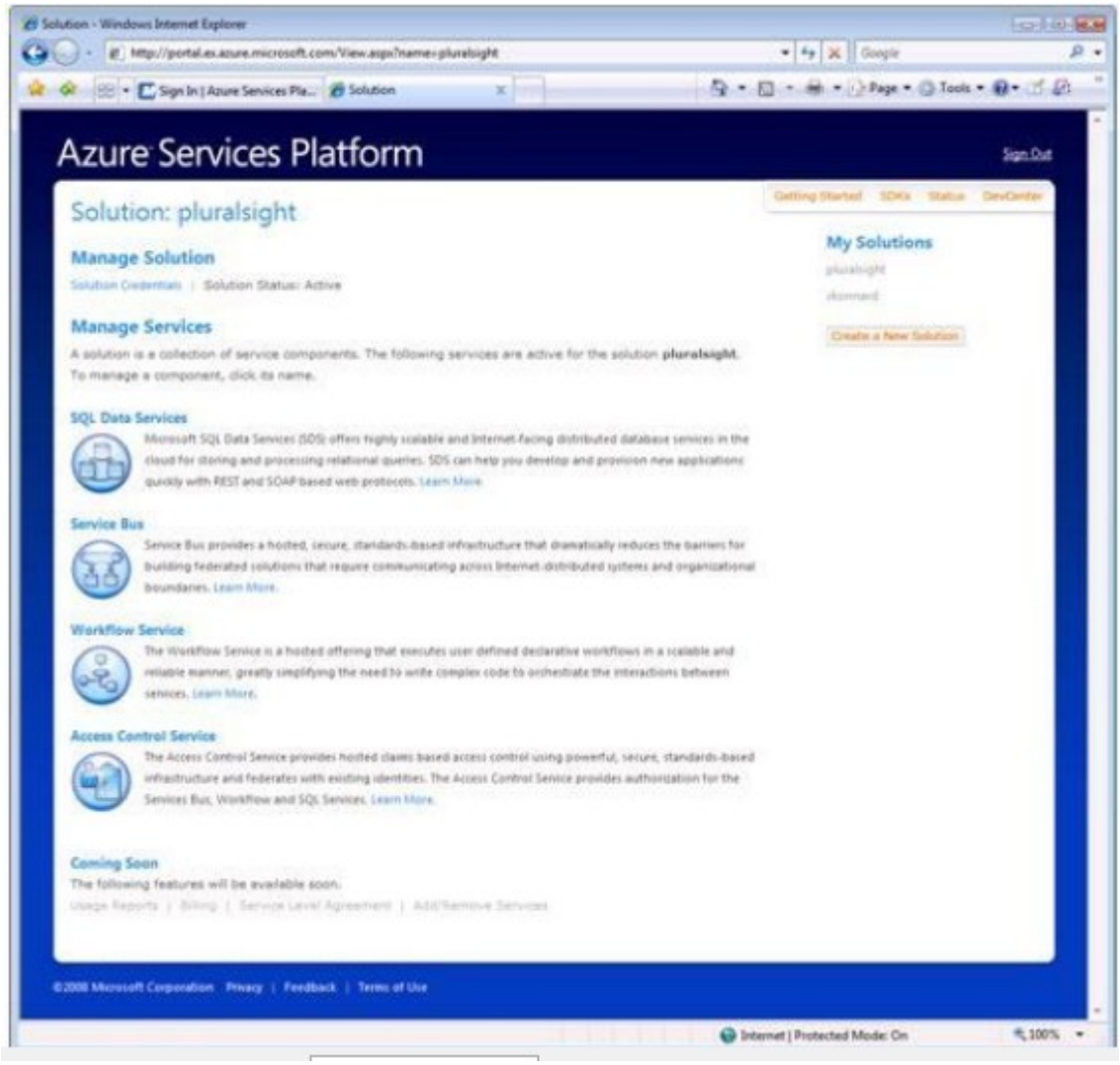


Рис.3.5.7 Керування окремим рішенням

Пароль рішення, наданий в процесі підготовки, можна змінити на сторінці «Credential Management» (Керування обліковими даними) (просто клацніть посилання «*Solution Credentials*» (Облікові дані рішення), яку можна

бачити на рис.3.5.8. З цієї сторінки можна також конфігурувати інформаційні картки Windows CardSpace, асоційовані з даним рішенням, і також будь-які сертифікати, які необхідно асоціювати з рішенням (рис.3.5.8).

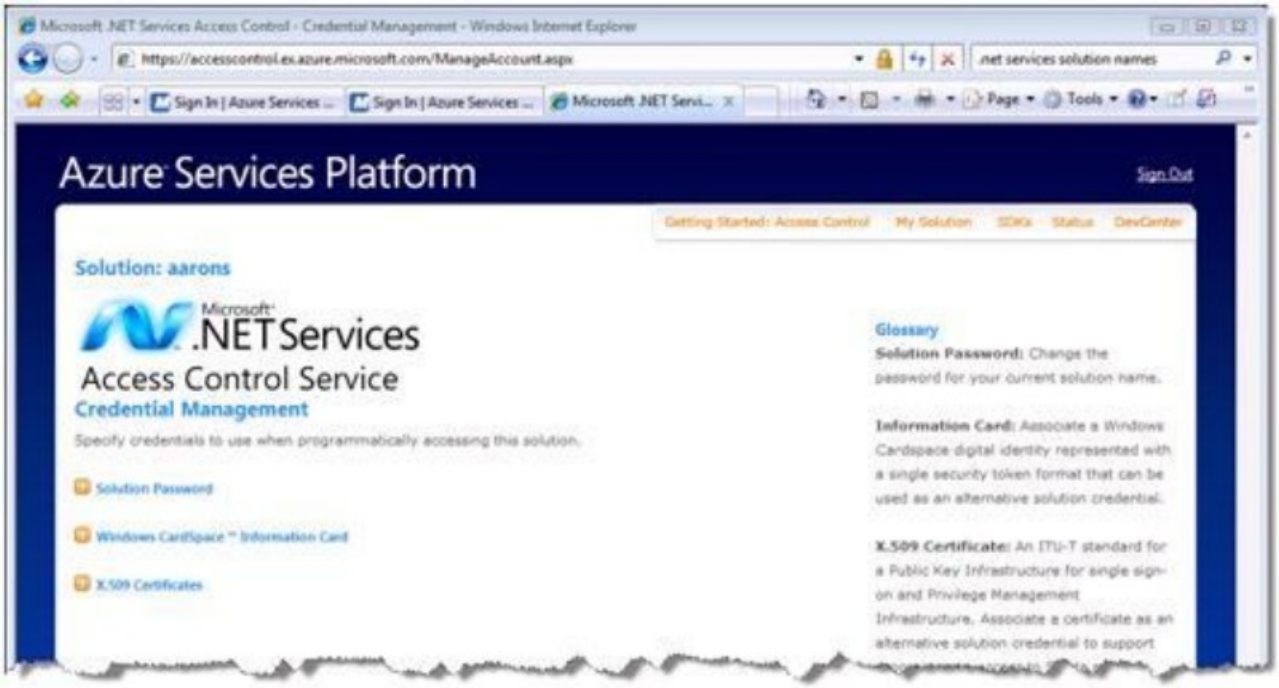


Рис.3.5.8 Керування обліковими даними свого рішення

Для Windows CardSpace і сертифікатів Вам запропонують вибрати необхідну картку / сертифікат, після чого відповідна інформація буде передана в обліковий запис Вашого рішення. З цього моменту в поєднанні зі своїм обліковим записом Ви можете використовувати облікові дані зазначеної картки / сертифікату. Найпоширенішою вимогою в розподілених додатках з високим рівнем масштабованості є можливість підключення додатків. Зазвичай інтеграція додатків - одна з найдорожчих і морочливих областей IT. Сьогодні для цих завдань багато організацій використовують рішення Enterprise Service Bus.

Enterprise Service Bus (сервісна шина підприємства, ESB) - підхід до побудови розподілених корпоративних інформаційних систем. Зазвичай включає в себе проміжне ПЗ, яке забезпечує взаємозв'язок між різними додатками з використанням різних протоколів взаємодії. Архітектура ESB полягає у взаємодії всіх додатків через єдину точку, яка, при необхідності, забезпечує транзакції, перетворення даних, збереження звернень. Даний підхід забезпечує більшу гнучкість, простоту масштабування і перенесення: при заміні одного додатку, підключеного до шини, немає необхідності переналпштовувати інші.

Одним зі стандартів взаємодії є веб-сервіси. У популярних реалізаціях ESB додаються шлюзи для обміну даними з корпоративним ПЗ. З використанням ESB може бути реалізована сервісно-орієнтована архітектура. Існує деяка розбіжність, що саме вважати ESB – архітектуру, чи програмне забезпечення. Обидві точки зору мають право на існування.

.NET Service Bus є основною частиною пропозиції .NET Services. Її основне завдання - зробити шаблон ESB реальністю в Інтернеті в рамках платформи Azure ™ Services Platform. Надані .NET Service Bus архітектурні характеристики багато в чому аналогічні запропонованим типовими рішеннями ESB, включаючи ідентифікацію та керування доступом, присвоювання імен, реєстр сервісу і загальне середовище обміну повідомленнями. Основна відмінність в області застосування. У випадку з .NET Service Bus компоненти повинні розроблятися для роботи в хмарі, в глобальній області Інтернету, із забезпеченням високого рівня масштабованості і інтегрованості. Саме тому в минулому цей пропонований сервіс називався Microsoft Internet Service Bus (рис.3.5.9).

Internet Service Bus дозволила б інтегрувати Ваш локальний ESB-продукт з вашими власними хмарними сервісами, з різними сторонніми сервісами, наданими Microsoft, або іншими виробниками (такими, як пропонуються в рамках платформи Azure ™ Service Platform) і з різними настільними, RIA3 і веб- додатками, які можуть виконуватися на допоміжних майданчиках поза брандмауера корпорації. Щоб це стало реальністю, реалізація повинна забезпечувати інтегровані рішення, засновані на відкритих Інтернет-стандартах, і насичене середовище обміну повідомленнями з можливістю двостороннього зв'язку в Інтернет.

1. Сервісна шина підприємства.
2. Ця термінологія застосовувалася в документації BizTalk Services, але більш не є офіційним найменуванням, використовуваним Microsoft.
3. RIA = Rich Internet Application (Насичений Інтернет- додаток).

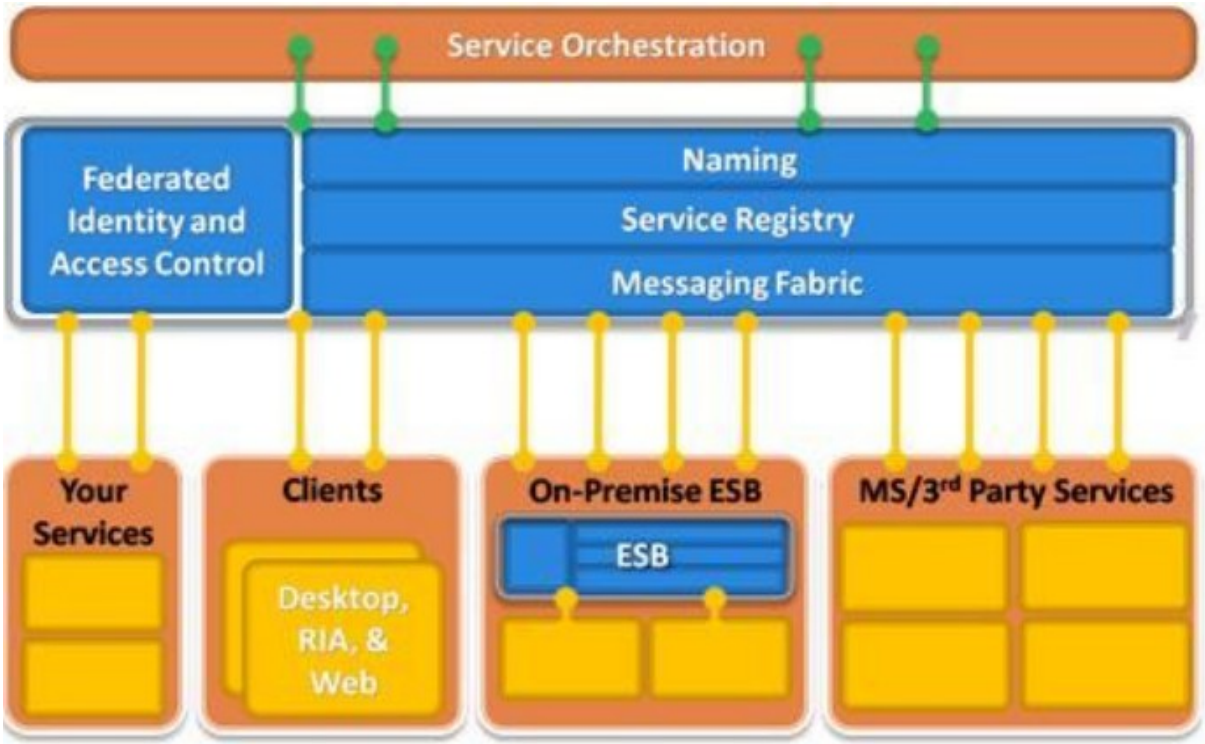


Рис.3.5.9 Сервісна шина Інтернет

Реалізація двостороннього зв'язку в Інтернеті не таке вже тривіальне завдання із-за деяких реалій організації сучасних мереж. Переважно бар'єри в

мережі створюють міжмережеві екрани і пристрої, які працюють по протоколу NAT, і які ускладнюють обмін інформацією з вузлами, розташованими за ними. Уявіть ситуацію: торговий агент використовує Ваш додаток по бездротовій мережі в випадковій готелі в деякій точці земної кулі. Як при такому сценарії визначити його місце розташування і ініціювати зв'язок?

Часто компанії вирішують ці проблеми зв'язку, відкриваючи вхідні порти міжмережевих екранів (що завдає чимало клопоту системним адміністраторам), або використовуючи різні обхідні прийоми, такі як динамічна DNS, зіставлення портів NAT , або технологію UpnP. Всі ці методи нестійкі, важко керовані і сприйнятливі до загроз безпеки. Кількість додатків, для яких потрібно такий тип двостороннього зв'язку, стає дедалі більше. .NET Service Bus покликана задовольнити цю потребу.

Network address translation - Перетворення мережевих адрес.

Domain Name System - Служба доменних імен.

Universal Plug and Play – Універсальне автоматичне налаштування мережевих пристроїв.

Рішення ідентифікації, реалізацією якої Microsoft займається останні кілька років, засноване на концепції тверджень. Модель ідентифікації на базі тверджень дозволяє виносити загальні функції аутентифікації і авторизації з додатків і здійснювати їх централізовано в зовнішніх сервісах, написаних і обслуговуюємих експертами безпеки та ідентифікації, що вигідно всім, хто бере участь в цьому процесі.

Microsoft® .NET Access Control Service - це сервіс в хмарі, який виконує саме цю функцію. Замість того, щоб створювати власну базу даних користувацьких облікових записів і ролей, можна надати можливість .NET Access Control Service керувати аутентифікацією і авторизацією Ваших користувачів. .NET Access Control Service використовує існуючі сховища облікових записів користувачів, такі як Windows Live ID і Active Directory, а також будь-які інші сховища, які підтримують стандартні протоколи інтегрування. Таким чином, використання єдиної реєстрації для доступу до всіх програм стає цілком природним. Також це забезпечує централізацію логіки аутентифікації і керування доступом, що спрощує Ваші додатки.

В підтримуючих твердження додатках користувач представляє своє посвідчення як набір тверджень. Одним твердженням може бути ім'я користувача; іншим - його адреса електронної пошти. Ці твердження надаються організацією, яка видає посвідчення, яка знає, як аутентифікувати користувача і де знайти його атрибути. Клієнтський додаток, в ролі якого може виступати браузер, або насичений клієнт, безпосередньо отримує твердження від цієї організації і передає їх в Ваш додаток (рис.3.5.10).

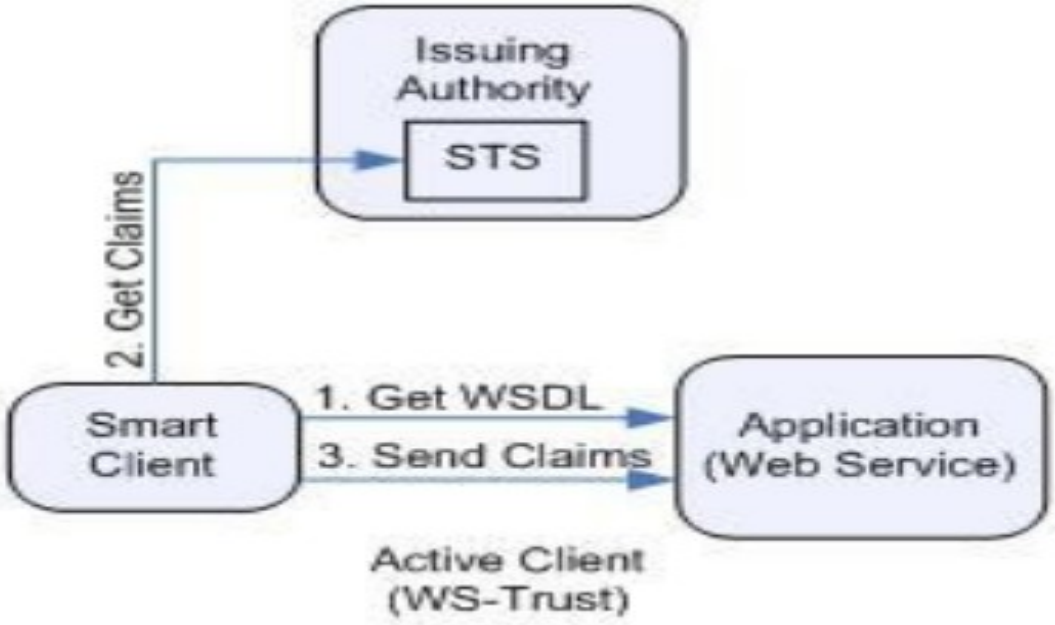


рис.3.5.10: Використання ідентифікації на базі тверджень для веб-сервісів

У підсумку додаток отримує всі відомості, необхідні для ідентифікації користувача, у вигляді набору тверджень. Ці твердження підписуються, що забезпечує криптографічне підтвердження їх походження. Модель ідентифікації на базі тверджень спрощує реалізацію єдиної реєстрації, при цьому додаток більше не відповідає ні за один з перерахованих нижче аспектів безпеки:

• Аутентифікація користувачів
• Зберігання облікових записів користувачів і паролів
• Звернення до каталогів підприємства в пошуках даних посвідчення користувача
• Інтеграція з системами посвідчень інших платформ або компаній

Використовуючи таку модель, додаток може приймати рішення про ідентифікацію на підставі наданих користувачем тверджень. І діапазон таких рішень великий: від простої персоналізації додатку по імені користувача до авторизації користувача для доступу до особливо важливих функцій і ресурсів додатку.

.NET Access Control Service реалізовує ідентифікацію на базі тверджень в рамках платформи Azure ™ Services Platform. Система адміністрування є важливою частиною .NET Access Control Service.

.NET Access Control Service надає портал адміністрування (рис.3.5.11) в рамках порталу Azure ™ Services Portal. Тут Ви виконуєте налаштування правил, які визначають схему випуску тверджень для різних користувачів.

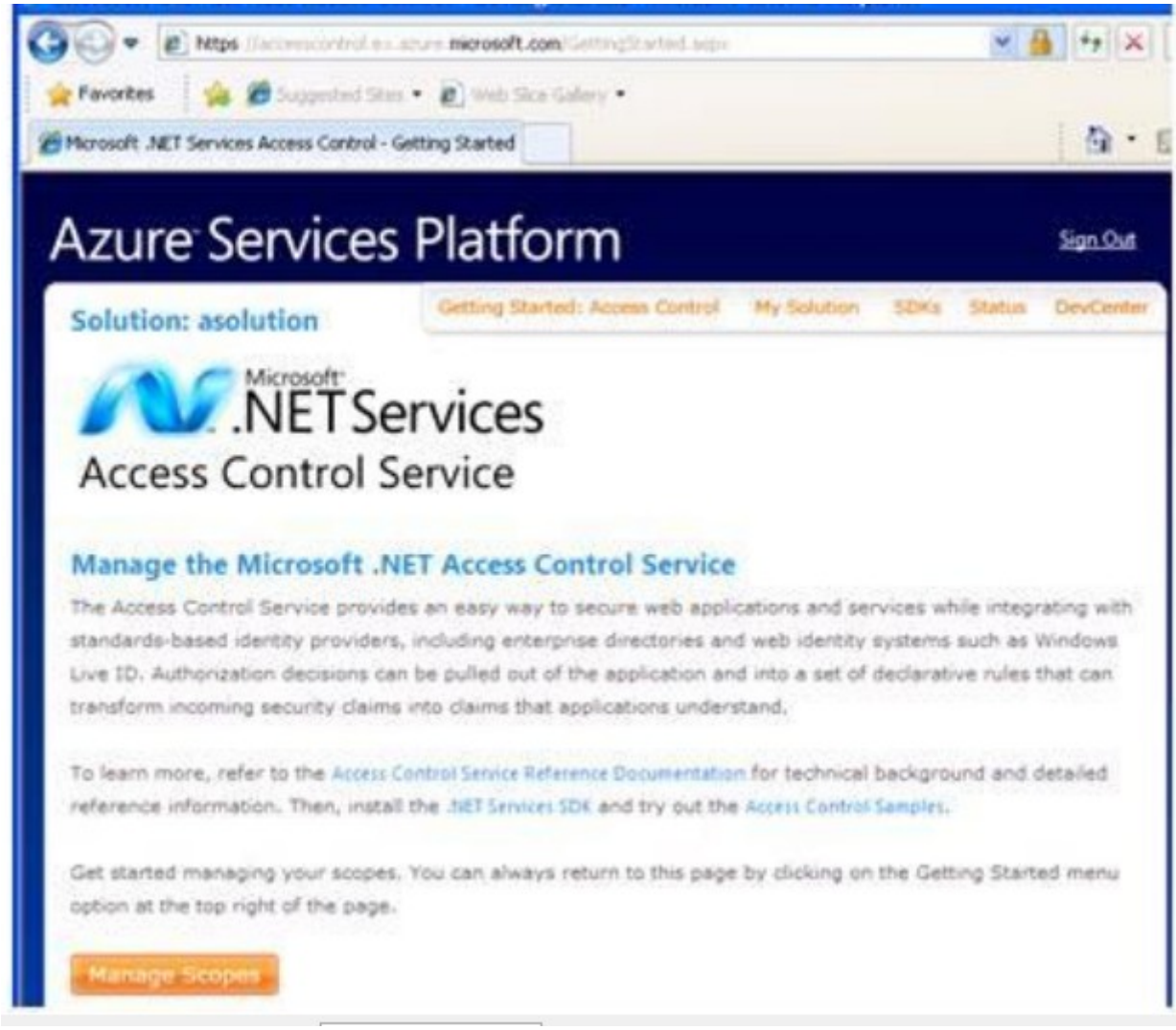


Рис.3.5.11. Портал ACS

Портал Access Control Service - хороший засіб для дослідження, вивчення і початку роботи з ACS. І для відносно простих додатків він може бути єдиним необхідним інструментом. Але для нетривіальних систем з сотнями або тисячами користувачів і, можливо, такою ж кількістю правил, використання порталу стає громіздким. У таких випадках програмний інтерфейс - більш кращий варіант, тому ACS також надає інтерфейс AtomPub для програмного адміністрування. AtomPub - це протокол RESTful, який стандартизує базові операції CRUD (Create, Retrieve, Update і Delete) для керування віддаленими ресурсами. Це відкриває абсолютно нові можливості. Сервіс .NET Access Control Service також включає кінцеві точки SOAP і REST для програмного адміністрування, а також ряд .NET-класів, які спрощують виклик цих кінцевих точок. Отже, якщо Вам не подобається портал, який надається ACS, або Ви бажаєте реалізувати налаштування, характерні для певної предметної області, можна створити власну консоль адміністрування.

Найбільшою проблемою в побудові великомасштабних розподілених додатків є прийняття рішення про моделювання складних схем взаємодії через обмін повідомленнями. Microsoft .NET Workflow Service дозволяє розробляти логіку взаємодії повідомлень за допомогою WF і забезпечує розміщене масштабіруєме середовище для виконання і керування екземплярами робочого

процесу WF в хмарі, звільняючи розробника від необхідності створення власного хоста для WF.

.NET Workflow Service є частиною Azure ™ Services Platform і інтегрується з сервісами .NET Service Bus і .NET Access Control Service для безпечного координування взаємодії за допомогою повідомлень. .NET Workflow Service також забезпечує інструменти керування для створення і керування типами та примірниками робочих потоків і API веб-сервісів для ситуацій, коли потрібно створити власні інструменти.

Оскільки керуюче середовище побудоване на платформі Windows® Azure ™, воно може масштабуватися на вимогу і в значній мірі при цьому організації, або розробнику не доводиться турбуватися про планування більшої кількості обладнання, або програмного забезпечення. Завдяки використанню середовища виконання WF екземпляри робочого потоку можуть виконуватися в пулі серверів і переміщуватися з одного сервера на інший в кожному епізоді виконання. Керуюче середовище включає сервіс зберігання, який використовує безпечні тиражовані сервіси Microsoft SQL Service для збереження стану виконуючихся робочих процесів і для забезпечення можливості відновлення. На період переходу до обробки даних в хмарі .NET Workflow Services надає спрощений підхід для керування складними взаємодіями .NET Service Bus в створюваних Вами складових рішеннях «в хмарі».

Побудова хоста для робочих процесів WF означає прийняття рішень про те, які можливості підтримуватиме середовище та як краще зробити її безпечною, масштабованою і стабільною. Сьогодні .NET Workflow Service побудований на .NET Framework 3.5 і діях і компонентах WF, які входять в дану версію інфраструктури. Однак для забезпечення найкращих умов Microsoft були додані декілька спеціальних дій і сервісів, які наклали деякі обмеження на робочі процеси, які виконуються в хмарі.

В хмарі не використовується сервіс зберігання *SqlWorkflowPersistenceProvider*, який отримав найбільшу популярність серед розробників, які використовують WF. Щоб використовувати операційне середовище Azure і забезпечити найкращі можливості масштабування і стабільності, в інфраструктурі хмари є спеціальний провайдер послуг зберігання, який реалізує збереження стану виконуючихся робочих процесів за допомогою можливостей зберігання Microsoft SQL Services. Крім усього іншого, для Інтернет-сервісу необхідна Інтернет-технологія зберігання та вилучення даних. Але оскільки механізм WF єдиний - як в хмарі, так і в Ваших локальних рішеннях, - застосування спеціального провайдера послуг зберігання прозоро для розробників робочих процесів. Все робиться так саме, як в будь-якому іншому середовищі WF.

При побудові робочих процесів для хмари розробники використовують звичні інструменти Visual Studio, включаючи той же дизайнер робочого процесу для створення XML-файлів робочих процесів і файлів правил. Потім ці XML-файли завантажуються на сервер в хмарі, де вони можуть використовуватися для створення екземплярів робочого процесу. .NET Services SDK включає шаблон проекту для створення *SequentialCloudWorkflow* (Послідовний робочий процес в хмарі), який є спеціальною версією

стандартного шаблону *SequentialWorkflow* (Послідовний робочий процес). Одним з обмежень поточної інфраструктури є те, що при визначенні робочих процесів, які будуть виконуватися в хмарі, можна використовувати тільки підмножину дій базової бібліотеки дій, а також комплект спеціальних дій, що надається як частина .NET Services SDK.

Набір дій вимагає, щоб робочі процеси були повністю декларативними і обмежуючими. Це запобігає введенню користувацького коду, тобто дозволяє гарантувати стабільність середовища. При побудові керуючого середовища для робочих процесів, написаних будь-якою кількістю розробників, розкиданих по всьому світу, такий рівень контролю є обов'язковим. Оскільки сьогодні для виконання WF необхідно повна довіра, ми не можемо забезпечити обмежений набір функціональності, просто виділивши користувальницький код в безпечне ізольоване програмне середовище на серверах. Слід зазначити, що з часом доступний сьогодні обмежений набір дій буде розширено для збільшення можливостей робочого процесу в хмарі. У міру виходу нових версій .NET Framework і .NET Workflow Service також буде підтримувати їх. Крім того, якщо знадобляться спеціальні етапи, можливості локальних робочих процесів можуть комбінуватися з робочими процесами в хмарі за допомогою .NET Service Bus.

При написанні робочих процесів в хмарі необхідно бути акуратним з використовуваними діями (дизайнер запропонує тільки допустимі дії). Дозволеними є деякі основні дії потоку керування WF, включаючи *IfElse* (Якщо ... то), *While* (Поки), *Sequence* (Послідовність), *Suspend* (Призупинити), *Terminate* (Завершити) і *FaultHandler* (Оброблювач збоїв). Крім базових дій потоку керування, в робочих процесах в хмарі можуть також використовуватися *CancellationHandlerActivity* і *FaultHandlersActivity* для моделювання обробки виключень і логіки скасування. Зверніть увагу, що ці дії зазвичай не додаються в модель безпосередньо; для цього використовується дизайнер складових дій, який вводить їх автоматично, коли уявлення переходить до цього дійства.

Оскільки основним завданням .NET Workflow Service є спрощення взаємодії повідомлень, ці дії, головним чином, стосуються відправки, отримання та обробки повідомлень. Відправляти / приймати повідомлення можна за допомогою традиційних HTTP-запитів, або через .NET Service Bus. Ці дії можна буде знайти на панелі інструментів Visual Studio при використанні шаблону проекту послідовного робочого процесу в хмарі.

Отже, платформа Azure ™ Services Platform представляє комплексну стратегію, розроблену Microsoft для полегшення розробникам завдань щодо реалізації можливостей обробки даних в хмарі. Microsoft® .NET Services - ключова складова цієї платформи, створена спеціально, щоб допомогти .NET-розробникам зробити перший крок. .NET Services пропонує орієнтовані на роботу в хмарі стандартні блоки і інфраструктуру для забезпечення можливості підключення додатків, керування доступом, розміщення та керування робочим процесом. Ці стандартні блоки стануть основними засобами організації роботи «з хмарою» для .NET-розробників і в майбутньому. Більше інформації

про .NET Service Bus, .NET Access Control Service і .NET Workflow Service можна знайти в документах даної серії, присвячених кожній з цих тем окремо.

Приклади хмарних сервісів Microsoft і Google пропонується засвоїти в рамках самостійної роботи студента.

Питання для самоконтролю.

1. Office Live Workspac – призначення, вимоги до комп'ютера, доступність, безпечність, функціональність.
2. Склад набору Office Web Apps.
3. Додаток Microsoft Word Web App.
4. Компоненти Microsoft Silverlight.
5. Додаток Microsoft Excel Web App.
6. інтернет-сервіс SkyDrive.
7. Пакет послуг Microsoft Office 365.
8. Середовище Google Apps – склад і функції.
9. Поштовий сервіс від Google: архівація пошти, фільтрація спаму, можливості пошуку по всіх поштових повідомленях, створення фільтрів, пересилка пошти.
10. Основні можливості календаря в Google Apps.
11. Редактори Google Docs & Spreadsheets.
12. Google App Engine.
13. Сховище даних App Engine.
14. Служба Memcache.
15. Інструментарій розробки App Engine (SDK).
16. Що являє собою Microsoft® .NET Service Bus?
17. Що являє собою Microsoft® .NET Access Control Service?
18. Що являє собою Microsoft® .NET Workflow Services?
19. Що таке .NET Services SDK?
20. Що таке Microsoft® .NET Access Control Service?
21. Що таке Network address translation?
22. Що таке Domain Name System?
23. Що таке Universal Plug and Play?

Література:

1. Аарон Сконнард (Aaron Skonnard) http://msdn.microsoft.com
2. http://windows.azure.com
3. http://microsoft.com/faculty
4. Практичні матеріали Azure SDK
5. http://help.live.com
6. http://www.ixbt.com/soft/msolive-1.shtml
7. http://www.ixbt.com/soft/msolive-2.shtml
8. http://www.microsoft.com/rus/newscenter/news/read/?id=post/2010/10/19/Microsoftd0bfd180d0b5d0b4d181d182d0b0d0b2d0bbd18fd0b5d182-Office-365.aspx
9. http://www.google.com/apps