ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи
на ступінь вищої освіти магістр

на тему: «ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОРКЕСТРАЦІЇ ЗАВДАНЬ В СЕРЕДОВИЩАХ З ПЕРИФЕРІЙНИМИ ОБЧИСЛЕННЯМИ»

Виконав: студент 6 курсу, групи ПДМ-61
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Конішевський В.І.
(прізвище та ініціали)

Керівник Негоденко О.В.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Магістр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
_____ Негоденко О.В.
"____" _____2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

КОНІШЕВСЬКОМУ ВЛАДИСЛАВУ ІГОРОВИЧУ

(прізвище, ім'я, по батькові)

1. Тема роботи: «Підвищення ефективності оркестрації завдань в середовищах з периферійними обчисленнями»

Керівник роботи: Негоденко О.В., , к.т.н., доцент, завідувач кафедри Інженерії програмного забезпечення

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «12» жовтня 2022 року №122.

2. Строк подання студентом роботи «31» грудня 2022 року

3. Вхідні дані до роботи: Науково-технічна література з питань, пов'язаних з побудовою та проектуванням сучасних систем оркестрації, особливості їх розгортання у середовищах з використанням периферійних обчислень

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

4.1 Аналіз наявних систем оркестрації завдань

4.2 Дослідження сучасних методик та реалізацій оркестраційної бази системи, її склад, архітектура компонентів, рівень спроможності обробки великих масивів даних

4.3 Побудова методики підвищеної ефективності оркестративного середовища, основаної на модифікації модулів обробки, передачі даних та автоматизації механізму підключень приладів в мережі

5. Перелік демонстраційного матеріалу (назва основних слайдів)

1. Оркестраційний елемент системи
2. Рівень обробки завдань. Використання нечіткої логіки в процесі обробки завдань
3. Рівень запиту. Внутрішня модель роботи gRPC
4. Модель розгортання власної мережі оркестратора
5. Рівень обробки завдань. Результати емуляції потоку завдань

Дата видачі завдання «14» жовтня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір технічної літератури	16.10-18.10	Виконано
2	Вимоги півищення оркестації завдань	19.11-1.11	Виконано
4	Аналіз сучасних методик піходів оркестрації	2.11-10.11	Виконано
5	Побудова методики підвищення ефективності оркестрації завдань в середовищах з периферійними обчисленнями	11.11-29.11	Виконано
6	Реферат, Вступ, висновки	1.12-04.12	Виконано
7	Розробка демонстраційних матеріалів	4.12-5.12	Виконано
8	Попередній захист роботи	21.12	
9	Здача роботи	17.01	

Студент _____ Конішевський В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 71с. 14 рис., 11 джерел

Об'єкт дослідження: Процес оркестраціїї завдань в системах з периферійними обчисленнями.

Предмет дослідження: методика підходів щодо окрестрації та механізмів, що її обслуговують на високих рівнях

Мета роботи: Дослідити сучасні системи з периферійними обчисленнями, удосконалити підходи щодо їх проектування та підтримки в майбутньому

Методи дослідження – аналіз, порівняння, імітаційне моделювання

В роботі було проведено аналіз існуючих оркестраційних рішень , що використовуються в різних сферах обчислень, починаючи з простого управління потоком обробки завдань, закінчуючи обробкою завдань, що використовуються в середовищах з використанням машинного навчання.

Здійснено аналіз життєвого циклу даних, специфіку роботи та застосування пристроїв що передають дані до оброблювача даних, проблематику оркестрації для систем нового покоління.

Галузь використання – системи, що використовують обробку завдань в середовищах з периферійними обчисленнями

Дослідження полягає у розробці теоретико-методичних та практичних рекомендацій, що дозволять спроектувати та розробити систему, спроможну ефективно оркеструвати завдання, швидко транспортувати дані між модулями або сервісами на рівні запиту та автоматично конфігурувати підключення пристроїв між собою, розділяючи мережу на групи.

ЗМІСТ

ВСТУП

Ообґрунтування теми та її актуальність: зростання обсягів даних залишається безпрецедентним, причому вони надходять з безлічі джерел. Дедалі складніші переміщення даних по великому різноманіттю екосистем істотно ускладнюють управління, і в майбутньому ця тенденція збережеться. Однак дані - такий самий важливий актив, як корпоративне майно та інтелектуальна власність. І що більше даних, то складніше ними керувати. Тому сучасним розподіленим системам потрібні нові методи переміщення та оркестрації даних.

Сутність вивчення проблеми: сьогоднішні оркестраційні елементи в цілому є частиною систем хмарних обчлислень та знаходяться виключно в зовнішній мережі і ніяк не зв’язані з самими пристроями які збирають необхідну інформацію. Але в деяких випадках є нагода обробки даних лише у внутрішній, закритій мережі, де після отримання дані обробляються, та надсилаються на сервер та не потребують їх подальшу модифікацію. Такі рішення не завжди мають узагальненого рішення, на базі якого персоналізувати конфігурацію пристроїв відповідно своїх потреб, та не мають в своєму арсеналі широкий спектр функціоналу, що міг би задовольнити кінцевого користувач.

Підхід централізованого управління потоками обробки даних, є одним із прогресивних течій в галузі роботи з даними. Такі рішення можуть використовуватись в більшості сферах, які в процесі роботи генерують та оброблюють великі трафіки інформації. Особливо важливим пунктом існування таких даних є контроль та правильність їх маршрутизації. Існують системи, в роботі яких необхідним фактором організації безпеки даних є модулі внутрішньої організації мережі , що не мають напряму контактувати з зовнішніми мережами. Оскільки в таких системах життєвий цикл даних має бути узгоджений з унікальною будовою мережі, в ній можуть бути використані такі компоненти, фреймворки та архітектурні підходи, що можуть сповільнити весь процес роботи даних в внутрішній мережі.

Мета роботи: Дослідити сучасні системи з периферійними обчисленнями, удосконалити підходи що стосуються їх проектування та підтримки в майбутньому.

Об'єкт дослідження: Процес наповнення, передачі, обробки завдань в системах з периферійними обчисленнями.

Предмет дослідження: моделі та методики підходів щодо окрестрації та її внутрішніх механізмів, які обслуговують систему на різних рівнях

1. АНАЛІЗ ІСНУЮЧИХ ОРКЕСТРАЦІЙНИХ РІШЕНЬ

1.1 Огляд існуючих систем оркерстрацій

Оркестрація - це координація та управління декількома комп'ютерними системами, додатками та/або послугами, об'єднання декількох завдань з метою виконання більшого робочого процесу або процесу. Ці процеси можуть складатися з декількох завдань, які автоматизовані і можуть включати декілька систем.

Метою оркестрування є впорядкування та оптимізація виконання частих, повторюваних процесів і, таким чином, допомога командам з обробки даних легше керувати складними завданнями та робочими процесами. Щоразу, коли процес повторюється, а його завдання можуть бути автоматизовані, оркестрування можна використовувати для економії часу, підвищення ефективності та усунення надмірностей.

Оркестрування даних є відносно новою дисципліною в комп'ютерній інженерії, особливо з огляду на те, що оркестрування даних пов'язане з хмарними обчисленнями та зберіганням даних. Концепція управління даними таким чином, щоб об'єднати правильні дані для правильної мети, була темою системного адміністрування протягом десятиліть, хоча і не настільки ефективно, як вважалося спочатку.

В основі інфраструктури оркестрування даних лежить створення конвеєрів даних і робочих процесів для переміщення даних з одного місця в інше, координуючи при цьому об'єднання, перевірку і зберігання цих даних для того, щоб зробити їх корисними. У перші дні системного адміністрування інженери та програмісти використовували інструмент під назвою "cron", утиліту в системах Linux, яка дозволяла їм планувати такі завдання, як передача даних з різних місць.

Очевидно, що зі зростанням складності системи та потреб у даних, створення складних завдань cron ставало все більш і більш складним завданням,

майже самостійною дисципліною. Ці ранні форми ручної оркестрації даних страждали від таких проблем, як:

Залежності між різними завданнями повинні були оброблятися вручну, що означало трудомісткі і схильні до помилок оцінки. Оцінка ефективності включала тривалу оцінку журналів аудиту, створених вручну за допомогою утиліт, що кодуються вручну. Помилки часто були фатальними, і персоналу доводилося виправляти ці помилки вручну, щоб відновити їхню роботу.

Таким чином, сучасна оркестрація даних розвинулася з ручних зусиль з оркестрування з акцентом на автоматизацію, концептуалізацію та аналітику для підтримки оптимізації. Насправді, термін "оркестрування даних" не був введений в інженерну мову до 2017 року.

У міру того, як оркестрування ставало все більш і більш поширеним, стало зрозуміло, що старі методи не усувають одне з основних обмежень на шляху до оптимального використання даних: "сховища даних". Сховища даних - це не буквально сховища, а концептуальний опис явища, коли дані потрапляють в пастку в одному місці, організації або додатку без простого або чіткого способу доступу до них і їх використання. Оркестрування, багато в чому, є практикою руйнування "силосів", роблячи ці дані доступними. Після цього завданням сучасної оркестрації стало сприяння руйнуванню ізоляції та забезпечення більшої доступності і корисності даних в організації.

Для цього сучасна оркестрація даних передбачає визначення основного завдання або завдань в системі даних і запуск так званого прямого ациклічного графа (DAG), який ілюструє всі відповідні завдання і їх взаємозв'язок один з одним. Автоматизація за допомогою коду може визначити структуру цих завдань з точки зору лінійних робочих процесів "Якщо-Тоді-Інакше", подій, що запускаються в часі, умовного виконання завдань або навіть шляхом вимірювання часу між одним завданням і іншим.

Багато років тому workflow був чимось на кшталт темного мистецтва, яким займалися лише багаті компанії, що могли дозволити собі дорогі системи управління документообігом та вузькоспеціалізованих консультантів.

Але сьогодні технологія робочих процесів є широко доступною, а її переваги та підводні камені - більш зрозумілими. Однак слід зробити застереження: Хоча документообіг не обов'язково повинен належати до таємничих знань, його також не можна тривіалізувати. Якщо програма документообігу не узгоджена з бізнесом, в якому вона впроваджена, це може бути гірше, ніж ручна, паперова. Тому важливо, щоб як бізнес, так і IT-зацікавлені сторони дотримувалися обґрунтованої методології BPM, перш ніж намагатися впровадити програму документообігу.

YAWL, що розшифровується як Yet Another Workflow Language, є повністю відкритою системою управління робочими потоками (або "системою управління бізнес-процесами). Її назва приховує той факт, що YAWL є досить унікальною. Вона базується на дуже багатій мові визначення робочих процесів, здатній врахувати всі види залежностей між завданнями. Вона має відкриті інтерфейси, засновані на веб-стандартах, що дозволяє розробникам підключати існуючі додатки, а також розширювати і налаштовувати систему різними способами. Вона також надає графічний редактор з вбудованою функцією верифікації, що допомагає архітекторам рішень і розробникам фіксувати моделі робочих процесів і автоматично виявляти тонкі, але потенційно неприємні помилки на ранній стадії роботи над проектом.

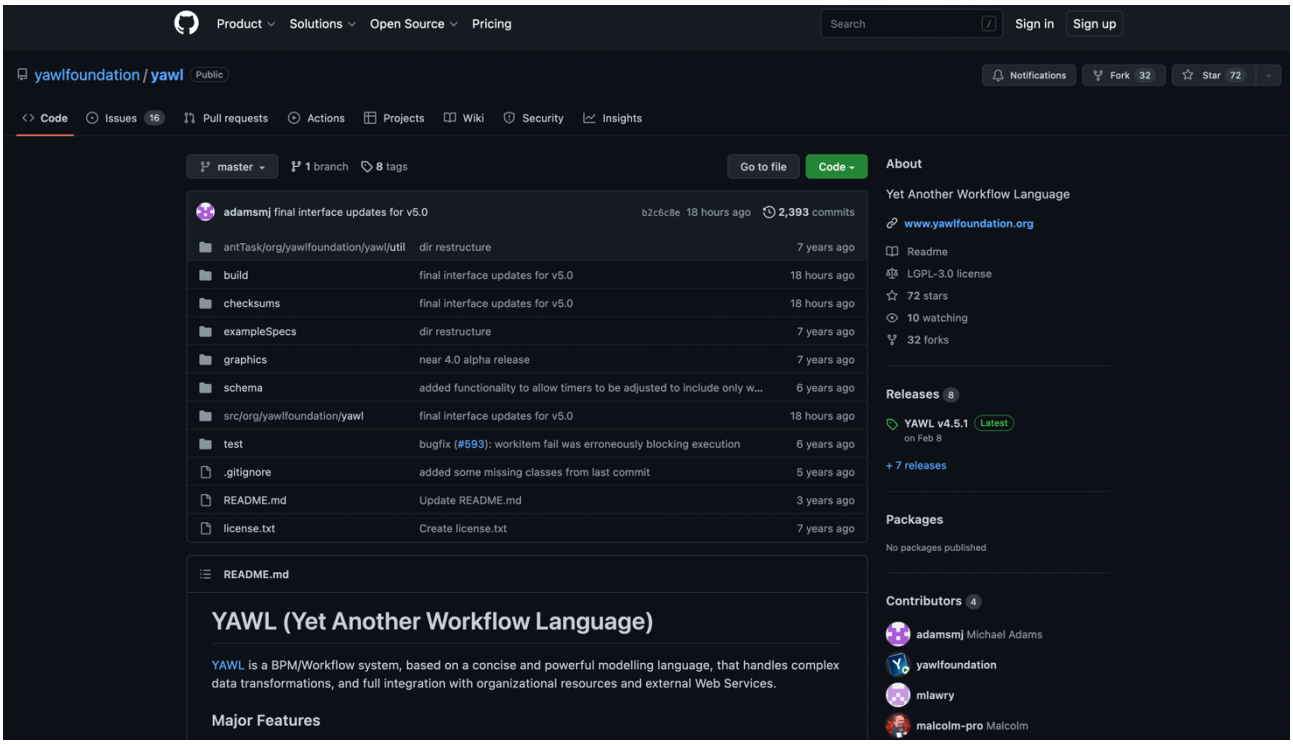


Рисунок 1.1.1 – Репозиторій YAWL

YAWL є, системою управління робочим процесом з відкритим вихідним кодом. Від свого початку як академічний продукт, YAWL перетворився на систему управління робочими процесами корпоративного рівня завдяки внескам організацій і приватних осіб, які її використовували. Ця цілеспрямована прихильність користувачів та спільноти розробників також забезпечує безперервність роботи системи.

CGAT-core - це система управління робочими потоками, яка дозволяє користувачам швидко і відтворювано будувати масштабовані конвеєри аналізу даних. CGAT-core - це набір бібліотек та допоміжних функцій, які використовуються для того, щоб дозволити дослідникам проектувати та будувати обчислювальні робочі процеси для аналізу великих масивів даних.

У поєднанні з CGAT-додатками має функціональність гнучкої системи управління робочими процесами на прикладі простого RNA-seq пайплайну.

Ядро CGAT є відкритим, потужним та зручним у використанні, і протягом останніх 10 років постійно розвивалося як система управління робочим процесом секвенування наступного покоління (NGS).

CGAT-core розширює популярний механізм робочого процесу Python Ruffus, додаючи бажані функції з безлічі інших систем робочого процесу, щоб сформувати надзвичайно простий, гнучкий і масштабований пакет. CGAT-core забезпечує безперебійну взаємодію високопродуктивного обчислювального кластера та вперше додає інтеграцію середовища Conda. Крім того, структура зосереджена на спрощенні розробки конвеєра та процесу тестування, надаючи зручні функції для параметризації, взаємодії з базою даних, журналювання та взаємодії конвеєра. Легкість конвеєрної розробки дозволяє CGAT-core подолати розрив між дослідницьким аналізом даних і створенням робочих процесів виробництва.

Головний принцип полягає в тому, що виконувати серію завдань за допомогою простого конвеєра має бути так само легко (або простіше), ніж за допомогою інтерактивної підказки, особливо якщо розглядається подання кластера. CGAT-core дає змогу створювати конвеєри аналізу, які можна легко запускати в кількох середовищах, щоб полегшити спільний доступ до коду в рамках процесу публікації. Таким чином, CGAT-core заохочує підхід до відтворюваних досліджень, роблячи його шляхом меншого опору[1].

Наприклад, пошуковий аналіз у Jupyter Notebooks можна перетворити на сценарій Python або використовувати безпосередньо в конвеєрі. Подібним чином пошуковий аналіз даних на R або будь-якій іншій мові можна легко перетворити на сценарій, який можна запускати конвеєром. Ця легка упаковка швидко створеного прототипу аналізу утворює лабораторну книгу, що дозволяє швидко відтворювати аналізи та повторно використовувати код для різних наборів даних.

Для забезпечення відтворюваності, ведення обліку та обробки помилок CGAT-core забезпечує багаторівневе протоколювання під час виконання

робочого процесу, записуючи повну інформацію про параметри виконання, конфігурацію середовища та відстежуючи відправлення завдань. Крім того, CGAT-core надає простий, легкий інтерфейс для взаємодії з реляційними базами даних, такими як SQLite (cgatcore.database), полегшуючи завантаження результатів аналізу на будь-якому кроці робочого процесу, включаючи об'єднання результатів паралельних кроків в єдині широкоформатні або довгоформатні таблиці.

CGAT-core може завантажувати окреме середовище Conda для кожного кроку аналізу, що дозволяє використовувати інструменти з суперечливими вимогами до програмного забезпечення. Крім того, надання файлів середовища Conda разом зі сценаріями конвеєра забезпечує повне відтворення результатів аналізу.

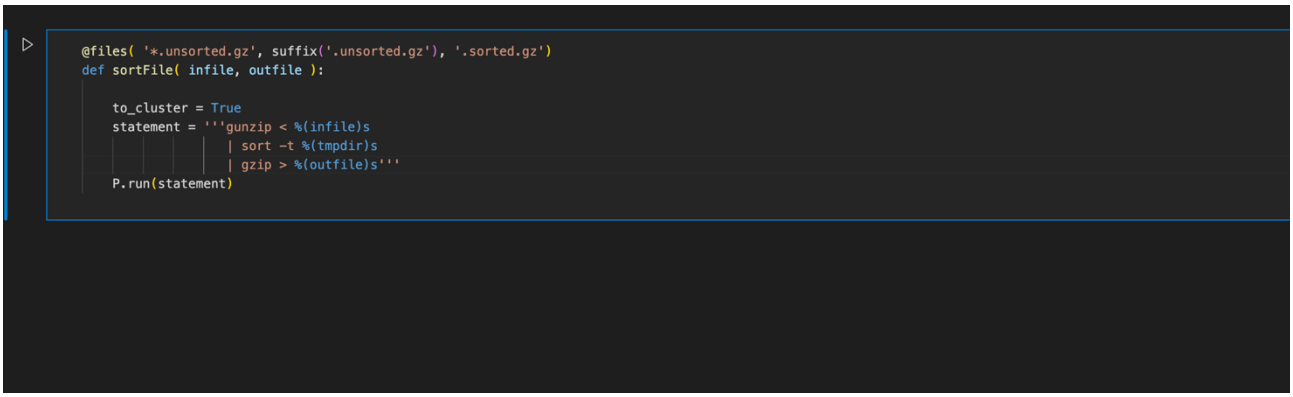


Рисунок 1.1.2 - Приклад швидкого запуску пайплайну на умовному кластері

Робочі процеси ядра CGAT є скриптами на мові Python, і як такі є автономними утилітами командного рядка, які не потребують встановлення спеціального сервісу. Для відтворюваного виконання робочих процесів надаються утиліти для розбору аргументів, логування та ведення записів всередині скриптів (cgatcore.experiment). Запуск, перевірка та налаштування сценаріїв здійснюється через інтерфейс командного рядка. Таким чином, робочі процеси стають ще одним інструментом і можуть бути повторно використані в

інших робочих процесах. Крім того, робочі процеси можуть використовувати всю потужність Python, що робить їх повністю розширюваними та гнучкими.

ZenML створений для вирішення дилеми власності в командах, що займаються аналізом даних. Він розроблений з використанням абстракцій більш високого рівня, які дозволяють аналітику даних повністю володіти конвеєром до моменту виробництва. Фахівцю з аналізу даних не потрібно розуміти деталі інфраструктури або інструментів розгортання, але він зможе використовувати їх за допомогою ZenML[2].

Звичайно, є й інші інструменти, які виглядають подібно до вищезгаданих, але ZenML орієнтований на робочі процеси ML. Конвеєри залежать від даних, а не від завдань. Це означає, що артефакти, які проходять через конвеєри, можуть бути змодельовані певним чином, щоб включити такі функції, як кешування і лінійний аналіз.

Артефакти, що проходять через етапи конвеєра, можуть бути стандартизовані (додавання стандартного етапу перевірки та розгортання для стандартних даних та артефактів моделі).

Кроки можуть бути стандартизовані для забезпечення однакового ефекту. Потім можна включити спеціальні функції для певних кроків (наприклад, розподілене навчання для кроку тренера).

ZenML може автоматично матеріалізувати (читати/записувати) загальні об'єкти, такі як фрейми даних Pandas і модулі PyTorch, незалежно від середовища, в якому працює цей конвеєр (локально або в хмарі). Після цього фахівець з даних може використовувати ці об'єкти, як він завжди це робить.

Навіть з усіма вищезазначеними особливостями, ці конвеєри та інтеграції повинні працювати в різних середовищах і з різними вимогами до інфраструктури для будь-якого варіанту використання. Саме тут з'являється поняття стеку MLOps[2].

Такий стек складається з

- оркестратора, для управління запуском робочих процесів (наприклад, Kubeflow)
- сховище метаданих, база даних для відстеження прогонів та артефактів у сховищі артефактів
- сховища артефактів, файлової системи для зберігання артефактів (AWS/GCP/Azure bucket)

В основі ZenML лежить конвеєрний робочий процес проектів з даними. Конвеєр складається з серії кроків, організованих у будь-якому порядку, який має сенс для вашого варіанту використання.

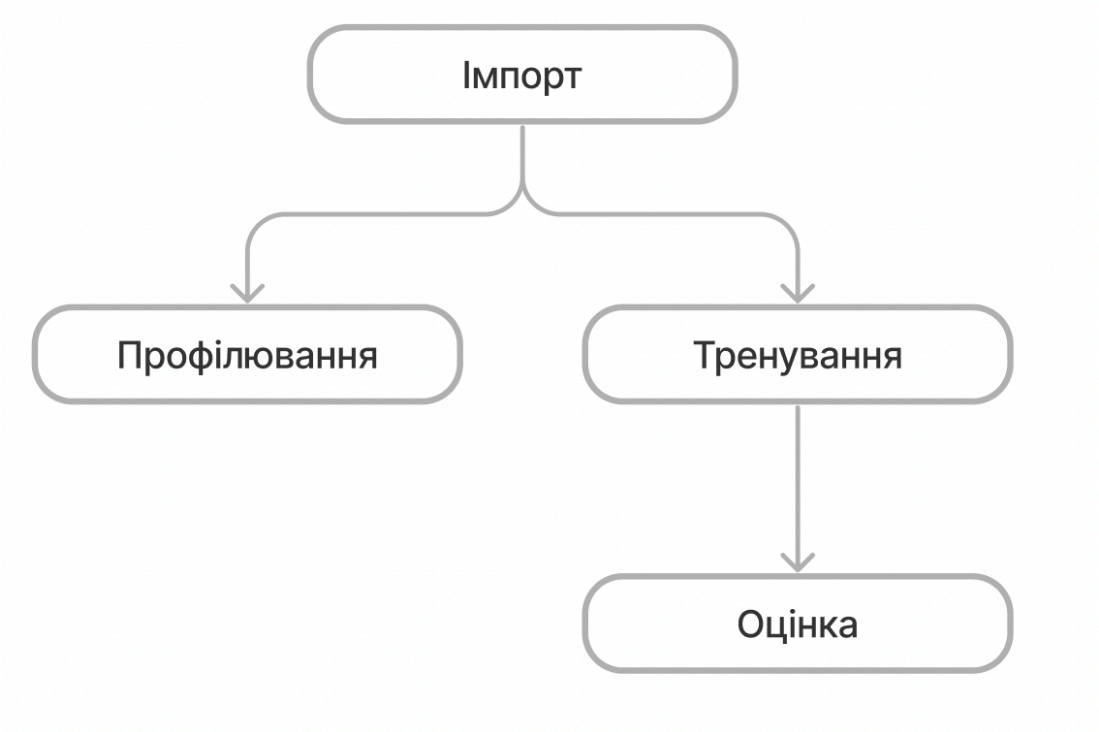


Рисунок 1.1.3 – Етапи роботи конвеєра ZenML

Кроки можуть мати залежності між собою. Наприклад, крок може використовувати результати попереднього кроку і, таким чином, повинен дочекатися завершення попереднього кроку, перш ніж почати його виконання. Це те, що можна мати на увазі при організації кроків.

Стек - це конфігурація базової інфраструктури та вибір способу запуску вашого конвеєра. Наприклад, таким чином можна вибрати запуск конвеєра локально або у хмарі, змінивши стек.

Оркестратор - це робочий механізм, який координує всі етапи роботи в конвеєрі. Оскільки конвеєри можуть складатися зі складних комбінацій кроків з різними асинхронними залежностями між ними, Оркестратор - це компонент, який вирішує, які кроки запускати, коли і як передавати дані між кроками.

ZenML постачається з локальним оркестратором за замовчуванням, призначеним для роботи локальній машині. Це корисно, особливо на етапі побудови проекту. В такому разі не потрібно орендувати хмарний екземпляр лише для того, щоб спробувати базові речі.

Сховище артефактів - це компонент, в якому зберігаються всі дані, що проходять через пайплайн. Дані в сховищі артефактів називаються артефактами. Ці артефакти можуть бути створені кроками конвеєра, або це можуть бути дані, що потрапляють в конвеєр через крок імпортера. У сховищі артефактів зберігаються всі результати проміжних кроків конвеєра.

Той факт, що всі вхідні та вихідні дані відстежуються та версіонуються в сховищі артефактів, дозволяє використовувати надзвичайно корисні функції, такі як кешування даних, що прискорює робочий процес.

Існує багато різних способів використання ZenML, які залежать від конкретного випадку використання. Наступні концепції та компоненти стеку - це те, з чим можливо стикнутися у подальшому під час використання ZenML

Матеріалізатори - ZenML зберігає вхідні та вихідні дані для ваших кроків у сховищі артефактів. Для того, щоб зберігати дані, він повинен серіалізувати все у форматі, який може помістится у Сховище Артефактів. ZenML обробляє серіалізацію (і десеріалізацію) найпоширеніших артефактів. ZenML CLI дасть знати за допомогою чіткого повідомлення про помилку, коли потрібно буде це зробити.

Сервіс - це довгоживучий об'єкт, який розширює можливості ZenML за межі роботи конвеєра. Наприклад, сервіс може бути службою прогнозування, яка завантажує моделі для висновків у виробничих умовах

Проект Kubeflow покликаний зробити розгортання робочих процесів машинного навчання (ML) на Kubernetes простим, портативним та масштабованим. Мета проекту - не відтворити інші сервіси, а надати простий спосіб розгортання найкращих у своєму класі систем з відкритим вихідним кодом для ML у різноманітних інфраструктурах. Скрізь, де використовується Kubernetes, користувач повинен мати можливість запустити Kubeflow. Kubeflow - це інструментарій машинного навчання для Kubernetes[3].

Для використання Kubeflow необхідно виконати наступну послідовність дій:

- Завантажити та запустити бінарний файл розгортання Kubeflow.
- Налаштувати отримані конфігураційні файли.
- Запустити вказаний скрипт для розгортання контейнерів у конкретному середовищі.

Інструментарій має можливість адаптувати конфігурацію для вибору платформ і сервісів, для кожного етапу робочого процесу ML:

- підготовка даних
- навчання моделі
- обслуговування прогнозування
- управління сервісами

Мета Kubeflow - зробити масштабування моделей машинного навчання (ML) та їх розгортання у виробництво якомога простішим: Просте, повторюване, портативне розгортання на різноманітній інфраструктурі (наприклад, експерименти на персональному комп’ютері, а потім переміщення на локальний кластер або в хмару)[3]

Оскільки фахівці використовують різноманітний набір інструментів, одна з ключових цілей полягає в тому, щоб налаштувати стек на основі вимог користувача (в межах розумного) .

Зрештою, розробники хочуть мати набір простих маніфестів, які дають простий у використанні стек ML в будь-якому місці, де вже працює Kubernetes, і який може самостійно конфігуруватися на основі кластера, в який він розгортається

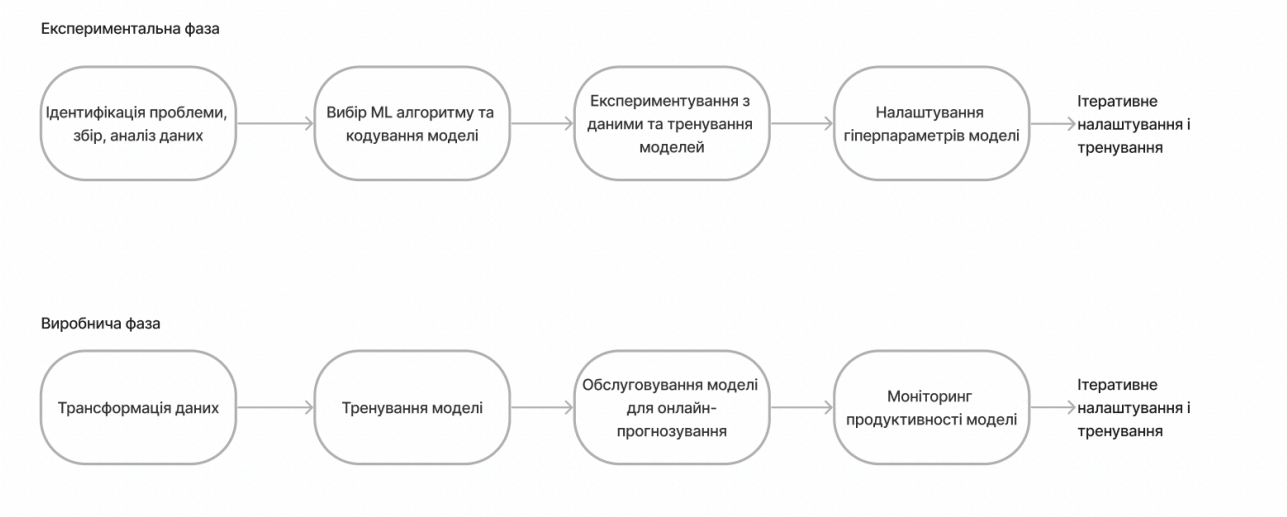


Рисунок 1.1.4 - Робочі процеси всередині Kubeflow

При розробці та розгортанні системи ML системи, робочий процес, як правило, складається з декількох етапів. Розробка системи ML системи є ітеративним процесом. Необхідно оцінювати результати різних етапів робочого процесу і вносити зміни в модель і параметри, коли це необхідно, для того, щоб модель продовжувала давати потрібні результати.

Детальний опис деталей:

На експериментальному етапі ведеться розробка моделі на основі початкових припущень, тестування та оновлювання модель ітеративно, щоб отримати потрібні результати, тому в процесі роботи присутні наступні кроки:

- Визначення проблеми, яку потрібно вирішити за допомогою системи машинного навчання.

- Збір та аналіз даних , необхідних для навчання моделі ML.
- Вибір фреймворку і алгоритму ML, а кодування початкової версії моделі.
- Експериментування з даними та навчанням моделі.
- Налаштування гіперпараметри моделі, щоб забезпечити найбільш ефективну обробку та максимально точні результати.

На виробничому етапі розгортується система, яка виконує наступні процеси:

- Перетворення даних у формат, який потрібен навчальній системі. Щоб гарантувати, що модель поводиться послідовно під час навчання і прогнозування, процес перетворення повинен бути однаковим на експериментальній і виробничій фазах.
- Навчання моделі ML.
- Запуск модель для онлайн-прогнозування або для запуску в пакетному режимі.
- Відстежування продуктивність моделі та використання результати у процесах для налаштування або перенавчання моделі.

1.2 Причина необхідності використання нових методів обробки даних та завдань

Оркестрування даних - це, по суті, ліквідація "силосів" даних, щоб дані не були фрагментованими і до них можна було швидко отримати доступ. Теоретично, якби компанія керувала своїми даними досить добре, вона не потребувала б оркестрування даних і могла б задовольнити всі свої потреби в даних самостійно. Але управління даними "досить добре" рідко буває практичним через швидкість, з якою змінюються технології, і багато компаній застосовують підхід до великих даних, який призводить до ізольованих даних та фрагментованих систем.

Оркестрування даних є найкращим варіантом для більшості компаній з декількома системами даних, оскільки воно не вимагає масової міграції або додаткових місць для зберігання даних, що іноді може призвести до того, що користувач отримає ще одне сховище даних.

Але це не єдина перевага оркестрації даних. Вона також допомагає дотримуватися законів про конфіденційність даних, усуваючи вузькі місця в роботі з даними і забезпечуючи управління даними.

Відповідність законам про конфіденційність даних - GDPR, CCPA та інші закони про конфіденційність даних вимагають від компаній довести, що їхні дані були зібрані етично. Це включає в себе деталізацію того, коли, де і чому були зібрані дані. Якщо користувач не організує свої дані, важко довести, що він дотримується цих законів.

Крім того, GDPR надає споживачам можливість відмовитися від збору даних або вимагати, щоб ваша компанія видалила всі дані, які раніше зібрали у них. Якщо користувач не має чіткого уявлення про те, де зберігаються ці дані та хто має до них доступ, може бути важко дозволити споживачам відмовитися від ваших методів збору даних. Але гарантувати, що були видалені всі їхні дані на вимогу ще складніше, коли ви не знаєте, де вони зберігаються.

Запити на відмову та видалення даних трапляються часто. Вкрай важливо чітко розуміти , де знаходяться самі дані, і оркестрування може допомогти досягти цього.

Усунення вузьких місць в даних - у традиційній, неоркестрованій екосистемі даних, якщо користувач хоче проаналізувати свої дані, процес може загрузнути. Може бути кілька сховищ даних та інших систем зберігання, до яких потрібно зробити запити. Якщо не володіти знаннями про ці системи, доведеться звернутися до того, хто має можливість виконувати запити до ваших даних.

Швидше за все, у цієї людини є багато інших співробітників, які також запитують дані. Таким чином, запити додаються до списку справ. Коли ця людина нарешті добереться до вашого, вона витягне потрібні дані і надішле їх

користувачу. Але тепер потрібно буде вручну перетворити дані так, щоб їх використати. Після цього все одно потрібно буде завантажити їх в інструмент бізнес-аналітики або інший інструмент аналізу. Коли нарешті завершиться весь цей процес обробки даних, тільки тоді є можливість приступити до аналізу.

В керованому середовищі майже всі ці кроки усуваються. Всі дані будуть активовані в кінцевій точці, тому буде можливість відразу перейти до інструментів аналізу і приступити до роботи. Дані також будуть стандартизовані, тож не потрібно буде перетворювати їх вручну.

Велика кількість роботи, пов'язаної з аналізом даних, - це просто отримання та підготовка даних. Саме тут виникає багато вузьких місць. Отже, ключова перевага оркестрування даних полягає в тому, що воно може значно скоротити кількість часу, витраченого на ці два етапи, оскільки воно здатне автоматично впоратися з важкою роботою зі збору та підготовки даних.

Впровадження системи управління даними - є складним завданням, коли власне конвеєр даних розподілений між декількома системами їх зберігання. Оскільки інструмент оркестрування даних з'єднує всі системи даних, йому легше впроваджувати стратегію управління даними.

Слід пам’ятати, що оркестрування може організувати дані в режимі реального часу. Якщо було створено план відстеження або структуру стратегії управління даними, інструмент оркестрування даних може забезпечити відповідність зібраних даних цьому плану.

1.3 Принципи роботи оркестрації. Порівняння оркестрації та хореографії

Мікросервіси можна легко визначити як набір послуг, які можуть працювати як окремий додаток. Існує безліч переваг, які може надати

мікросервіс. У випадку з мікросервісом, ми маємо звернути увагу на сервіс, який поводиться абсурдно, замість того, щоб перевіряти весь додаток.

Дотримуючись мікросервісної архітектури, кожен сервіс може бути оптимально масштабований в залежності від навантаження. В тому випадку, коли не потрібно масштабувати весь додаток, як фронтенд, проміжне програмне забезпечення, бекенд тощо. Це економить величезну кількість витрат на інфраструктуру.

Мікросервіси є відмовостійкими, якщо вони реалізовані належним чином. Кожним сервісом може керувати окрема команда в залежності від складності проекту. Масштабованість не є проблемою для мікросервісів. Розгортання повністю відокремлене від програми, що робить його кращим вибором. У гіршому випадку, тільки цей конкретний додаток може піти не так, а не весь додаток.[5]

Після того, як вже існує кілька мікросервісів, що працюють під потреби проекту, потрібно, щоб вони обмінювалися даними і будували бізнес-логіку. Оркестрування - це один із способів зробити це. Оркестрування відштовхується від концепції оркестру, де є декілька музикантів, які є майстрами гри на своєму інструменті, що є сервісом в архітектурі мікросервісів. Диригент оркестратор дає напрямок всім, про вузлам.

Гравці (Сервіси) самі по собі достатньо здатні зіграти всю пісню, оскільки мають під рукою ноти (бізнес-логіку) пісень. Але для того, щоб зіграти музику (Проект) разом, в симфонії, їм потрібен диригент, який буде всім керувати.

Подібним чином працює і оркестрування мікросервісів, коли батьківський сервіс керує запитом. Він перенаправляє запит до відповідного сервісу і збирає дані. Остаточна відповідь формується в головному сервісі, який називається оркестратором, а потім передається клієнтському додатку. Роль оркестратора полягає у спілкуванні з кожним сервісом, який необхідний для виконання будь-якого запиту API.

Існує декілька патернів для проектування архітектури мікросервісів, одним з таких патернів є патерн "Сага", який може бути використаний для проектування архітектури мікросервісів або за принципом хореографії, або за принципом оркестрування.

У хореографії мікросервіси працюють паралельно, на відміну від оркестрування. Вся система працює на основі подієвої архітектури, де сервіс збирає дані з шини повідомлень і виконує бізнес-логіку, а натомість передає дані в іншу шину повідомлень. Такий підхід при правильній реалізації не забезпечує відмовостійкості, оскільки кожен сервіс працює сам по собі. Якщо у випадку виходу з ладу одного з них, одразу можна дізнатися , який саме сервіс викликає проблему , і можна безпосередньо працювати над цим.

Cloud Native додатки майже завжди будуються як набір мікросервісів, які працюють в контейнерах (наприклад, Docker) і можуть керуватися і розгортатися за допомогою робочих процесів DevOps і CI/CD (Continuous Integration/Continuous Deployment - безперервна інтеграція/безперервне розгортання).

Існує кілька варіантів побудови такої реалізації. Одним з варіантів є "безсерверна" модель, запропонована, наприклад, AWS Lambda, яка виконує блоки коду у відповідь на певні події, і автоматично управляє базовими ресурсами обробки.

Повертаючись до мікросервісної архітектури, її перевага полягає в поділі додатку на низку модульних та багаторазово використовуваних сервісів. Невеликі, легкі та прості в реалізації сервіси призводять до зниження витрат на розробку та модифікацію, а також забезпечують швидке та легке масштабування.

Для того, щоб ця система працювала, необхідний метод координації, який дозволить мікросервісам працювати разом. Існує в основному два підходи: Orchestration та Choreography - англійські терміни, які не тільки ідеально відповідають італійським еквівалентам "orchestrazione" та "coreografia", але й відображають їх значення з точки зору виконавського мистецтва.

Звичайно, як і всі аналогії, ця не є ідеальною, але вона відображає принципову різницю між двома підходами. В оркестрі є диригент, який щохвилини вказує музикантам, що вони мають робити. Хореографія, з іншого боку, заздалегідь задумана і викладена хореографом, і під час виконання танцюристи рухаються вільно.

В основі оркестрування лежить ідея центральної системи, яка контролює всі взаємодії між різними елементами системи (тобто мікросервісами). У цьому підході ключовим словом є "нагляд" або контроль за дотриманням окремих інструкцій. В оркестрації, по суті, є служба, яка виступає в ролі контролера і обробляє комунікації між окремими мікросервісами. Таким чином забезпечується виконання кожним сервісом своєї частини роботи.

Контролер слугує проміжним програмним забезпеченням, тобто програмним забезпеченням, яке надає додаткам спільні функції та сервіси. Взагалі кажучи, можна мати на увазі проміжне програмне забезпечення як сполучна тканина між різними рівнями програми. У конкретному випадку Orchestration - компонент проміжного програмного забезпечення виступає в ролі супервізора, який контролює взаємодію між різними мікросервісами.

Оркестрування може використовуватися для широкого спектру завдань. Наприклад, Kubernetes дозволяє управляти оркестрованими ресурсами декларативно, абстрагуючись від базового фізичного рівня, тому ними можна керувати як єдиним пулом обчислювальних ресурсів.

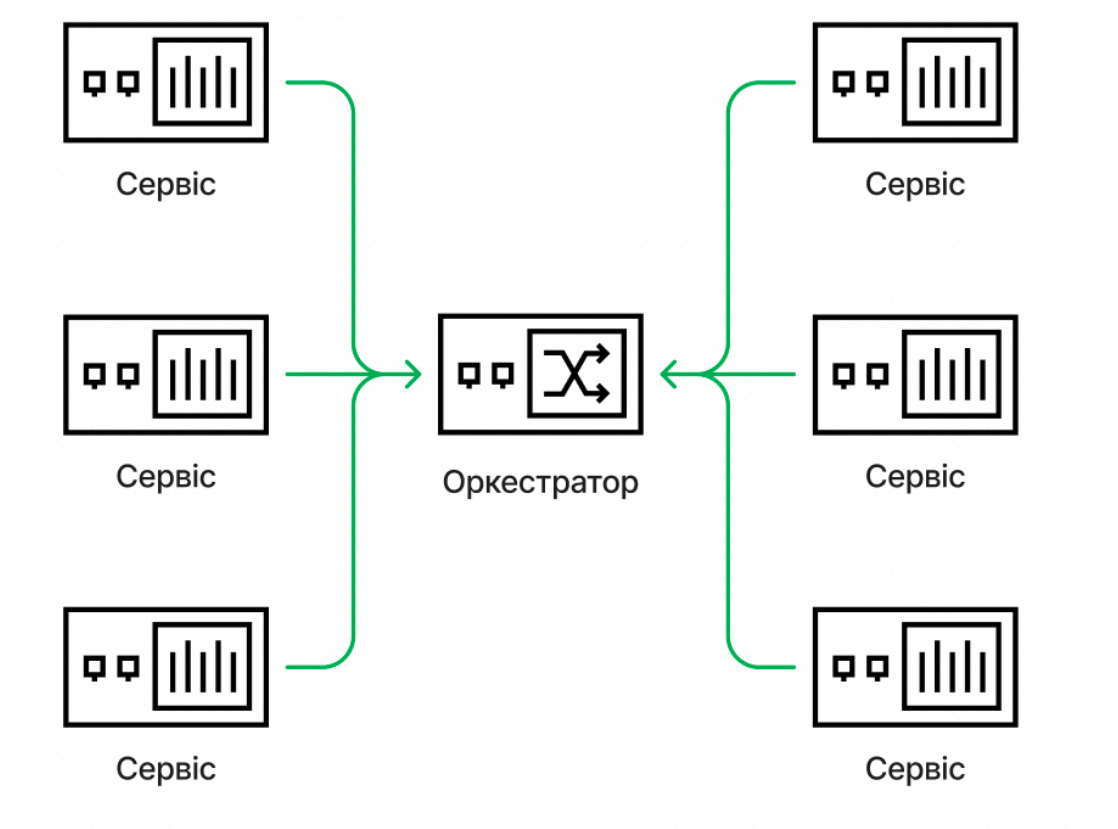


Рисунок 1.3.1 – Схема роботи оркестраційного середовища

Управління декларативною конфігурацією здійснюється за допомогою контуру управління. Подібно до того, як це робить термостат в кімнаті, коли визначається бажаний стан, термостат перевіряє поточний стан і максимально наближає його до бажаного. У випадку з Kubernetes, контролери спостерігають за кластером і при необхідності вносять або запитують зміни в поточний стан.

Таким чином, в основі "оркестрованого" підходу лежить ідея створення централізованої і добре організованої системи управління бізнес-процесами (або BPM), яка може забезпечити стабільність роботи додатка і зробити його стан легко вимірюваним і модифікованим.

Хореографія використовує зовсім інший і принципово децентралізований підхід. Мається на увазі, що постановка відбувається тоді, коли учасники самостійно виконують свої ролі. Жоден елемент не виконує чіткого наказу, скоріше, кожен знає набір дій, які він повинен виконати по відношенню до інших елементів, з якими він взаємодіє.

Хореографічний підхід базується на ідеї, що центральний елемент управління, оркестратор (тобто проміжне програмне забезпечення, таке як Kubernetes) є надлишковим. Окремі компоненти, тобто мікросервіси, повинні бути здатні до самоуправління без зовнішнього втручання і пов'язані між собою вільно (loose coupling), а не жорстко (tight coupling), іншими словами, в загальному і не надто жорсткому порядку.

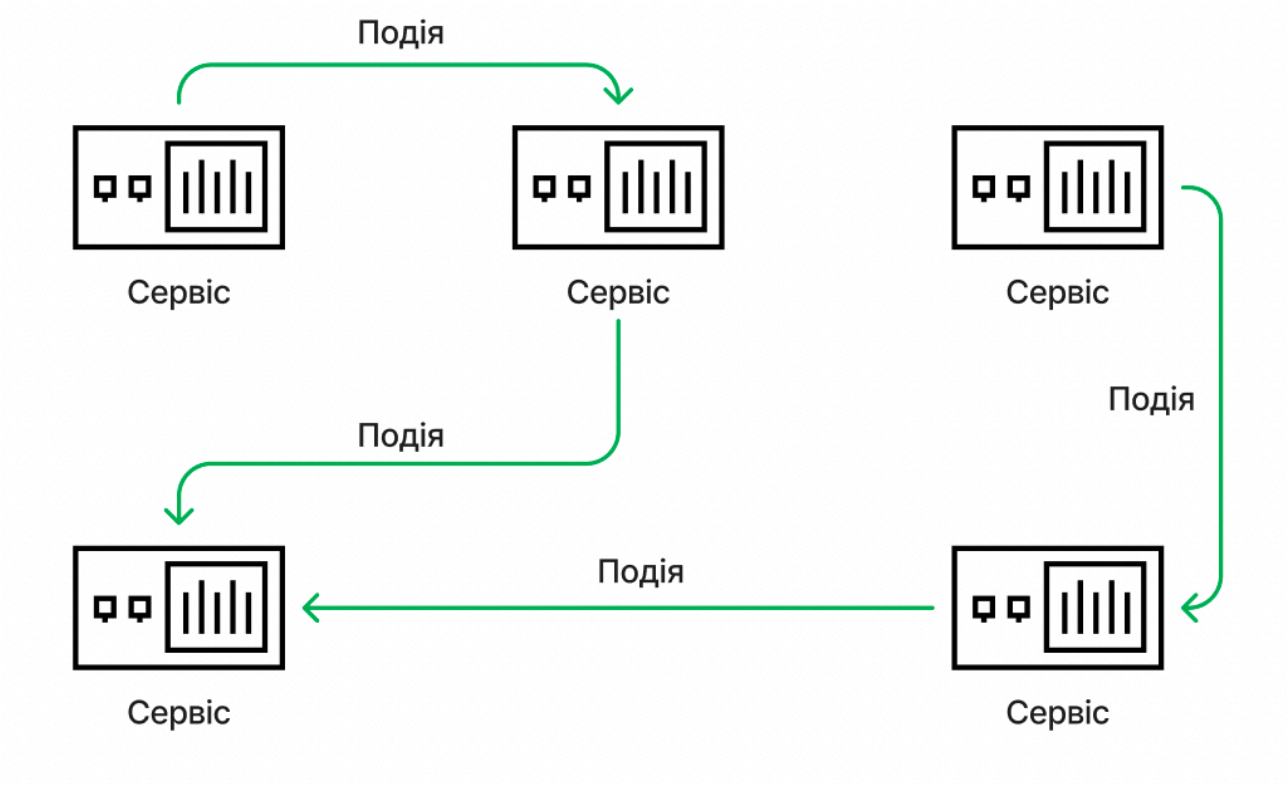


Рисунок 1.3.2 – Схема роботи хореографічного середовища

Це означає, що вільно пов'язані мікропослуги повинні бути здатні забезпечити максимально можливу цінність для бізнесу самостійно, без прямого впливу на інші компоненти. Таким чином, її складність зменшується, оскільки немає контролера для програмування і управління. Крім того, як можна здогадатися, відсутній єдиний вузол, тобто потенційно критичний елемент для всієї інфраструктури у разі виникнення проблем.

У хореографічному підході використовується компонент, який називається "брокер подій" (або брокер повідомлень). Хореографія відбувається відповідно

до послідовності кроків. Кожен мікросервіс, який виконав дію, надсилає повідомлення по певних каналах. Інші мікросервіси, підписані на цей конкретний канал, отримують повідомлення. У цей момент вони самі знають, що їм робити, оскільки вони спроектовані так, щоб автоматично реагувати на певні події.

Використання вільної логіки зв'язку сервісів дозволяє змінювати мікросервіси (додавати або видаляти за необхідності), не роблячи при цьому логіку, що лежить в основі, непрацездатною і такою, що потребує переписування.

Основна відмінність між мікросервісною хореографією та оркестровкою полягає в тому, що у випадку з хореографією немає нічого центрального. Прийняття будь-якого з них залежить від варіанту використання. Якщо потрібна дуже часта комунікація між службами, тоді потрібно вибрати оркестровку, в іншому випадку хореографія є кращим варіантом.

У випадку з оркестровкою потрібно мати належне протоколювання, інакше буде дуже складно знайти, який сервіс не зміг повернути правильні дані. Логування помилки - це завжди рятівник.

Здебільшого архітектор використовує об'єднання обох підходів для створення остаточного проекту архітектури, що робить всю систему відмовостійкою. Різні операції є синхронними за своєю природою, в той час як інші - асинхронними, тому обидва ці підходи необхідні для того, щоб зробити продукт відмовостійким.

1.3.2 Використання двигунів робочих процесів для оркестрування мікросервісів

Сервіс-орієнтовані додатки все частіше стають хмарними і будуються як набір невеликих, незалежних і слабко пов'язаних між собою мікросервісів.

Великі веб-компанії, такі як Tencent, Uber, Netix та Airbnb, все частіше будують свої основні бізнес-системи з використанням мікросервісної

архітектури. Сама перспектива мікросервісів полягає в тому, що кожен мікросервіс може бути незалежно розроблений, розгорнутий, протестований, модернізований та масштабований. Це робить їх придатними для систем, що працюють на хмарних інфраструктурах. Однак, такі переваги мають певну ціну, і системи на основі мікросервісів ускладнюються через складну взаємодію різних сервісів, події, що відбуваються одночасно, збої в роботі компонентів, відсутність у розробників глобального бачення та конфігурацію середовища. Ця складність і динамічність мікросервісних систем створює унікальні виклики для розробників систем і ускладнює їх впровадження та налагодження. Хореографія традиційно є більш поширеним способом організації мікросервісів. Вона слідує парадигмі, керованій подіями, в якій кожен сервіс працює незалежно. Між мікросервісами немає жорстких залежностей, і вони вільно пов'язані лише через спільні події. Кожен сервіс прослуховує події, які його цікавлять, і несе повну відповідальність за реагування на ці події та виконання своїх операцій або бізнес-логіки незалежно. Це включає в себе роботу з усіма типами збоїв[6].

Такі фреймворки, як Spring Boot, дозволяють легко створювати автономні мікросервіси, вбудовуючи можливості сервісів, клієнтські можливості REST, інтеграцію з базами даних та зовнішню конфігурацію в автономний Spring-додаток. Контейнери, такі як Docker , дозволяють легко збирати та розгортати ці додатки. Організація та управління такими сервісами включає в себе виявлення та управління навантаженням, що забезпечується такими фреймворками, як Kubernetes.

Мікросервіси об'єднуються разом для реалізації робочих процесів, які є повторюваними шаблонами діяльності, що природно виникають в результаті систематичної організації ресурсів та інформації. Це відбувається органічно в хореографії, і жоден суб'єкт не відповідає за наскрізний моніторинг системи або бізнес-процесів. Це полегшує реалізацію бізнес-логіки, але значно ускладнює логіку відновлення після збоїв. Для забезпечення доступності в разі збоїв інфраструктури в хмарі мікросервіс повинен захищати себе від збоїв своїх залежностей. На відміну від цього, механізм робочих процесів полегшує явну

організацію мікросервісів за рахунок управління низькорівневими проблемами розподіленого програмування і дозволяє мікросервісу зосередитися на реалізації бізнес-логіки. Історично, розподілені робочі механізми були реалізовані для виконання складних, обчислювально інтенсивних і повторюваних завдань, таких як секвенування генів в біоінформатиці.

Однак, такі ранішні робочі механізми були би дуже специфічними для домену, нестійкими і непрограмованими. Прикладом програмованого розподіленого фреймворку є MapReduce , оскільки він надає модель програмування, яка дозволяє аналітикам даних писати розподілені завдання без необхідності мати справу з деталями низькорівневого розподіленого програмування. Однак MapReduce не є рушієм в тому сенсі, що він не контролює і не керує життєвим циклом робочого і не є відмовостійким. Temporal - це рушій робочих процесів для мікросервісів, який управляє життєвим циклом процесу, а також забезпечує модель програмування, що не пам'ятає про помилки, для організації мікросервісів.

Використовуючи механізм Temporal, програмісти пишуть свій код звичною мовою, в той час як платформа організовує виконання завдань і робить додаток стійким до всіх збоїв. Це означає, що будь-який стан в робочому середовищі є довговічним і дозволяє розробнику зосередитися на бізнес-логіці додатку, а не витрачати більшу частину часу на створення відмовостійкості. Код, написаний для Temporal, повинен дотримуватися певних обмежень, наприклад, чисто випадкова змінна в частині коду, яка повинна виконуватися повторно, може порушити припущення робочої області станів: виконання повинно бути імпульсивним і детермінованим. Це семантичні помилки, які порушують частину коду, яка в іншому випадку буде вважатися правильною[6].

Мікросервіси є однією з найбільш швидкозростаючих областей в інженерії програмного забезпечення , однак, існує обмежена кількість високоякісних досліджень, що проводяться в цій галузі. Hassan та ін. розглянули методи проектування мікросервісів за такими параметрами, як розмір/кількість та глобальне/локальне задоволення нефункціональних вимог. Більш пізня робота

зосереджена на якісному аналізі композиції мікросервісів за допомогою хореографії або оркестрування. Чжоу та ін. надають гарний літературний огляд загальнодоступних еталонів мікросервісів, на основі якого автори зробили свій вибір[6].

Налагодженню мікросервісів останнім часом приділяється багато уваги. Це стало основою роботи авторів та порівняння. Георгіаді та ін. надали основу для системного систематичного тестування можливостей обробки відмов мікросервісів. Графи викликів сервісів, журнали використання сервісів та графи атрибутів останнім часом використовуються для аналізу першопричин виявлення аномалій та налагодження проблем доступності. Temporal надає детальні журнали та історії викликів, які можуть бути корисними для такого аналізу.

Хореографія: Вільне з'єднання сервісів і сильна згуртованість в хореографічній архітектурі мікросервісів здаються дуже перспективними з точки зору гнучкості та відмовостійкості. Це робить додавання і видалення послуг таким же простим, як підключення або відключення послуги від відповідного каналу в брокері подій. Вільне з'єднання також означає, що хореографія ізолює мікросервіси, так що якщо один сервіс виходить з ладу, інші сервіси, що не залежать від нього, можуть продовжувати роботу, поки проблема вирішується.

Хореографічні, керовані подіями мікросервіси дозволяють командам розробників працювати більш незалежно і зосередитися на своїх ключових послугах. Після того, як ці сервіси створені, їх легко можна використовувати спільно з іншими командами. Однак команди, які створили більші системи, дізнаються, що в процесі роботи виникає багато технічної заборгованості .

Процеси "вбудовуються" в код декількох сервісів. Така реплікація коду є головним викликом для технічного обслуговування. Крім того, існують тісні зв'язки та припущення щодо угод про введення/виведення та інших угод про рівень обслуговування, які ускладнюють адаптацію до мінливих потреб. Після цього з'явилися стандартизовані бібліотеки та ієрархічні системи, поки Кодд не

представив реляційну модель даних , а Чемберлен і Бойс - структуровану англійську мову запитів . Авторами передбачено, що механізми Workow зроблять для мікросервісів те, що реляційні системи баз даних зробили для інформаційних систем. Автори стверджують, що механізми Workow можуть звільнити розробників від зосередження на проблемах низькорівневого розподіленого програмування (таких як реалізація обмежень ACID) і дозволити їм зосередитися на реалізації бізнес-логіки. Відповідно до теореми CAP , кожен екземпляр сервера в кінцевому підсумку є доступним і високо узгодженим.[6]

Таким чином, Temporal Server забезпечує довговічну віртуальну пам'ять на кожну робочу операцію, яка не пов'язана з будь-яким конкретним процесом. Він зберігає повний стан програми (включаючи програмні стеки з локальними змінними) при всіх видах збоїв, пов'язаних з програмним і апаратним забезпеченням. Temporal SDK спирається на ці можливості і дозволяє користувачам писати код своїх додатків, використовуючи всю потужність мови програмування. Temporal гарантує, що викликаний виклик ніколи не призведе до збою, що робить корисним довготривалий код, який виконується протягом декількох днів або навіть місяців (скажімо, метод, який повинен виконувати щось через кожні 7 днів).

1.4. Огляд використання механізмів на основі «довіри» в хмарних обчисленнях

Хмарні обчислення, відомі як "хмара", - це сучасна технологія обробки даних, яка дозволяє комп'ютерам обробляти дані через Інтернет. Це форма обчислень, яка забезпечує масштабованість і еластичність. Клієнти отримують ці функціональні можливості через різні інтернет-сервіси. Хмарні обчислення, за визначенням Національного інституту науки і технологій, - це парадигма надання мережевого доступу на вимогу до спільного пулу програмованих

ресурсів, які можуть бути швидко доставлені і видалені з мінімальними зусиллями з адміністрування або залученням постачальників послуг [7].

Хмарні обчислення створюють гнучке онлайн-середовище, яке дозволяє збільшувати обсяги робіт, не впливаючи на реалізацію фреймворку . Хмарні обчислення об'єднують багато сучасних технологій в парадигмі інфраструктури на основі веб-сервісів для забезпечення гнучкості бізнесу, підвищеної масштабованості, спрощеного управління та доступності ресурсів за вимогою. Кінцевим споживачам не потрібно знати про внутрішні технології.

Хмарні технології часто мають стратегію швидкого розгортання, мінімальні початкові інвестиції, систему оплати, споживання та спільного використання спільних ресурсів, що є елементами, які великі підприємства використовують для перетворення своїх бізнес-додатків у віртуальні додатки . Компанії можуть сплачувати плату за використання для вибору постачальників хмарних послуг, щоб отримати функціональність системи без необхідності купувати ліцензії на апаратне чи програмне забезпечення або платити за технічне обслуговування що є набагато дорожчим ніж взяти в оренду віртуальні обчислювальні ресурси. Як наслідок, хмарна модель є значно більш вигідним способом отримання та споживання IT-послуг .

Хмара - це величезна колекція ресурсів і послуг, до яких можна отримати доступ за певну плату в будь-який час, коли вони можуть знадобитися:

Загальний доступ до мережі: хмарні сервіси доступні з будь-якого місця за допомогою звичайних терміналів, таких як мобільні телефони, ноутбуки та персональні цифрові помічники.

- Простота у використанні, де хмарний провайдер пропонує веб-інтерфейси, які простіші у використанні, ніж інтерфейси прикладного програмного забезпечення, що дозволяє клієнтам швидко отримати доступ до хмарних сервісів.
- Бізнес-модель, де послуги або ресурси оплачуються по мірі їх використання.

- Об'єднання ресурсів незалежно від місця розташування є об'єднання ресурсів незалежно від місця розташування використовується для обслуговування численних клієнтів з різними фізичними та віртуальними потребами з використанням багатокористувацької архітектури.

Функціональні аспекти хмарних обчислень: в ідеалі користувачі мають отримувати свою обчислювальну платформу або IT-інфраструктуру з хмари, а потім запускати свої додатки в ній. В результаті, хмарні обчислення надають споживачам прозорий доступ до апаратних, програмних та інформаційних ресурсів. Хмара виконує три основні функції[7].

Програмне забезпечення як послуга (Saas): кінцеві користувачі отримують програмне забезпечення на основі запитуваних ними послуг, що часто здійснюється через браузер. Це позбавляє клієнтів необхідності мати справу з такими питаннями, як розгортання та обслуговування програмного забезпечення. Програма часто використовується спільно численними орендарями, отримує автоматичні хмарні оновлення і не потребує окремої ліцензії. Базова конфігурація знаходиться поза контролем користувача. Функції можуть бути замовлені на вимогу та регулярно розсилатися. SaaS часто легко поєднується з іншими додатками завдяки своїм сервісним характеристикам. Карти Google є прикладом продукту SaaS.

Платформа як послуга (Paas) - послуга, яку також називають хмарним програмним забезпеченням, є платформою для розробки, яка включає в себе набір сервісів для проектування, розробки, тестування, розгортання, моніторингу та хостингу хмарних додатків. Зазвичай це не вимагає завантаження або встановлення програмного забезпечення і дозволяє географічно віддаленим командам співпрацювати над проектами.

Цей рівень включає Google App Engine, Microsoft Azure та Amazon Map Reduce / Simple Storage Service . Користувачі цих сервісів не мають контролю або доступу до основної фізичної інфраструктури, на яку покладаються їх додатки.

Інфраструктура як послуга (IaaS): рівень, що віртуалізує обчислювальні потужності, сховища та підключення в центрах обробки даних і надає їх споживачам як послуги. Ці обчислювальні ресурси можуть динамічно збільшуватися і зменшуватися на вимогу користувачів. Кілька орендаторів зазвичай спільно використовують одні й ті ж інфраструктурні ресурси. Фундаментальні ресурси знаходяться під повним контролем постачальника хмарних послуг

Архітектура хмарних обчислень: систему хмарних обчислень можна розділити на дві частини: front end і back end . Вони обидві з'єднані мережею, якою, як правило, є Інтернет. Споживач (користувач) бачить фронт-енд, а бек-енд - це може бути хмара системи. Клієнтський комп'ютер і додатки, необхідні для доступу до хмари, знаходяться на передньому кінці, тоді як задній кінець містить послуги хмарних обчислень, такі як комп'ютери, сервери та сховища даних. Центральний сервер має змогу відповідати за моніторинг трафіку, системне адміністрування та запити клієнтів. Він дотримується певних інструкцій або протоколів і використовує проміжне програмне забезпечення. Проміжне програмне забезпечення дозволяє комп'ютерам, об'єднаним у мережу, взаємодіяти . Рівень інфраструктури, рівень платформи, центр обробки даних та рівень додатків є чотирма підрозділами архітектурного дизайну хмарних обчислень.

- Рівень центру обробки даних: центр обробки даних, часто відомий як апаратний рівень, має відповідати за обробку та адміністрування фізичних ресурсів хмари. Маршрутизатор, системи охолодження живлення і фізичні сервери є частиною цього рівня. Сотні і тисячі серверів об'єднані в мережу через маршрутизатори і комутатори в центрі обробки даних.
- Інфраструктурний рівень: це другий рівень, фізичні ресурси центру обробки даних розділені за допомогою віртуалізації для створення пулу обчислювальних ресурсів і ресурсів зберігання даних. Цей рівень є критично важливим, оскільки технологія віртуалізації дозволяє забезпечити багато хмарних функцій.

- Рівень платформи: поверх інфраструктурного рівня встановлюється цей рівень. Він включає в себе як операційну систему, так і фреймворк додатків. Платформний рівень використовується для зменшення робочого навантаження при розгортанні програми безпосередньо в контейнер віртуальної машини.
- Рівень додатків: цей рівень є найвищим рівнем ієрархії, та містить хмарний додаток, а також є найбільш видимим для користувача. Для доступу до послуг, що надаються на цьому рівні, використовується онлайн-портал або спеціальні додатки.
- Хмарний клієнт: включає в себе комп'ютери та/або комп'ютерні програми, що використовують хмарні обчислення для надання додатків або спеціально створені для надання хмарних послуг.
- Безпека є основним фактором при зберіганні конфіденційної інформації в хмарі. Дані користувача є конфіденційною інформацією, і для них має бути забезпечена висока безпека.

Наступними наведені основні загрози безпеки при зберіганні та обміні даними в хмарі

- За контролем доступу може бути несанкціоноване використання конфіденційної інформації (даних клієнта) в хмарі можна запобігти за допомогою політик контролю доступу. Багато організацій можуть дозволяти доступ до ресурсів тільки тим користувачам, які попередньо зареєструвалися за допомогою своїх дійсних облікових даних. Політики контролю доступу різняться в залежності від їх ролі. Політики контролю доступу, передані на аутсорсинг в хмару, не повинні витікати назовні.
- Конфіденційність - це процес збереження конфіденційності конфіденційної інформації користувача, що зберігається в хмарі, якщо користувач не надає дозвіл на її розміщення. Вона повинна підтримуватися уповноваженими особами та постачальником хмарних послуг. Користувачі зберігають дані в зашифрованому вигляді; це можливо перетворити звичайний текст в ASCII-код для забезпечення

конфіденційності. При відправці даних через Інтернет повинна зберігатися конфіденційність. При зберіганні та отриманні даних повинні бути вирішені питання управління ключами. Авторизовані користувачі можуть отримати доступ до своїх конфіденційних даних у будь-який час і можуть виконувати будь-які операції, такі як читання, запис, оновлення тощо, а також визначати, як їхні конфіденційні дані передаються іншим особам. Це передбачає збереження конфіденційності.

- Аутентифікація: конфіденційні дані мають бути доступні лише авторизованим користувачам. Облікові дані, надані користувачами, повинні відповідати обліковим даним, збереженим користувачами в процесі автентифікації. Якщо облікові дані розголошуються, неавторизовані користувачі можуть отримати доступ до конфіденційних даних авторизованих користувачів.
- Доступність даних: конфіденційні дані користувачів хмари, що зберігаються в хмарі, можуть бути доступні лише авторизованим користувачам у будь-який час. Можна припустити , якщо ресурси недоступні для клієнтів з іншого місця, вони не зможуть переглянути конфіденційну інформацію. Недоступність може виникнути через поганий інтернет-зв'язок, а також через доступ до інформації несанкціонованих осіб

Право власності на дані: Більшість несанкціонованих користувачів мають доступ до конфіденційної інформації в банківських даних через відсутність ідентифікації власника зашифрованих банківських даних. Використання алгоритмів шифрування, шляхом видачі відкритих та закритих ключів, ускладнює зловмисникам доступ до них.

Наступними у дослідженні представлено застосування хмарних обчислень Деякі з багатьох додатків для управління бізнесом, які базуються на хмарних обчисленнях, - це управління взаємовідносинами з клієнтами (CRM) і планування ресурсів підприємства (ERP). Метод розгортання цих послуг - програмне забезпечення як послуга. Хмарні CRM-додатки створюють

можливості для стартапів і невеликих організацій отримати повнофункціональне програмне забезпечення CRM без величезних витрат і абонентської плати. Salesforce є добре відомою і добре розвиненою CRM-платформою. Ця CRM надає конфігуровані рішення, які можуть бути об'єднані з функціями від розробленої третьої сторони[7] .

Аналіз великих даних, де марні обчислення дозволяють фахівцям з аналізу даних отримувати організаційну інформацію, яка дозволяє їм отримати уявлення та прогнозувати прийняття майбутніх рішень на основі цих даних. Для невеликих компаній використовуються деякі інструменти з відкритим вихідним кодом, які базуються на хмарних обчисленнях, такі як Hadoop, Cassandra, HPCC тощо. Хмарні обчислення є досягненням для такого роду додатків для отримання значущих результатів.

До прикладу супутникове зондування Землі генерує велику кількість необроблених зображень. Обробка цих даних вимагає високої обчислювальної потужності як на вході, так і на виході. Великі обсяги даних можуть бути передані з локальної станції на землі до хмарних обчислювальних центрів для додаткової обробки. Хмарні обчислення забезпечують точну інфраструктуру для такого роду послуг. Хмарні додатки, як правило, доступні через веб-браузер і можуть працювати в будь-якому місці і в будь-який час завдяки підключенню до Інтернету. Додатки для підвищення продуктивності в хмарі виконують аналогічні завдання.

- Модель довіри на основі інфраструктури відкритих ключів – така модель покладається на декілька провідних вузлів для забезпечення безпеки всієї системи. Сертифікати повноважень для посадових осіб підписуються ЦСК, оскільки вона занадто покладається на первинні вузли, архітектура ІПК може призвести до нерівномірного навантаження або єдиної точки відмови.
- Модель довіри на основі мережевої топології - модель довіри побудована на основі мережевої архітектури. Довіра кожного суб'єкта оцінюється на основі його положення в топології системи і, як правило, використовує

алгоритм обходу дерева або графа. Механізм управління довірою в цій моделі є відносно простим. Однак, через високу складність мережевого середовища, значення довіри часто є неточними, що може призвести до ризиків безпеки в системі

- Модель довіри, заснована на поведінці - модель використовує історичні торгові записи для обчислення довіри. Довіра одного суб'єкта досягається шляхом врахування як попереднього торгового досвіду, так і рекомендацій інших вузлів. Значення довіри в цій моделі є відносно повним і надійним, але в той же час пов'язане з великими обчислювальними та іншими навантаженнями
- Модель суб'єктивної довіри - розподілені додатки часто стикаються з двома основними сценаріями безпеки. По-перше, програми користувачів можуть містити шкідливий код, який може скомпрометувати або послабити ресурси. По-друге, під впливом мережевих атак ресурси можуть пошкодити програми користувачів. Таким чином, модель довіри, заснована на предметній логіці, поділяє довіру на кілька підкатегорій: довіра до виконання, довіра до коду, довіра до авторитету, пряма довіра, довіра до рекомендацій тощо. Крім того, він розробляє різні стратегії для кожного типу довіри. Довіра - це особисте рішення щодо певного рівня особистості або поведінки суб'єкта. Суб'єкт (1) довіряє суб'єкту (2) означає, що (3) вірить, що (4) виконає певні дії в даній ситуації. Теорія ймовірності, наприклад теорія або нечітка математика, є основним інструментом для визначення впевненості. Але в загальному випадку впевненість у собі не може відображати невизначеність і викликає лише навколо ймовірнісні моделі, які є занадто формальними і далекими від істинної суті довірчого управління. В літературі запропоновано нову модель довіри до себе на основі хмарної моделі, яка може краще описувати невизначеність та випадковість. Є й інші недоліки, такі як те, що неможливо досягти цілісності сертифікату ідентичності, а поведінка і механізм настільки складні, що важко досягти системи на його основі.

- Модель довіри на основі домену: модель довіри в основному використовується в сіткових -обчисленнях. Вона розділяє мережеве середовище на кілька доменів довіри і розрізняє два типи довіри. Один з них - це відносини довіри в домені, а інший - відносини довіри між доменами. Для них встановлюються різні стратегії. Механізм цієї моделі розумний тим, що оскільки вузли в одному домені, як правило, більш знайомі, вони, як правило, мають більш високий ступінь довіри один до одного. Цей алгоритм має низьку обчислювальну складність, оскільки розрахунок довіри в домені залежить тільки від кількості вузлів в домені, а довіра між доменами залежить тільки від кількості доменів. Доменну модель можна розглядати як компроміс між PKI та мережевою топологією. Але, як і інфраструктура відкритих ключів (PKI), вона може викликати вузькі місця в мережі і єдину точку відмови, а також ігнорувати рішення довіряти незалежності суб'єктів.
- Динамічна модель довіри - побудова динамічних довірчих відносин повинна вирішити наступні математичні задачі: Визначитися з простором ступенів довіри. Він завжди визначається нечіткою логікою. Спроектувати механізм набуття значення довіри. Існує два види методів: прямі та непрямі[7].

2. ОРКЕСТРАЦІЙНИЙ ЕЛЕМЕНТ СИСТЕМИ. КЛЮЧОВІ АСПЕКТИ ПРОЕКТУВАННЯ

2.1 Дослідження проблематики та викликів оркестрації систем нового покоління

Провівши аналіз літератури, зв’язаної з балансуванням навантаження та оркестрації сучасних систем було досліджено виклики з якими зіштовхується кожен, хто має на меті побудувати систему яка має на меті обробляти великі вхідні масиви даних, на основі такого дослідження були виділені наступні виклики:

Безпека та конфіденційність - IoT на основі SDN та туманних мереж є вразливими до атак нового потоку, які можуть вивести з ладу IoT на основі SDN шляхом виснаження комутаторів або контролера. У цьому сенсі автори дослідження представляють інтелектуальний механізм безпеки (SSM) для захисту від атаки нового потоку в IoT на основі SDN, який відрізняє атаку нового потоку від звичайного сплеску потоку, перевіряючи швидкість потрапляння записів потоку. Всі механізми управління та захисту сховища абстрагуються від апаратних пристроїв в площині даних і встановлюються всередині контролера, що дозволяє централізовано приймати рішення на основі політик безпеки. Таким чином, коли хост відправляє пакети управління сховищем і трафік даних на інший хост в мережі, контроль безпеки, такий як аутентифікація і заповнення, відбувається на рівні управління, а не на рівні пристрою. Що стосується безпеки безсерверних і FaaS, деякі початкові роботи починають забезпечувати ізоляцію мікросервісів і безсерверних обчислень[8].

Контейнери можуть мати вбудовані вразливості через неправильні конфігурації або просто за фактом включення виконуваних двійкових файлів з порушеннями безпеки. В роботі автори пропонують підхід Security-as-a-Service для хмарних додатків на основі мікросервісів, що забезпечує гнучку інфраструктуру моніторингу та застосування політик для мережевого трафіку з

метою захисту хмарних додатків. На жаль, існує все ще мало досліджень і рішень, спрямованих на вирішення нових проблем безпеки в цій галузі. Що стосується прикордонних і туманних обчислень, існує аутентифікація на різних рівнях шлюзів як основну проблему безпеки в туманних обчисленнях.

Мультиадресна автентифікація або техніка інформаційної технології приманки також були запропоновані авторами для протистояння зловмисникам шляхом маскування інформації, щоб запобігти зловмисникам ідентифікувати реальні конфіденційні дані клієнта. Автори досліджують ідею збереження конфіденційності глобальних ідентифікаторів, які є універсально дійсними для суб'єкта. Така функція має важливе значення для обробки авторизації у великомасштабному розподіленому середовищі. Автори представляють схему агрегації даних, що зберігає конфіденційність, для IoT з підтримкою туманних обчислень, яка може агрегувати дані гібридних IoT-пристроїв в одне ціле в деяких реальних IoT-додатках, так що приватні дані користувача приховуються. Нещодавно автори в визначили основні атаки, які можуть відбуватися в Edge-Fog і IoT, включаючи, серед інших: DDoS, атака маршрутизації, атака вузла поглинання, атака введення в оману щодо напрямку, атака чорної діри, атака затоплення, атака Sybil або атака Spoofing. Для протидії цим атакам основні контрзаходи на мережевому рівні сьогодні зосереджуються на забезпеченні конфіденційності, цілісності і доступності. З цією метою нові механізми наскрізного шифрування, спеціально розроблені для IoT на різних рівнях , включаючи однорангову аутентифікацію і управління узгодженням ключів, можуть бути організовані і налаштовані на вимогу на периферії в якості безпечних VNF.

Питання витоку даних, та їх ненадійність - граничні ресурси за своєю природою є нестабільними. Поява “туману” розширює хмару до межі, де пристрої кінцевих користувачів можуть використовуватися в якості інфраструктури для розгортання послуг або сервісних функцій. Крім того, оскільки туман все частіше використовується для підтримки перехідних функцій FaaS, наприклад, для підтримки функцій мобільності, що залежать від контексту,

функції є ефемерними, що зумовлює швидкість зміни набагато вищу, ніж у хмарних середовищах. Такий підхід створює значні проблеми для різних функцій, які забезпечують оркестрування. Опис ресурсів та функціональності не завжди може бути точним, оскільки він може швидко застаріти, що ускладнює надійне розгортання та гарантії угоди про рівень обслуговування (SLA).

Таким чином, виявлення має бути динамічним, щоб скористатися перевагами нових ресурсів, коли вони стають доступними, а також відійти від виведених з експлуатації/несправних ресурсів. Крім того, моніторинг повинен завжди мати актуальну інформацію, уникаючи застарілих даних та емпірично доповнюючи самодекларацію пристроїв.

Виявлення мобільних периферійних пристроїв шляхом сканування всіх підключених комунікаційних інтерфейсів та складання списку всіх локально доступних мобільних периферійних пристроїв є першим поширеним підходом . Прикладом такого поширеного підходу є Foggy , який покладається на централізований сервер оркестрування та реєстр контейнерів для розгортання. Однак, високий відтік може застерігати від цієї практики, оскільки це може виснажити батареї віддалених пристроїв або збільшити рахунок за електроенергію постачальників інфраструктури. На відміну від стабільного надання ресурсів у хмарних центрах обробки даних, периферійні пристрої можуть бути динамічно вимкнені, щоб задовольнити робочі навантаження на периферії та вимоги до затримок, місцезнаходження та приватності.

Таким чином, периферія є нестабільним операційним середовищем, де доступність ресурсів може значно змінюватися з плином часу і розподіляється між декількома доменами тих, хто експлуатує периферійні ресурси. Туман працює як стабілізаційний шар, пропонуючи більш надійну інфраструктуру в безпосередній близькості від периферійних пристроїв. Проте, робота з оркестровкою в ненадійних середовищах пов'язана зі специфічними проблемами на кожному з етапів процесу оркестровки.

За неоднорідністю, невід'ємною частиною завдання оркестрування є робота з ресурсами, що мають різну природу та методи доступу, і які управляються в рамках різних адміністративних доменів. Крім того, fog-парадигма пропонує альтернативу централізованій моделі хмари. Таким чином, будь-яка спроба вирішити вищезазначені проблеми за допомогою центральних елементів для моніторингу, планування, конфігурації тощо підірве переваги дезагрегації.

Такі виклики вимагають зміни певних підходів до оркестрування. По-перше, розподілене оркестрування необхідне для реалізації потенціалу парадигми туману, де елементи оркестровки керують різними граничними доменами і координують свої дії в ієрархічній або одноранговій манері. Наявні на сьогоднішній день інструменти пропонують розробникам високорівневі інструменти для створення взаємопов'язаних потоків. Однак, вони пристосовані спеціально для функцій IoT. Необхідні більш загальні інструменти для підтримки широкої і індивідуальної координації між розподіленою мережею оркестраторів. По-друге, необхідний складний рівень абстракції, щоб приховати складність гетерогенності від процесів розробки і розгортання додатків. Необхідні набори інструментів, які не тільки спрощують завдання виявлення і моніторингу ресурсів, але і наскрізного управління життєвим циклом, а також створення адаптивної політики і механізмів міграції. Управління гетерогенними ресурсами в різних адміністративних доменах вже є складним завданням, але незалежність між управлінням ресурсами і плануванням робочого навантаження на "хмарних" пристроях збільшує складність їх оркестрування.

Швидкість FaaS забезпечує гнучку, високодинамічну конфігурацію. FaaS розділяє мікросервіси на менші програмні фрагменти, які можуть бути виконані дуже швидко (ціна, заснована на використанні процесора/пам'яті, є сильним стимулом для оптимізації виконання функцій). Менші одиниці виконання, які завершуються за лічені секунди, краще підходять для ресурсів з коротким терміном служби, де збої є звичайним явищем. Це виклик для оркестраторів, які повинні вирішити, де виконувати дану функцію FaaS і перепланувати (або

виконати випереджувальне виконання), щоб впоратися зі збоями. Повільна, пакетна глобальна оптимізація більше не є варіантом. Замість цього необхідно дослідити методи онлайн, з дедлайнами, щоб скористатися перевагами гнучкості функцій. Основною особливістю безсерверних обчислювальних архітектур є можливість/необхідність розгортання нових екземплярів в часовому масштабі. Це також вірно для підтримки флеш-подій, коли мільйони клієнтів заходять на веб-сайт, наприклад, для конкретного стимулювання збуту.

Функції NFV повинні бути розгорнуті/розгорнуті в найменші проміжки часу. Контейнери є основною одиницею розгортання для безсерверних і багатьох функцій NFV. Служби FaaS на основі контейнерів, як правило, повторно використовують один і той же контейнер для виконання декількох функцій, але навіть при такій оптимізації безсерверні функції працюють значно повільніше, ніж контейнери, при низьких обсягах запитів . Можливо використовувати в такому випадку деякі хитрощі, щоб уникнути накладних витрат, пов'язаних з використанням постійних сховищ блоків для отримання даних та конфігурації. Планувальник, який знає, що дві різні функції значною мірою покладаються на ті самі пакети, може приймати кращі рішення щодо їх розміщення.

Локальність сеансу є важливим фактором: якщо виклик функції є частиною тривалого сеансу з відкритими TCP-з'єднаннями, оркестратор повинен запускати її на машині, де підтримуються TCP-з'єднання (уникаючи перенаправлення трафіку через проксі-сервери). Управління мобільністю, однак, створює додаткові проблеми для оркестратора. Крім того, локальність даних буде важливою для запуску безсерверних функцій, які витягують/проштовхують або сканують масивний стан. Оркестрантам можуть знадобитися можливості прогнозування, щоб передбачити, які дані буде зчитувати конкретна функція, і переконатися, що вони будуть доступні для роботи вчасно.

Всі ці проблеми з локалізацією даних і коду існують на рівні центру обробки даних, але вони стають більш важливими на периферійних вузлах мережі. Необхідні передові методи для визначення того, з яким периферійним вузлом слід розділити робоче навантаження, і яка частина робочого

навантаження повинна бути розподілена на кожен вузол, оскільки гетерогенність і відтік є основними проблемами розгортання.

2.2 Життєвий цикл даних

Життєвий цикл даних визначається операціями з ними протягом їхнього існування. Дані перебувають у русі, переміщаючись від джерела до точки первісного приймання, а звідти до систем зберігання та аналізу. Упродовж усього життєвого циклу і під час переміщення інфраструктурою обчислень, зберігання та мережі даним потрібен захист.

Незважаючи на незліченну кількість переваг управління даними, можна виділити три основні цілі цієї моделі:

Безпека - основною метою управління життєвим циклом даних є забезпечення відповідного зберігання даних. Створюючи протоколи для управління даними з моменту їх створення до моменту видалення, користувач допомагає запобігти доступу до них зловмисників та інших несанкціонованих користувачів, а також їх пошкодженню шкідливим програмним забезпеченням та іншими інфекціями.

Доступність - хоча однією з цілей DLM є забезпечення недоступності даних для певних користувачів, не менш важливою метою є забезпечення доступності даних для потрібних користувачів у потрібний час. Якщо це не так, то численні процеси та робочі процеси можуть бути перервані або збійні.

Цілісність - ще однією метою DLM є підтримка цілісності даних, що означає, що у базі даних створюються і зберігаються тільки найсвіжіші і високоякісні дані. Без DLM користувачі могли б отримувати доступ, використовувати та зберігати застарілі або різні версії даних.

Генерація даних – це етап створення даних. Існує безліч джерел даних з різними характеристиками, які необхідно враховувати. Наприклад, датчики на

виробничому обладнанні або безпілотних автомобілях генерують величезні обсяги даних. Непрактично було б передавати всі ці дані в центральну систему обробки. Замість цього їх можна обробляти прямо в місці створення, на кордоні мережі, а потім надсилати результати первинного опрацювання в центр дротовою мережею або зберігати на фізичних пристроях зберігання, якщо обсяг занадто великий, а швидкість має вирішальне значення.

Якщо говорити більш детально, то створення даних складається зі збору та отримання даних. Стратегія DLM визначає типи даних де вони використовуються, для чого вони використовуються і хто може їх використовувати. Поряд із визначенням типів файлів, зазвичай вказується чутливість даних.

Зберігання даних – після генерації даних наступає процес первинної оброки та зберігання та вживання заходів щодо захисту даних для запобігання їх втрати. Повинна існувати ефективна стратегія резервного копіювання та запобігання втратам, щоб гарантувати безпеку даних протягом усього їх життєвого циклу.

Важливо встановити політику збереження даних, яка включає архівування даних та інші процеси управління життєвим циклом. Деякі ресурси не повинні зберігати надлишкову інформацію, оскільки користувачі можуть генерувати помилкові уявлення та приймати неправильні рішення. Заходи безпеки даних є критично важливими на всіх етапах життєвого циклу даних в управлінні даними.

Використання даних – як тільки було виконано належні процедури захисту даних, користувачі можуть легко переглядати дані в інформаційних системах, вносити зміни та зберігати оновлені дані. Користувачі також можуть надавати доступ до даних іншим відділам своєї групи або організаціям за межами своїх груп, оскільки вони знають, що інформація має надійний захист та протоколи резервного копіювання.

На цьому етапі відбувається синтез даних, коли інформаційна система використовує індуктивні міркування для перевірки даних. Це передбачає

використання програмного забезпечення для порівняння різних типів даних для забезпечення точності.

Крім того, потрібно визначити різні плани відновлення даних. Тобто, якщо станеться збій, у вас повинен бути план продовження доступу до даних. Це може бути використання тимчасової резервної копії, поки користувач відновлює диск, або інша стратегія. Тут є тонка різниця між зберіганням даних та їх архівацією. Зберігання даних відноситься до активних даних. Користувачі і системи повинні мати можливість швидкого доступу до них в своїх повсякденних процесах. Вони повинні бути як в живих базах даних, так і в резервних копіях. Якщо одна з них виходить з ладу, стратегії повинні мінімізувати час простою.

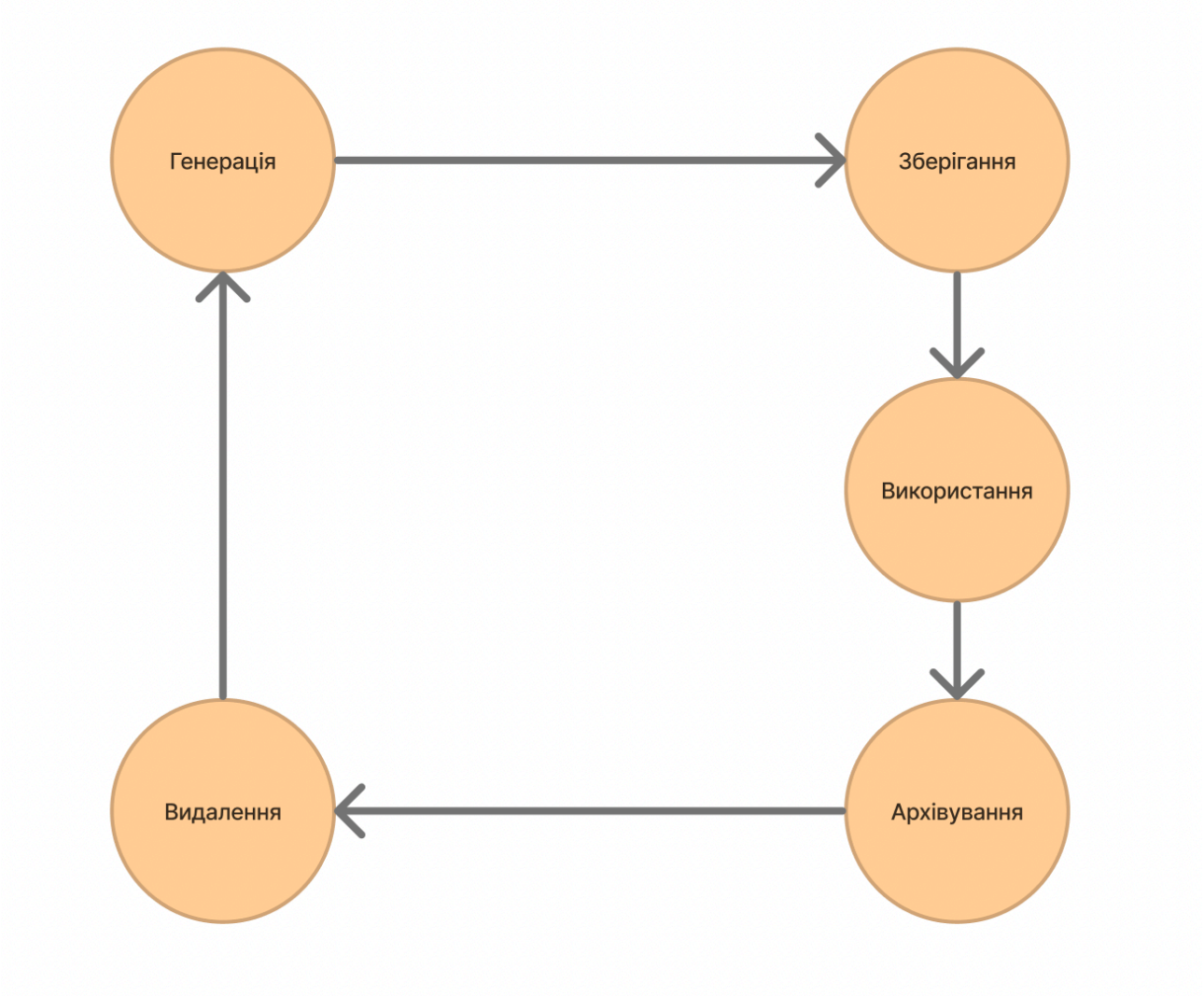


Рис 2.2.1 – Життєвий цикл даних

Четвертий етап життєвого циклу даних - архівування. Цей етап передбачає копіювання даних в інформаційній системі, аби була змога зберігати їх і звертатися до них у майбутньому. Архівування даних дає користувачам запис про застарілі дані та дозволяє їм завантажувати старішу інформацію. Як правило, можна позначати дані як активні, що означає, що даний час ці дані використовуються і оновлюються. Якщо дані застаріли, то вони видаляються з усіх активних середовищ, таких як база даних або інформаційна система.

На цьому етапі стане в пригоді комплексна платформа оркестрації, що охоплює всю інфраструктуру від ЦОД до хмари. Раніше для цієї мети можна було використовувати найкращі пристрої та ПЗ для зберігання, але зараз все змінилося. Кожен компонент цього набору інструментів локально оптимізує частину життєвого циклу.

Видалення даних - заключний етап передбачає видалення архівних даних, щоб користувач мав більше місця для зберігання інформації. Цей етап передбачає видалення копії кожного фрагмента архівних даних, залежно від того, яка була вибрана стратегія видалення даних . Видалення даних може відбуватися за наступними критеріями:

- на основі обсягу: видалення даних, як тільки обсяг архівів, які вони зберігають, досягне певного обсягу.
- залежно від часу: Компанії можуть вибрати певний проміжок часу для видалення архівних даних.

2.3 Аналіз використання IoT систем для комерційного ринку. Використання індустріальних датчиків

В результаті дослідження індустріального ринку IoT систем було виділено наступні найпопулярніші датчиків:

- Датчики вібрації - часто вібрація може сигналізувати про потенційну поломку, яка знаходиться на горизонті. Для перевірки обладнання, яке вібрує, може бути направлений технік з технічного обслуговування, який негайно виконає регулювання або ремонт.
- Датчики температури - датчики температури зазвичай гарантують, що об’єкт залишається в межах безпечного температурного діапазону. Це може запобігти перегріву такого активу, як котел. Наприклад у харчовій промисловості дуже важливо, щоб продукти залишалися в межах безпечного діапазону в морозильній камері або холодильному обладнанні.
- Датчики наближення - датчики допомагають попередити оператора, коли одна частина обладнання знаходиться занадто близько до іншої частини обладнання. Одне з поширених застосувань таких датчиків - в автопарку. Якщо вантажівка або навантажувач збирається щось вдарити, датчик попередить оператора до того, як відбудеться зіткнення.
- Датчики газу – вони можуть попередити обслуговуючий персонал про витік диму або іншого небажаного газу в приміщення. У випадку з небезпечними газами це доповнення може зробити значний внесок у здоров'я та безпеку працівників.
- Датчики охорони - можуть бути розміщені біля ключових вікон і дверей для моніторингу руху в цих зонах. Якщо компанія вимагає високого рівня безпеки, датчики безпеки можуть допомогти виявити небажаних відвідувачів.
- Датчики вологості - залежно від чутливості розташованого поблизу обладнання, на об'єкті може знадобитися моніторинг рівня вологості навколишнього середовища. Можна встановити прийнятний діапазон, і коли рівні виходять за межі діапазону, можна негайно відправляти оповіщення.
- Датчики тиску - під час установки технічний фахівець може встановити максимальний тиск, допустимий для конкретного об'єкта. Якщо обладнання перевищує цей тиск, цей актив може бути автоматично

налаштований на відключення в якості запобіжного заходу безпеки. Після цього може бути виданий наряд на виконання ремонту.

- Датчики рівня - можуть контролювати рівень певної рідини в обладнанні. Коли низький рівень загрожує продуктивності об’єкту, може бути надіслано попередження, щоб рідини можна було поповнити. Крім того, датчики рівня можуть вимірювати рівень порошку або інших матеріалів, навіть сміття у сміттєвому контейнері, щоб спонукати до дій з технічного обслуговування.
- Інфрачервоні датчики - випромінюють або виявляють інфрачервоне випромінювання або вимірюють виділене тепло. Інфрачервоні датчики можуть допомогти контролювати такі речі, як потік крові в системах охорони здоров'я або сигнали дистанційного керування.
- Датчики крадіжки - часто використовувані в роздрібній торгівлі, датчики крадіжки можуть бути прикріплені до цінних предметів, щоб гарантувати, що вони залишаються в межах прийнятного місця. Установи можуть прикріплювати датчики крадіжки до дорогих інструментів або інших предметів, схильних до зникнення.

Широкий спектр галузей промисловості може отримати вигоду від використання промислових датчиків, підприємства, орієнтовані на сільське господарство і охорону здоров'я, потребують моніторингу обладнання та об’єктів, управління автопарком і нафтогазові об'єкти також можуть використовувати датчики для відстеження продуктивності машин. Підприємства в сільськогосподарському секторі зазвичай використовують датчики IoT для моніторингу широкого спектру обладнання. Датчики можуть допомогти з метеостанціями, іригаційним обладнанням і сільськогосподарською технікою

Датчики IoT можуть допомогти медичним організаціям контролювати не тільки обладнання та інвентар, а й персонал і пацієнтів. Багато медичних галузей використовують пристрої Інтернету речей для моніторингу загального стану здоров'я і життєво важливих показників пацієнтів. У деяких випадках ця

інформація може бути відправлена постачальнику медичних послуг для довгострокового догляду та лікування.

Виробничі підприємства можуть використовувати широкий спектр датчиків IoT для моніторингу активів і обладнання, а також виробничих ліній. У деяких випадках машини можуть бути запрограмовані на автоматичне відключення, коли датчики досягають порогових значень, щоб запобігти поломкам, дорогим пошкодженням або нещасним випадкам.

Незалежно від того, чи експлуатує користувач парк автомобілів, вантажівок або навантажувачів, датчики можуть допомогти йому ефективно відстежувати все. Датчики, які досягають певних порогових значень пробігу або використання, можуть сигналізувати про необхідність технічного обслуговування. Датчики наближення можуть гарантувати, що машини не зіткнуться з іншими транспортними засобами або обладнанням.

Нафтові і газові вишки, трубопроводи та інше обладнання отримують вигоду від датчиків IoT, які можуть виявляти витоки і відстежувати об’єкти. Крім того, можуть спрацьовувати оповіщення, якщо будь-яка кількість заходів випадає з безпечних діапазонів, викликаючи профілактичне обслуговування в міру необхідності.

Коли датчики відзначають, що певні порогові значення або умови були досягнуті, вони відправляють дані в хмару для обробки. Інформація передається в CMMS для подальшого аналізу і створення основи для прийняття розумних бізнес-рішень.

Процес передачі даних з датчиків до CMMS відбувається наступним чином:

- Передача даних - датчики прикріплюються до критично важливих елементів, встановлюються пороги та діапазони, і починається моніторинг.
- Завантаження даних у систем - вони можуть постійно відправлятися в хмару для обробки.

- Обробка даних - програмне забезпечення визначає, чи перевищені порогові значення, або виконує більш складний аналіз.
- Публікація даних в CMMS. Інформація, передається в CMMS для звітів, подальшого розгляду або як основа для майбутніх бізнес-рішень.

Промислові датчики IoT будуть відігравати все більш важливу роль в широкому спектрі галузей протягом наступного десятиліття. Вартість технології продовжує знижуватися, в той час як економія коштів, підвищення безпеки і виробничі переваги будуть рости. Пошук правильних датчиків для моніторингу ваших найбільш важливих активів забезпечить вас великою кількістю даних. Це потім послужить основою для ефективної і дієвої програми профілактичного обслуговування.

2.4 Застосування нечіткої логіки на рівні обробки завдання

Нечітка логіка - це підхід до обробки змінних, який дозволяє обробляти декілька можливих значень істинності через одну і ту ж змінну. Нечітка логіка намагається вирішувати проблеми з відкритим, неточним спектром даних і евристик, що дає можливість отримати масив точних висновків.

Вона призначена для вирішення проблем шляхом розгляду всієї доступної інформації та прийняття найкращого можливого рішення з урахуванням вхідних даних.

Нечітка логіка - це евристичний підхід, який дозволяє більш просунуту обробку дерев рішень і кращу інтеграцію з програмуванням на основі правил. Також це єузагальненням стандартної логіки, в якій всі твердження мають значення істинності один або нуль. У нечіткій логіці твердження можуть мати значення часткової істинності, наприклад, 0,9 або 0,5. Теоретично це дає підходу більше можливостей для імітації реальних обставин, де твердження абсолютної істинності або хибності зустрічаються рідко. Така логіка може

використовуватися кількісними аналітиками для покращення виконання своїх алгоритмів.

Через схожість зі звичайною мовою, нечіткі алгоритми порівняно просто кодувати, але вони можуть вимагати ретельної перевірки та тестування.

На відміну від чітких множин, нечітка множина допускає часткову приналежність до множини, яка визначається ступенем приналежності, що позначається через μ, який може приймати будь-яке значення від 0 (елемент взагалі не належить до множини) до 1 (елемент повністю належить до множини).

Очевидно, що якщо прибрати всі значення приналежності, крім 0 та 1, то нечітка множина згорнеться до чіткої множини.

Рисунок 2.1 – принцип роботи нечіткої логіки

Функція належності множини - це відношення між елементами множини та їх ступенем належності. Ілюстрація того, як функції належності можуть бути застосовані до вимірювання градацій будь яких вимірювальних процесів.

Найбільш поширеним методом обчислення об'єднання двох нечітких множин є застосування оператора максимуму на множинах. Існують і інші

методи, в тому числі використання оператора добутку двох множин. Аналогічно, найбільш поширеним методом обчислення перетину двох нечітких множин є застосування оператора мінімуму на множинах. Доповнення нечіткої множини обчислюється шляхом віднімання функції належності множини від 1.

Нечітка система - це сховище нечітких експертних знань, які можуть міркувати про дані в нечітких термінах замість точної булевої логіки. Експертні знання являють собою сукупність нечітких функцій приналежності та набір нечітких правил, які називаються базою правил.

Базова конфігурація нечіткої системи може виглядати наступним чином:

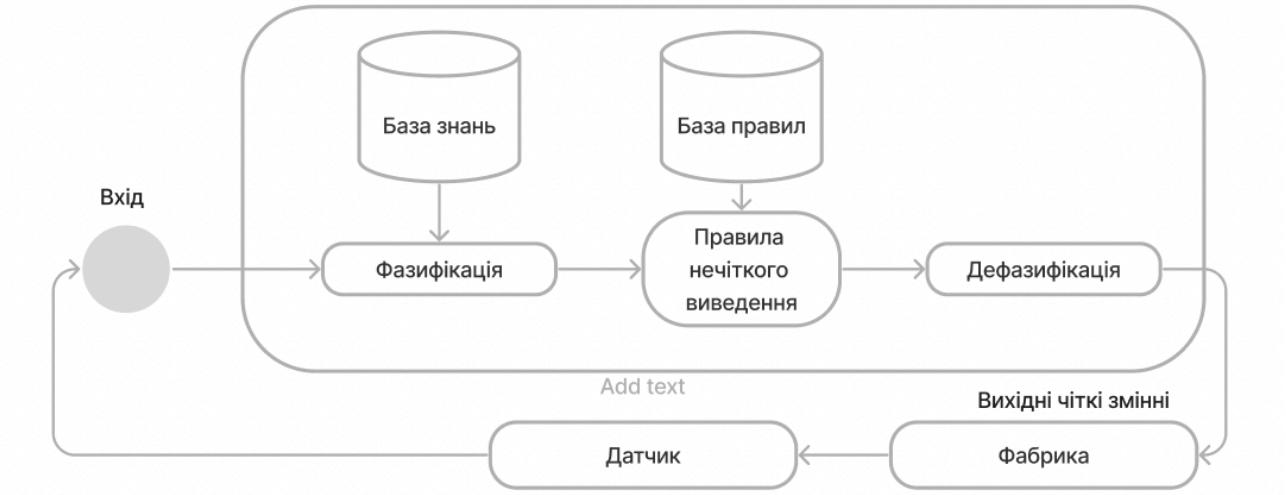


Рисунок 2.2 – Конфігурація нечіткої системи

Основні компоненти контролера нечіткої логіки (КНЛ/FLC):

- Нечіткий логічний елемент(Fuzzifier) - роль нечіткого логічного елемента полягає в перетворенні чітких вхідних значень у нечіткі значення.
- База нечітких знань(Knowledge base) - зберігає знання про всі нечіткі зв'язки між вхідними та вихідними даними. Вона також має функцію

приналежності, яка визначає вхідні змінні для бази нечітких правил і вихідні змінні для контрольованої установки.

- База нечітких правил(Rule base) - зберігає знання про функціонування процесу предметної області.
- Механізм виводу - виступає ядром будь-якої СППР. В основному він імітує рішення людини, виконуючи наближені міркування.
- Дефазифікатор - роль дефаззифікатора полягає в перетворенні нечітких значень в чіткі значення, отримані від системи нечіткого виводу.[9]

Етапи проектування КНЛ:

- Ідентифікація змінних - Тут повинні бути визначені вхідні, вихідні та змінні стану установки, що розглядається.
- Конфігурація нечітких підмножин - Всесвіт інформації поділяється на ряд нечітких підмножин, і кожній підмножині присвоюється лінгвістична мітка. Важливо переконатися, що ці нечіткі підмножини включають всі елементи універсуму.
- Отримання функції належності -отримання функції належності для кожної нечіткої підмножини, яку було отримано на попередньому кроці.
- Конфігурація бази нечітких правил -формулювання бази нечітких правил, призначивши відношення між нечітким входом і виходом.
- Нечіткість - На цьому кроці ініціюється процес нечіткості.
- Об'єднання нечітких виходів - Застосовуючи нечіткі приблизні міркування, знаходиться і об’єднується нечіткий вихід.
- Дефазифікація - формування чіткого результату.

Фаззифікатор відображає реальний чіткий вхід у нечітку функцію, визначаючи таким чином "ступінь приналежності" входу до нечіткої концепції.

У ряді контролерів значення вхідних змінних відображаються на діапазон значень відповідного універсуму дискурсу. Діапазон і роздільна здатність

вхідних нечітких множин та їх вплив на процес фаззифікації розглядаються як фактори, що впливають на загальну ефективність роботи контролера.

База знань включає в себе знання прикладної області та супутніх цілей управління. Вона може бути розділена на базу визначень, що використовуються для вираження лінгвістичних правил управління в контролері, і базу правил, що описує знання, якими володіють експерти предметної області. Інтуїтивно зрозуміло, що база знань є основним елементом нечіткого контролера, оскільки вона буде містити всю інформацію, необхідну для виконання завдань його функціонування. Різні дослідники застосовували методи для тонкої настройки бази знань нечіткого контролера, багато з них використовували інші дисципліни ШІ, такі як генетичні алгоритми або нейронні мережі.

Механізм виведення забезпечує логіку прийняття рішень контролером. Він виводить нечіткі дії управління, використовуючи нечіткі наслідки і нечіткі правила виведення. У багатьох аспектах це можна розглядати як емуляцію прийняття рішень людиною.

Процес дефаззифікації перетворює нечіткі значення управління в чіткі величини, тобто пов'язує окрему точку з нечіткою множиною, враховуючи, що точка належить до підтримки нечіткої множини. Існує багато методів дефаззифікації, найвідомішим з яких є метод центру області або центру ваги.

2.5 Модифікація транспорту даних. Рівень запиту

Під час дослідження було проведено аналіз підходу REST та фреймоворку gRPC. Зміни до внутрішньої організації запитів - в стандартній реалізації мікросервісних систем, що використовують оркестратор використовують REST, але в даному випадку буде доцільно використовувати gRPC, адже в такому випадку забезпечується швидкість передачі та безпека переданих даних, також строга типізація повідомлень буде відігравати важливу роль в розширенні та

підтримки такої системи, адже в повідомленнях завжди буде лише необхідна інформація.

RPC або віддалений виклик процедур був створений першим, в 1970-х роках і вважається попередником REST. Хоча, безумовно, можна використовувати REST API без знання RPC, але така модель передедачі даних також є серйозним конкурентом серед аналогів. RPC викликає функцію на віддаленому сервері, але на відміну від більш нових API, він використовує певний формат і повинен отримувати такий же формат у відповідь.

Основна концепція RPC API схожа на концепцію REST API. RPC API визначає правила взаємодії та методи, які клієнт може використовувати для взаємодії з ним. Клієнти подають виклики, які використовують "аргументи" для виклику цих методів. Однак техніка знаходиться в URL-адресі з RPC API. Аргументи, які викликають методи, знаходяться в рядку запиту.

Як варіант архітектури RPC, gRPC був створений компанією Google в 2015 році для прискорення передачі даних між мікросервісами та іншими системами, які повинні взаємодіяти. Ця архітектура API дещо відрізняється від попередніх API з кількох причин.

По-перше, gRPC використовує інший формат. Замість того, щоб використовувати JSON, API використовує Protobuf. Protobuf також був розроблений компанією Google і є нейтральним до мови і платформи. Він схожий на XML, але вважається більш ефективним.

Ця структура API також використовує HTTP2, замість оригінального HTTP1, і є значно швидшою, ніж оригінальний RPC. Це означає, що його легше реалізувати, оскільки він може передавати повідомлення до 10 разів швидше, ніж попередні версії. Для великих додатків це дає можливість управляти комунікаційною стороною справи, хоча це повільніше, ніж REST при реалізації API. Нарешті, gRPC використовує власну генерацію приватного коду, а не Swagger для комунікацій.

Основні відмінності роботи REST від gRPC:

Protobuf замість JSON/XML - формати повідомлень JSON і XML широко використовуються як REST API, так і RPC API для обміну повідомленнями. Хоча JSON є найпопулярнішим вибором, завдяки своїй гнучкості та нейтральності до мови/платформи, він може бути незручним та повільним у використанні.

На відміну від REST і RPC, gRPC долає проблеми, пов'язані зі швидкістю і вагою - і пропонує більшу ефективність при відправці повідомлень - за рахунок використання формату повідомлень Protobuf (буфери протоколу). Серед подробиць визначають: агностичний до платформи і мови, як JSON. Серіалізує та десеріалізує структуровані дані для передачі через двійковий формат Як сильно стиснутий формат, він не досягає рівня читабельності JSON.Прискорює передачу даних, усуваючи багато обов'язків, якими керує JSON, щоб він міг зосередитися виключно на серіалізації та десеріалізації даних. Передача даних відбувається швидше, оскільки Protobuf зменшує розмір повідомлень і служить легким форматом обміну повідомленнями

Побудований на HTTP 2 замість HTTP 1.1 - ще одним способом підвищення ефективності gRPC є використання протоколу HTTP 2. HTTP відноситься до протоколу передачі гіпертексту і існує з 1989 року і є методом зв'язку в Інтернеті. У той час як REST API використовують HTTP 1.1, gRPC API використовують HTTP 2. Між цими двома протоколами є деякі помітні відмінності.

HTTP 1.1: випущений в 1997 році, цей протокол зазвичай вважається стандартом для зв'язку у Всесвітній павутині. По суті, він передає інформацію між комп'ютером і веб-сервером, який може бути локальним або віддаленим. Комп'ютер-клієнт надсилає текстовий запит і отримує у відповідь ресурс. У звичайному використанні WWW це означає HTML-сторінку або PDF-документ.

HTTP 2: Цей протокол був опублікований в 2015 році і більшість сучасних браузерів використовують його. Той факт, що Chrome, Internet Explorer і Safari використовують HTTP 2, означає, що він широко прийнятий. Однак він працює

інакше, ніж HTTP 1.1. Замість того, щоб зберігати все у звичайному текстовому форматі, HTTP 2 використовує інкапсуляцію двійкового формату. Це прискорює весь процес і дозволяє більш широкі можливості доставки даних.

Простіше кажучи, HTTP 2 швидший, ефективніший і зменшує затримку в мережі завдяки використанню мультиплексування. Використовуючи HTTP 2, gRPC пропонує три типи потокової передачі даних:

- На стороні сервера: Клієнт відправляє повідомлення із запитом на сервер. Сервер повертає потік відповідей назад клієнту. Після завершення відповідей сервер відправляє повідомлення про стан (в деяких випадках, супутні метадані), яке завершує процес. Після отримання всіх відповідей клієнт завершує процес.
- На стороні клієнта: Клієнт надсилає потік повідомлень-запитів на сервер. Сервер повертає клієнту одну відповідь. Він (зазвичай) надсилає відповідь після отримання всіх запитів від клієнта та повідомлення про стан (а іноді й метадані).
- Двонаправлений: Клієнт і сервер передають дані один одному в довільному порядку. Клієнт є тим, хто ініціює цей вид двонаправленого потоку. Клієнт також завершує з'єднання.

Вроджена генерація коду замість використання сторонніх інструментів. На перший погляд, інтеграція зі сторонніми програмами, наприклад, використання REST API, здається гарною ідеєю. Однак вона може мати деякі недоліки, і багато хто віддає перевагу власній генерації коду gRPC.

API gRPC використовують власний компілятор Protoc, який дозволяє створювати власний код. Він працює на декількох мовах і може використовуватися в поліглотських середовищах. Маються на увазі групи мікросервісів, які працюють на окремих платформах і закодовані на декількох мовах.

REST API не має такої нативної генерації коду і повинен використовувати зовнішній інструмент. Часто він використовується в парі зі Swagger, щоб

забезпечити створення декількох мов. Для багатьох це вважається недоліком, але REST продовжує користуватися популярністю.

Швидка передача повідомлень - вони в 7-10 разів швидше ніж REST при отриманні даних і приблизно в 10 разів швидше, ніж REST при відправці даних для цієї конкретної корисного навантаження. Це в основному пов'язано з щільною упаковкою буферів протоколу і використанням HTTP/2 в gRPC".

Де може використовуватись gRPC

Мікросервіси: gRPC блищить як спосіб підключення серверів в сервіс-орієнтованих середовищах. Однією з початкових проблем, яку намагався вирішити його попередник, Stubby, було з'єднання мікросервісів. Він добре підходить для широкого спектру областей: від систем середніх і великих підприємств до "веб-масштабу" електронної комерції та пропозицій SaaS.

Клієнт-серверні додатки: gRPC працює так само добре в клієнт-серверних додатках, де клієнтський додаток працює на настільних або мобільних пристроях. Він використовує HTTP/2, який покращує HTTP 1.1 як по затримці, так і по використанню мережі.

Інтеграції та API: gRPC також є способом пропонувати API через Інтернет для інтеграції додатків з послугами сторонніх постачальників. Наприклад, багато хмарних API Google доступні через gRPC.

Архітектори можуть розглядати сучасні системи як сукупність численних хмарних компонентів. Хмарність є фундаментальною для рішення, незалежно від того, чи планується цільове рішення для хмари чи ні. Легше керувати компонентом рішення, щоб дотримуватися хмарних принципів, коли його розмір невеликий. Цей процес мислення рухає архітектури в бік все більшої кількості мікро- і наносервісів. Ступінь розподіленості пропорційно збільшує потребу в ефективних і гнучких схемах інтеграції.

3. УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1 Результати заміни логіки оркестрації на рівні обробки завдань.

В ході дослідження було розглянуто метод, що обробки завдань, що передбачає вирішення, та оцінку задачі з використанням нечіткої логіки

Аби розрахувати ефективність розглянутого методу було використано емулятор PureEdgeSim, середовище моделювання, яке дозволяє вивчати Інтернет речей у великих масштабах, як розподілену, динамічну та високо гетерогенну інфраструктуру, а також додатки, які працюють на цих речах. Вона містить реалістичні моделі інфраструктури, що дозволяє проводити дослідження континууму від межі до хмари. Він охоплює всі рівні моделювання і симуляції периферійних обчислень. Він має модульну конструкцію, в якій кожен модуль відповідає за певний аспект моделювання.

В особливості такого середовища входить гнучке налаштування пристроїв, наприклад їх пропускну здатність, тип за яким вони під’єднуються до мережі, тип живлення, ємність акумулятору, кількість ядер, оперативної пам’яті та кешованого сховища. Таким чином можна зконфігурувати необхідну систему для тесту, та виміряти необхідні показники шляхом емулювання генерації завдань та доставки їх до кінцевої точки, тільки в цьому разі система зарахує завдання, як виконане.

Такі дані є необхідними оскільки, є багато факторів , що впливають на працездатність системи з вибраною конфігурацією, навіть пристрій з недостатньо широким діапазоном підключення до оркестратора може стати критичною точкою у функціонуванні спроектованої системи, таким чином сповільнивши роботу багатьох її елементів.

Було досліджено роботу алгоритму циклічної обробки, та алгоритму основаному на Нечіткій логіці, для цього була вибрана наступна конфігурація:

Таблиця 3.1.1 – Початкові налаштування симуляції оркестрації завдань

Час симуляції	10 хвилин
Кількість завдань	5670
Інтервал оновлення інформації	1 секунда
Симуляційне покриття	150x150 метрів
Дистанція між елементами	15 метрів
Максимальна кількість кінцевих девайсів	100
Мінімальна кількість кінцевих девайсів	99
Інтервал оновлення мережі	1с
Пропускна здатність WAN	25 мбіт/секунда
Пропускна здатність WI-FI	500 мбіт/секунда
Пропускна здатність Ethernet	1000 мбіт/секунда

Провівши тест за вибраною конфігурацією було отримано результати емуляції 10 хвилин одночасної роботи 100 пристроїв, які могли підключатися через WAN, wi-fi та ethernet протоколами.

Таблиця 3.1.2 – Результати симуляції оркестрації завдань

Алгоритм обчислень	Нечітка логіка	Циклічна обробка
% виконаних завдань	98,3195	81.3933

Продовження таблиці 3.1.2 – Результати симуляції оркестрації завдань

Алгоритм обчислень	Нечітка логіка	Циклічна обробка
Запущені, але не виконані ізза високої затримки (%)	0	18.6067
К-сть незавершених завдань	92	1055
Енергоспоживання(загальне)	104.4506 Вт/г	81.8768 Вт/г
Використання мережі (Загальне, с)	69.3818 секунд	92.6843 с
Середня швидкість передачі даних	451.557 Мбіт/с	348.088 Мбіт/с
Трафік	3878 Мбайт	3825.25 Мбайт

За даними, наведених в таблиці, було досліджено роботу методик обробки інформації та проемульовано наведені методики в якості механізму оркестрації завдань. За результатами дослідження найкраще себе показав алгоритм на основі нечіткої логіки показавши результативність краще аніж циклічний (за результативністю було вибрано к-сть та % повністю виконаних завдань)

3.2 Результати заміни методу транспортування даних на рівні запиту

Для порівння було вибрано 2 технології обміну даними, REST та gRPC. Ці технології використовують різні підходи щодо передачі даних, стандартизації контрактів, форматів повідомлень, тип даних, який транспортується, типізація даних, що надсилаються та вбудовані інстументи генерації клієтського коду. Спочатку було виділено особливості кожного з підходів транспорту даних, починаючи від контракту, що застосовує підхід, його наявні протоколи, особливості направленості коду, та повідомлень, що передаються (Таблиця 3.2.2)

Таблиця 3.2.2 – Порівняльна характеристика gRPC та REST

Технологія	**gRPC**	**HTTP APIs з JSON**
Контракт	Суворий (.proto)	Опціональний (OpenAPI)
Протокол	HTTP/2	HTTP
Контент повідомлень	Protobuf (малий, бінарний)	JSON (громісткий, читаємий людиною)
Нормативність	Строга специфікація	Вільна. Будь який HTTP запит валідний
Стрімінг	Клієнт-сервер, двонаправлений	Клієнт-сервер,
Підтримка браузеру	Ні, потребує grpc-web	Так
Захист	Transport (TLS)	Transport (TLS)
Клієнтська генерація коду	Так	OpenAPI + сторонній інструментарій

Аби переконатися, як насправді працює транспортна складова двох технологій, було проведено бенчмарк тест, та опрацьовано за допомогою Chrome dev Tools потік даних, що надавався через 2 вибрані технології

Таблиця 3.2.3 – Результати бенчмарк тесту для порівняння gRPC та REST

	Використання процесора	Пропускна здатність (запитів/секунду)	Швидкість передачі даних	Час, за який виконано 50% запитів	Час, за який виконано 100% запитів
REST	~76%	17.51	217.18 [кБайт/секунда]	8.567 секунд	7.972 секунди
gRPC	~43%	39.91	503.32 [кБайт/секунда]	2.831 секунда	5.023 секунди

Таким чином було досліджено, що пропускна здатність gRPC майже в 2 рази більша, за пропускну здатність HTTP, також відсоток опрацьованих запитів різниться в ~1.5-4 рази в сторону першого. Тому при проектуванні майбутньої системи було рекомендовано використовувати саме gRPC для забезпечення з'єднання оркестратор – система.

3.3. Запропонована архітектура системи. Інтеграція механізму екстреної передачі повідомлень

При компонуванні стандартної системи, між кожними проміжними вузлами може ставитися шлюз, який буде збирати інформацію від пристроїв , та в подальшому передавати її на локальну оброблюючу систему чи в хмару. В такому випадку, не знаючи, як проходить внутрішня робота шлюзу по обробці та передачі даних можна отримати ризик не отримати такі дані кінцевому споживачу. Нестабільні скачки напруги, надзвичайні ситуації, високі температури або складні кліматичні умови можуть стати серйозними чинниками, що стають на заваді передачі даних та проведенню певних дій на їх основі.

На основі таких факторів було запропоновано архітектуру, що має мінімізувати ризики зв'язаними з такими ситуаціям, така система в свою чергу має забезпечувати відповідність до запропонованого протоколу щодо забезпечення цілісності даних в критичних ситуаціях – Last Call Emergency Messaging (LCEM)

Протокол невідкладного обміну повідомленням останнього виклику – є набором правил, що мають бути виставлені у відповідності до екстреної ситуації, що в може виникнути в середовищі використання обчислюючих приладів в системі.

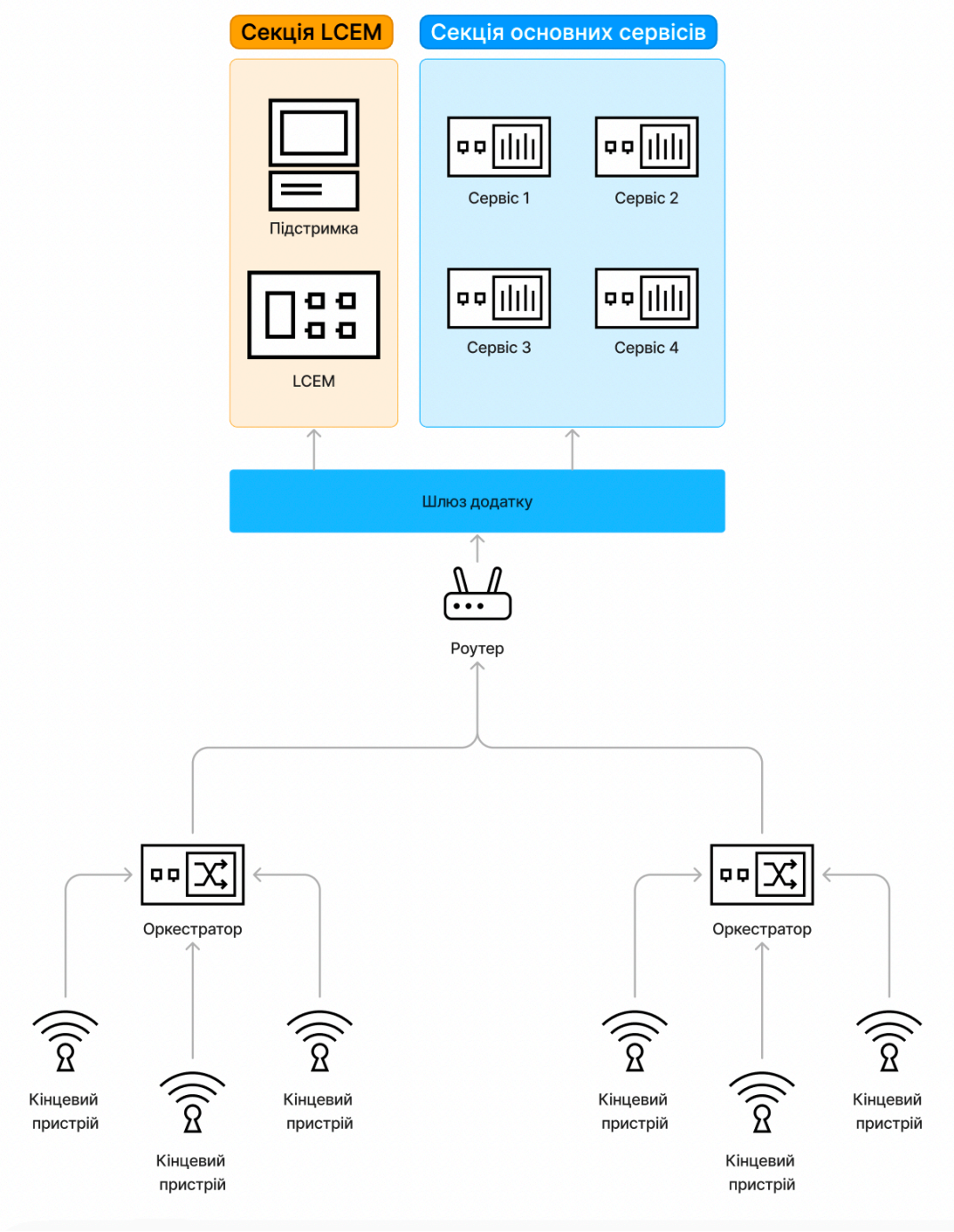


Рисунок 3.3.1 – Приклад інтеграції механізму екстреної передачі повідомлень

Такий протокол має на меті відправити останні показання приладів, якщо останні опинилися в екстреній ситуації, що виникли в середовищі, що дає змогу обслуговуючому персоналу швидко відреагувати та виконати певні дії щодо стабілізації ситуації в регіоні.

Така система завжди повинна мати первинну конфігурацію, що встановлює сутність екстреної ситуації. Це може бути висока температура, вологість, чи тиск, вміст шкідливих речовин у повітрі, чи у воді тощо.
Як тільки поріг встановлено, наступним етапом є встановлення слухача подій, відокремленою від самої системи, це може бути окремо розроблена система оперативного реагування, що лежить навіть на відокремленому середовищі чи порті, яка буде приймати тільки ці види екстрених запитів.

Використання такої системи має на меті мінімізувати наслідки не штатних ситуацій, та посилити надійність системи, завдяки безпекової допоміжної ланки.

3.4. Використання кластеризації конфігурування кінцевих пристроїв

В випадку, коли система має надавати змогу змінювати конфігурацію своїх пристроїв або мати змогу розгортатися з модифікованою конфігурацією всіх своїх ланок, важливо ретельно підбирати опції готової мережі, яка вона утворює.

Кластеризація в такому випадку допоможе значно зменшити час налаштування мережі в якій знаходяться пристрої, розподіливши пристрої на певні групи, які мають під’єднуватися до певних оркестраторів в системі.

Аби утворити правильну мережу, та мати мінімальні затримки в ній потрібно модифікувати спосіб за яким система а не людина має мати розуміння як правильно розгорнути за згрупувати пристрої між собою. Для розв’язання такого завдання було обрано використовувати алгоритм кластеризації «К-середніх».

K-means - це некерований алгоритм кластеризації, призначений для поділу немаркованих даних на певну кількість (тобто "K") окремих угруповань. Іншими словами, k-середні знаходять спостереження, які мають спільні важливі характеристики, і класифікують їх разом у кластери. Хорошим рішенням для

кластеризації є таке, що знаходить такі кластери, в яких спостереження в межах кожного кластера є більш схожими, ніж самі кластери.[10]

Існує незліченна кількість прикладів, коли таке автоматизоване групування даних може бути надзвичайно корисним. Наприклад, розглянемо випадок створення рекламної кампанії в Інтернеті для абсолютно нового асортименту продукції, що випускається на ринок. Набагато кращим підходом було б розділити населення на кластери людей, які мають спільні характеристики та інтереси, показуючи індивідуальну рекламу для кожної групи. K-середні - це алгоритм, який знаходить ці групи у великих масивах даних, де це неможливо зробити вручну.

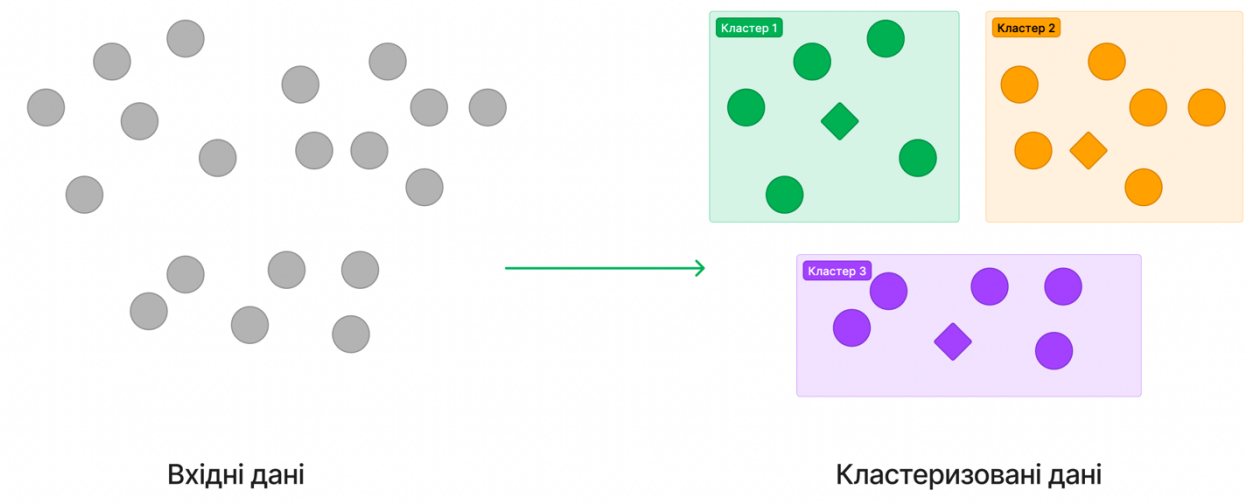


Рисунок 3.4.1 – Вигляд даних до та після кластеризації

Логіка, що лежить в основі алгоритму, не є складною. Для початку вибирається значення k (кількість кластерів) і випадковим чином вибирається початковий центроїд (координати центру) для кожного кластера. Потім застосовується двоетапний процес:

- Крок присвоєння - присвоєння кожному спостереженню його найближчий центр.

- Крок оновлення - оновлення центроїдів як центрів відповідних спостережень.

Ці два кроки повторюються знову і знову до тих пір, поки в кластерах не відбудуться подальші зміни. На цьому етапі алгоритм збігається, і є можливість отримати остаточні кластеризації.

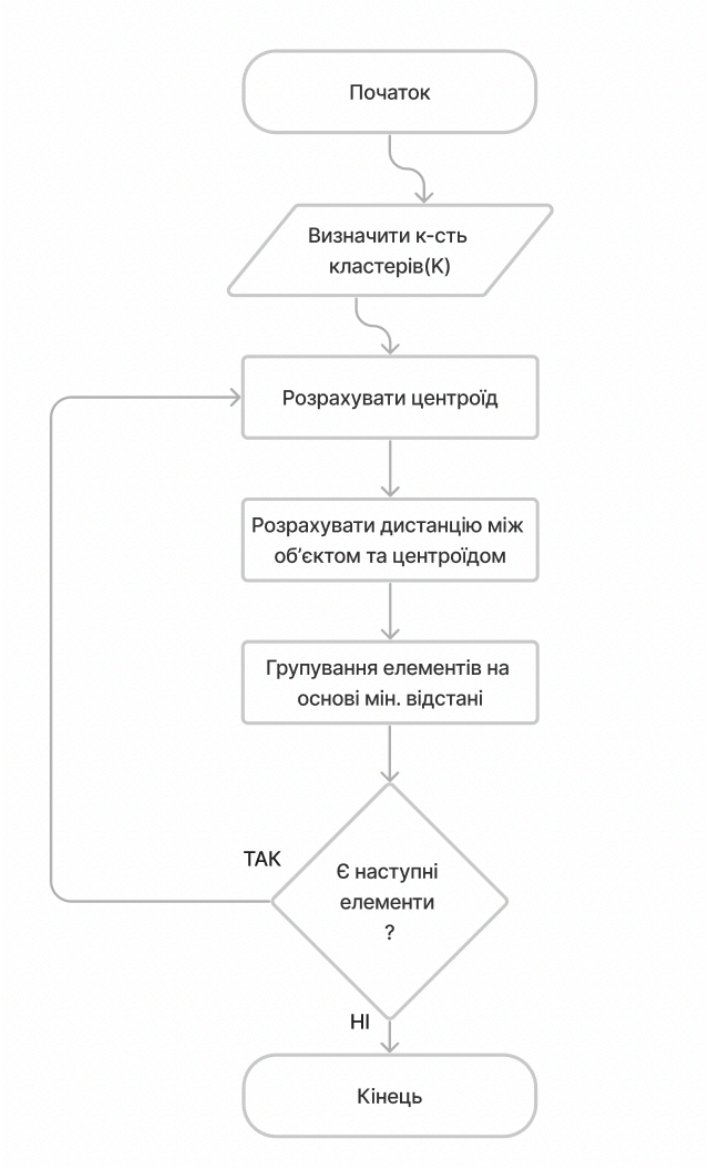


Рисунок 3.4.1 – Алгоритм роботи «к-середніх»

Кластеризація за методом K-середніх має на меті розбити n об'єктів на k кластерів, в яких кожен об'єкт належить до кластера з найближчим середнім значенням. Цей метод дає рівно k різних кластерів з найбільшою можливою

різницею. Найкраща кількість кластерів k, що призводить до найбільшого розділення (відстані), не відома апріорі і повинна бути обчислена на основі даних.

Метою кластеризації за методом K-середніх є мінімізація загальної внутрішньокластерної дисперсії, або функції квадрата помилки(3.4.1):

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\| \quad (3.4.1)$$

Де J – цільова функція, k – число кластерів, n - число випадків, x – випадок i, c - центроїд кластеру j, вираз $\left\| x_i^{(j)} - c_j \right\|$ – функція дистанції (Евклідова відстань між точкою даних xi та центроїдом кластера cj).

K-mean є відносно ефективним методом. Однак, необхідно заздалегідь визначити кількість кластерів, а остаточні результати є чутливими до ініціалізації і часто закінчуються на локальному оптимумі. На жаль, не існує глобального теоретичного методу для знаходження оптимальної кількості кластерів. Практичний підхід полягає в порівнянні результатів декількох прогонів з різними k і виборі найкращого на основі заздалегідь визначеного критерію. Загалом, велике k, ймовірно, зменшує похибку, але збільшує ризик надмірного припасування.

Рівняння знаходження дистанції виглядає наступним чином(3.4.2) (3.4.3) (3.4.4):

$$D1 = |x_i - c_1| \quad (3.4.2)$$

$$D2 = |x_i - c_2| \quad (3.4.3)$$

$$Dn = |x_i - c_n| \quad (3.4.4)$$

K-mean можна застосовувати до набору даних з меншою кількістю вимірів, числових і безперервних даних. Він підходить для сценаріїв, коли потрібно згрупувати випадково розподілені точки даних. Ось деякі з випадків використання, де можна застосувати K-середніх:

Сегментація клієнтів - задоволення потреб клієнтів є відправною точкою маркетингу взаємовідносин, і його можна покращити, розуміючи, що всі клієнти не однакові, і однакові пропозиції можуть не працювати для всіх. Сегментування клієнтів на основі їхніх потреб та поведінки може допомогти компаніям краще продавати свою продукцію правильним клієнтам. Наприклад, телекомунікаційні компанії мають велику кількість користувачів, і, використовуючи сегментацію ринку або клієнтів, компанії можуть персоналізувати кампанії та заохочення тощо.

Виявлення шахрайства - постійний розвиток Інтернету та онлайн-сервісів викликає занепокоєння з приводу безпеки. Облік цих загроз безпеці або шахрайських дій, наприклад, вхід в обліковий запис Instagram з незвичайного міста або приховування будь-яких фінансових порушень, є поширеним явищем у сьогоденні.

Використовуючи такі методи, як кластеризація K-середніх, можна легко виявити закономірності будь-якої незвичайної діяльності. Виявлення відхилення буде означати, що відбулася подія шахрайства.

Класифікація документів - к-середні відомі своєю ефективністю у випадку великих наборів даних, саме тому вони є одним з найкращих варіантів для класифікації документів. Кластеризація документів за кількома категоріями на основі тематики, змісту та тегів, якщо такі є. Документи конвертуються у векторний формат. Потім використовується частота термінів для виявлення загальних термінів, і на основі цього можна виявити схожість у групах документів.

Через ці обмеження керованих алгоритмів, потрібно використовувати некеровані алгоритми, такі як кластеризація за методом К-середнього, де буде можливість легко порівняти георозмаїття шляхом кластеризації даних.

Сегментація зображення - використовуючи K-середнє, ми можемо знайти закономірності в пікселях зображення, які дозволять пришвидшити обробку і зробити її більш ефективною. Після обчислення різниці між кожним пікселем

зображення та центроїдом, вона зіставляється з найближчим кластером. У кінцевому результаті кластери матимуть схожі пікселі, згруповані разом.

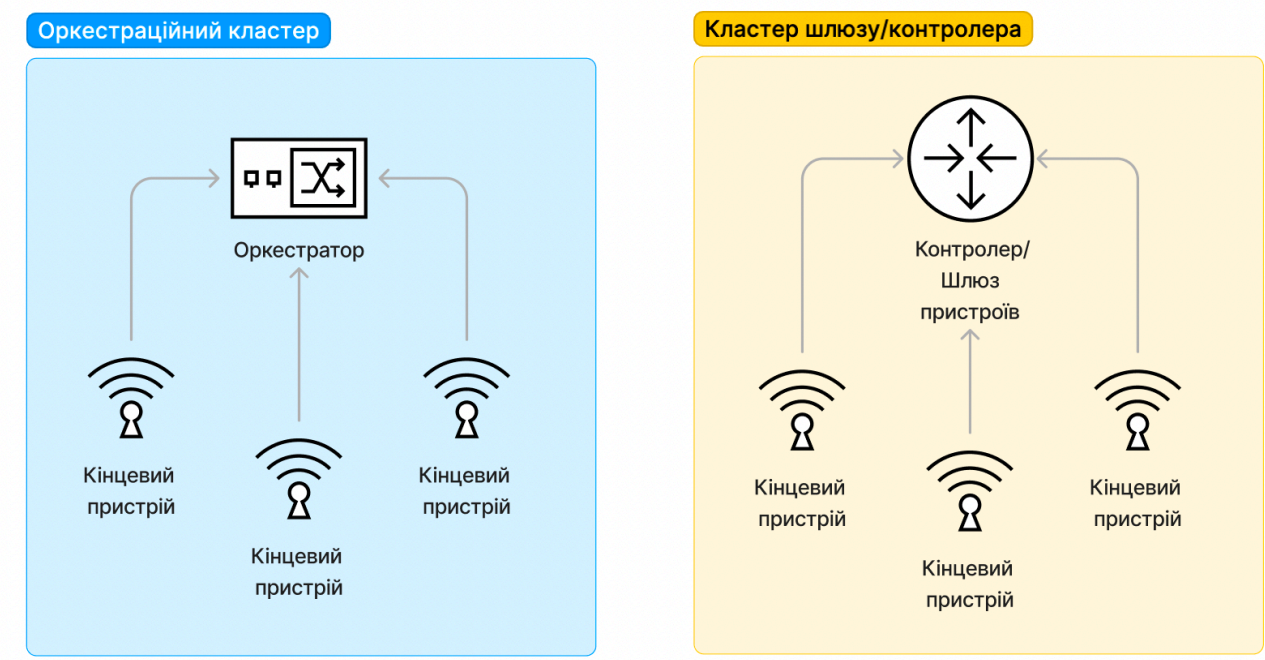


Рисунок 3.4.2 – Фрагмент системи, що має брати участь в кластеризації

Переваги алгоритму кластеризації K-mean:

- Відносно простий у розумінні та реалізації.
- Масштабованість до великих наборів даних.
- Краща вартість обчислень ніж в аналогів.
- Легкий призначень та позицій центроїдів

Орієнтуючись на вказаний метод кластеризації було вирішено таким чином створити реалізацію, що має на меті згрупувати в кластери пристрої, що за розрахунками будуть належати одній і тій ж групі та бути під’єднаними до оркестратора, чи шлюзу , що будуть з’єднувати пристрої в одну мережу.

Для цього умовно було створено площину, на якій знаходяться кінцеві пристрої та оркестратори/шлюзи для пристроїв, їх місце знаходження визначено

точкою, з координатами x та y. В цьому випадку розглядається вузол: периферійний пристрій – окрестратор або шлюз для пристроїв

В такому випадку, аби підключитися до потрібного первинного пункту обробки запитів з периферійних пристроїв можливо використовувати кластеризацію, аби кожен оркестратор міг визначати для себе оптимальну групу девайсів, які можуть до нього приєднатися.

Аби створити таку площину оркестратору знадобляться координати на яких знаходятся пристрої. Існують різні способи знаходження геолокації для пристроїв, деякі з них використовуюсть вбудовані сервіси, що працють в тандемі з супуниковим або з сотовим підєднанням. Найвідоміші можливості ідентифікації геолокації IoT є GPS та тріангуляція за допомогою веж стільникового зв'язку

Найбільш знайома технологія геолокації IoT, відома всім з телефонів і автомобілів, - це Глобальна система позиціонування (GPS). GPS використовує масив з 31 супутника, розташованих на висоті 22 000 кілометрів над Землею на геосинхронних орбітах, що означає, що вони з'являються в одній і тій же позиції в небі через період в одну зоряну добу.

Кожен супутник постійно транслює своє місцезнаходження і час з точністю до пікосекунди. Щоб знайти місцезнаходження за допомогою багатосторонньої синхронізації/часу прольоту, пристрій обчислює час, необхідний для проходження радіосигналу від супутників до власного місця розташування на передбачуваній відстані. Потім він вирішує задачу перетину наближених діапазонів, щоб знайти своє місцезнаходження і висоту. Для цього потрібен точний годинник на приймачі, і деякі розрахунки для визначення місцезнаходження - похідні від відстаней, що випливають з часових зсувів супутників. Пристрій Інтернету речей, що використовує GPS, одночасно фіксує час і місцезнаходження з трьох або більше супутників GPS, а потім використовує різницю для багатостороннього визначення місцезнаходження.

Супутники працюють на частотах 1,5 або 1,2 ГГц - частотах, які не дуже добре проникають через стіни або дахи - і супутники знаходяться дуже далеко, що обмежує потужність сигналу. Ось чому ваш GPS-приймач на вашому автомобілі, здається, відмовляє в гаражі або на міських вулицях з високими будівлями - він не може "бачити" кілька супутників в цих місцях, і, ймовірно, вгадує місцезнаходження на основі вже зібраних даних.

GPS добре працює з легковими автомобілями, вантажівками та човнами, але безперервна робота GPS є відносно енергоємною. Ось чому ввімкнення "інформації про місцезнаходження" в налаштуваннях телефону так швидко розряджає батарею

Стільникові технології для геолокації: говорячи про телефони, стільникові технології також можуть бути використані для геолокації. Мобільні телефони можуть отримувати відносне місцезнаходження за допомогою трилатерації - використання сили радіосигналів між пристроєм і найближчими вежами стільникового зв'язку для обчислення відносного положення між ними. Кожна з веж стільникового зв'язку має відоме місце розташування, яке може бути використане для подальшого обчислення абсолютного місцезнаходження пристрою, виходячи з позицій веж стільникового зв'язку.

Трилатерація стільникових веж є точною з точністю до декількох сотень футів і добре працює в приміщенні, але вона не дуже добре визначає висоту над рівнем моря. Однак, якщо у користувача є пристрій стільникового зв'язку, можна передавати багато інформації на додаток до місцезнаходження, що може допомогти уточнити його місцезнаходження. Недоліками є те, що він також досить енергоємний і вимагає, щоб пристрій мав постійний контракт з оператором стільникового зв'язку, який коштує грошей, поки пристрій використовується.

Найкраще підходить для відстеження пристроїв високої вартості, таких як медичні вантажі, будівельне обладнання та інші, які можуть підтримувати

поточні платежі за послуги, а також великі батареї, необхідні для тривалої роботи.

Для геолокації в межах будинку та прибудинкових територій можна використовувати службу визначення місцезнаходження WiFi в реальному часі (WiFi RTLS). Виявляється, що трилатерація між вишками стільникового зв'язку - не єдиний трюк, якого навчили нас мобільні телефони в сфері геолокації. WiFi Real Time Location Service (WiFi RTLS) - це існуюча послуга, що надається Google та іншими компаніями, яка зберігає оновлену карту WiFi-роутерів і розраховує місцезнаходження телефону на основі його звіту про рівень сигналу WiFi-роутерів, які він може бачити.[11]

Принцип роботи: Маршрутизаторам WiFi присвоюється відносне місце розташування, яке потім означає фізичну геолокацію в базі даних. Дані про рівень сигналу від декількох джерел, які бачить пристрій, потім можуть бути використані для розрахунку діапазонів для визначення оціночного положення. Ця сукупність декількох джерел може дати досить точне місцезнаходження - аж до декількох футів у приміщенні. Ось чому ваш телефон просить вас "увімкнути WiFi для кращої точності визначення місцезнаходження", щоб працювати разом з GPS і вежами стільникового зв'язку.

WiFi-роутери працюють в діапазоні 2,4 ГГц - 5 ГГц. Нижня частина частоти дещо проходить крізь стіни, але у великій будівлі потрібні підсилювачі. Світлодіодні лампочки, які змінюють колір, системи сигналізації для мобільних об'єктів, телефони, розумні контролери спринклерів та інша побутова техніка - ось деякі з поширених пристроїв IoT, які можуть використовувати WiFi RTLS.

Для роботи в приміщенні можна використовувати Bluetooth-маяки та UWB. Bluetooth-маяки - це ще один тип системи відносного місцезнаходження. Вони "пінгують" сусідні пристрої для визначення близькості (знову трилатерація) і можуть викликати відповідну реакцію, якщо пристрій знаходиться досить близько. Прикладом може бути сцена у фільмі "Особлива думка", в якій персонаж прогулюється по торговому центру, і конкретні

магазини налаштовують свою рекламу, коли він проходить повз. Bluetooth, як і WiFi, використовує частоти 2,4 ГГц.

Пристрій Tile, який може відстежувати ваші ключі, телефон і гаманець, використовує Bluetooth-маячок. Подібні системи використовуються для відстеження пацієнтів і обладнання в лікарнях. Вони можуть бути об'єднані в мережу, щоб забезпечити абсолютне місцезнаходження на невеликій території, наприклад, в будинку, але сигнали Bluetooth, як правило, слабкіші і не мають зовнішньої інфраструктури для застосування на великих територіях / в сільській місцевості.

Якщо потрібне надточне місцезнаходження в приміщенні, спробуйте надширокосмуговий діапазон (UWB). В даний час є лідером у відстеженні об'єктів з високою роздільною здатністю в приміщенні, такі варіанти, як Decawave, можуть дати вам місцезнаходження до чверті дюйма, використовуючи методи мультилатерації. Він призначений для відстеження активів високої вартості в невеликих приміщеннях, таких як медичне обладнання в операційній. Для цього потрібно кілька маяків на кімнату, а сигнал легко блокується стінами.

Для використання в дорозі чи в полі бажано використовувати широкосмугові мережі великого радіусу дії (LoRaWAN)[11]

Якщо потрібне відстеження об'єктів на великій відстані з низьким енергоспоживанням для об'єктів, яким не потрібно передавати багато даних, крім їх місцезнаходження - пристроїв, для яких стільниковий зв'язок або GPS були б надмірністю - розгляньте можливість використання широкосмугових мереж дальнього радіусу дії (LoRaWAN або LoRa, скорочено - LoRa).

LoRa визначає місцезнаходження за допомогою трилатерації, подібно до інших радіометодів. Вона має точність до кількох десятків футів .

Визначившись з координатами, на яких знаходяться точки можна проводити кластеризацію на їх основі. Аби показати як працює кластеризація пристроїв наочно було вирішено відтворити всі пристрої на координатній площині, де точками одного кольору виступають пристрої, що знаходяться в

одній групі(кластері). А точки червоного кольору – оркестратори, що мають установити з’єднання з цими пристроями

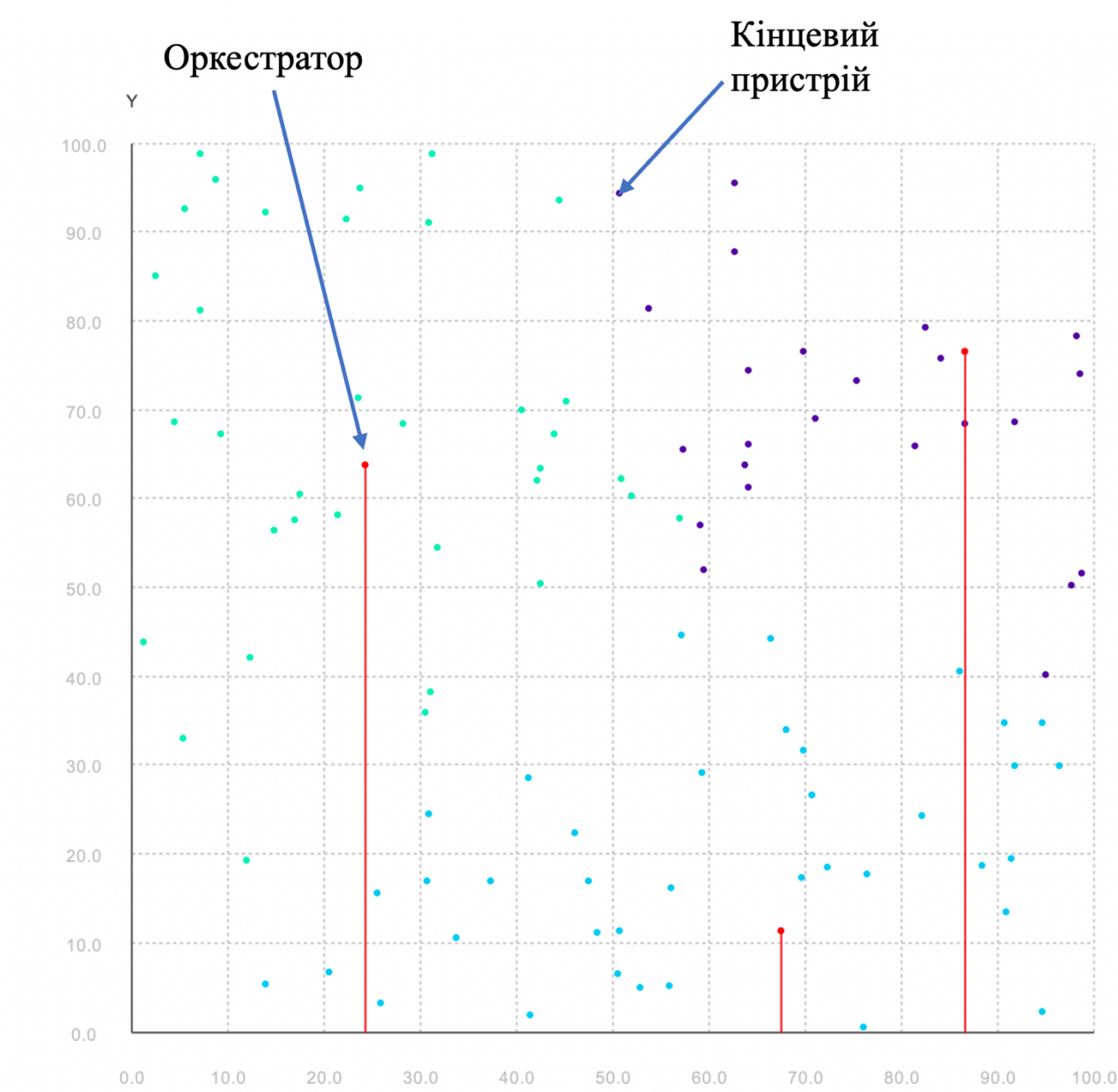


Рисунок 3.4.3. – Результат кластеризації кінцевих пристроїв в мережі.

Вхідними параметрами для моделі є кількість кластерів та координати точок всіх пристроїв, опціонально ще можуть бути координати оркестраторів, до яких будуть підключатися пристрої, в майбутньому логічним розвитком такої

системи буде динамічна конфігурація на основі параметрів кінцевих пристроїв, що можуть відігравати велику роль швидкості передачі даних.

Вирішальну роль у проектуванні таких систем буде відігравати головна ціль доставлення повідомлень/завдань між елементами системи, тому що деякі системи можуть бути унікальними в обслуговуванні, проектуванні, експлуатації та подальшої підтримки. Таким чином на основі розробленої методики можна поставити базу для майбутньої системи, яку згодом можна модифікувати, на кожному потрібному рівні у відповідності до поставлених вимог. Оскільки у проектувальників вже буде база, процес розробки системи буде пришвидшений за рахунок прискорення етапу проектування.

Шляхом розподілу кінцевих пристроїв на групи, які під свою відповідальність беруть оркестраційні елементи мережі буде досягнено балансування мережі, за рахунок обчислень мінімальних дистанцій оркестратор – пристрій. Також таким чином буде досягнена стабілізація трафіку, оскільки пристрої будуть находитись на оптимальній відстані до шлюзу, щоб входити в групу, бо як тільки їх відстань буде перевищувати максимальної відрахованої для елементу групи, вони одразу будуть назначені до іншої групи.

ВИСНОВКИ

В ході магістерської роботи було досліджено процеси обробки даних, їх проблематику та особливості використання самостійних оркестраторів, що сполучають між собою пристрої різного рівня та виду обчислень, було розроблено методику, що мала на меті удосконалити процес оркестрації в системі починаючи з низького рівня(рівня первинної обробки завдань) до рівня отримання пакетів даних, готових до цільового використання, таким чином, підвищивши рівень ефективності оброки даних в системі.

В ході роботи було виконано наступне:

- Проаналізовано існуючі методики проектування сучасних систем обробки інформації
- Досліджено алгоритми оркестрації та транспортування даних.
- Розглянуто дослідження з використанням захищеної методики передачі даних між компонентами системи та реалізацій протоколів захисту
- Розроблено модель системи з використанням окестраторів, для проміжної обробки даних з кінцевих пристроїв до шлюзу додатку та протоколом екстреної передачі даних.
- Проведено емулювання роботи оркестратора в високонавантажених системах з виставленою конфігурацією ключових характеристик кінцевих пристроїв та мережі, в якій вони знаходяться.
- Проведено моделювання мережі оркестратора завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CGAT core docs/ Writing a workflow [Електронний ресурс] - 2022 https://cgat-core.readthedocs.io/en/latest/defining_workflow/Writing_workflow.html#building-a-pipeline

2. ZenML/ Core Concepts [Електронний ресурс] - 2022 – Режим доступу https://docs.zenml.io/getting-started/core-concepts

3. Kubeflow team / An overview of Kubeflow's architecture - [Електронний ресурс] – Режим доступу: https://www.kubeflow.org/docs/started/architecture/.

4. The tech platform/ Microservices: Orchestration vs. Choreography - 2022 - [Електронний ресурс] – Режим доступу до ресурсу: https://www.thetechplatform.com/post/microservices-orchestration-vs-choreography

5. Aporv Tomar/ Microservice Orchestration vs. Choreography - [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/geekculture/microservices-orchestration-vs-choreography-technology-5dbe612cf7e9

6. Anas Nadeem Muhammad Zubair Malik /A case for microservices orchestration using workflow engines, 2022 – 1-3c.

7. Mageto Stephen N , N.V.Balaji/Usage of Trust Mechanisms in Cloud Computing Review, 2022 – 9-11c.

8. Zhao Ziming, Liu Fang, Cai Zhiping, Xiao Nong. Edge Computing: Platforms, Applications and Challenges[J]/Journal of Computer Research and Development- 2018- 55(2): 327-337c

9. The tutorial point/ Fuzzy logic – Inference system – [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/fuzzy_logic/fuzzy_logic_inference_system.htm

10. Alan Jeffales / K-means: A Complete Introduction -2019 - [Електронний ресурс], https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c

11. Delve/ IoT Geolocation: How To Choose the Best Technology for Your Device: https://www.delve.com/insights/iot-geolocation-how-to-choose-the-best-technology-for-your-device

ДОДАТКИ

Додаток А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

МАГІСТЕРСЬКА РОБОТА
«ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОРКЕСТРАЦІЇ ЗАВДАНЬ В СЕРЕДОВИЩАХ З ПЕРИФЕРІЙНИМИ ОБЧИСЛЕННЯМИ»

Виконав: студент групи ПДМ – 61, Конішевський Владислав Ігорович

Керівник: к.т.н., доцент, Негоденко Олена Василівна

Київ - 2023

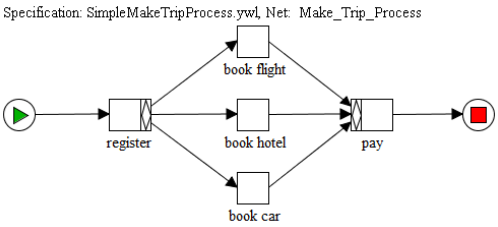
МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ 2

Мета роботи: Підвищення ефективності оркестрації завдань в середовищах з периферійними обчисленнями

Об'єкт дослідження: : Процес наповнення, передачі, обробки завдань в системах з периферійними обчисленнями.

Предмет дослідження: : моделі та методики підходів щодо окрестрації та її внутрішніх механізмів

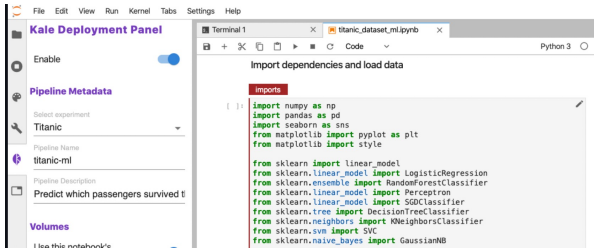
АНАЛІЗ ІСНУЮЧИХ ІТ-РІШЕНЬ ТА ЇХ МОДЕЛЕЙ 3



YAWL –мова потоку робіт, заснована на патернах потоків робіт та мережах Петрі

CGAT-core– система управління робочими потоками, яка дозволяє будувати масштабовані конвеєри аналізу даних

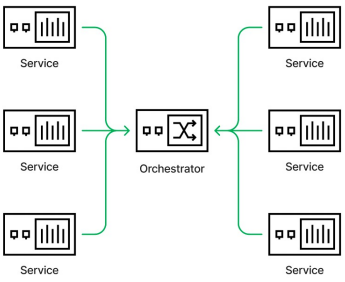
ZenML - це розширюваний фреймворк MLOps для створення портативних, готових до виробництва конвеєрів MLOps.



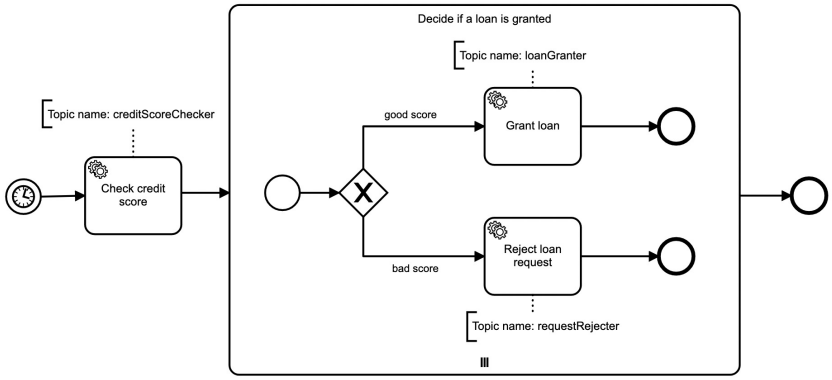
Kubeflow - платформа для організації складних робочих процесів на вершині Kubernetes.

Оркестраційний елемент системи

4



Загальноприйнята архітектура системи з оркестраційним елементом



Приклад розподілу завдань системи, та делегації її та результатів до наступних ланок

Нечітка логіка в процесі обробки завдань

5

Різниця між булевою логікою та нечіткою

Метод фазифікації

$$\widetilde{A} = \mu_1 Q(x_1) + \mu_2 Q(x_2) + \ldots + \mu_n Q(x_n)$$

де, Q(xi) – ядро нечіткості,
μi – визначена константа для множини

Метод дефазифікації

Для скінечних областей:

$$\bar{x} = \frac{\sum x_i A_i}{\sum A_i}$$

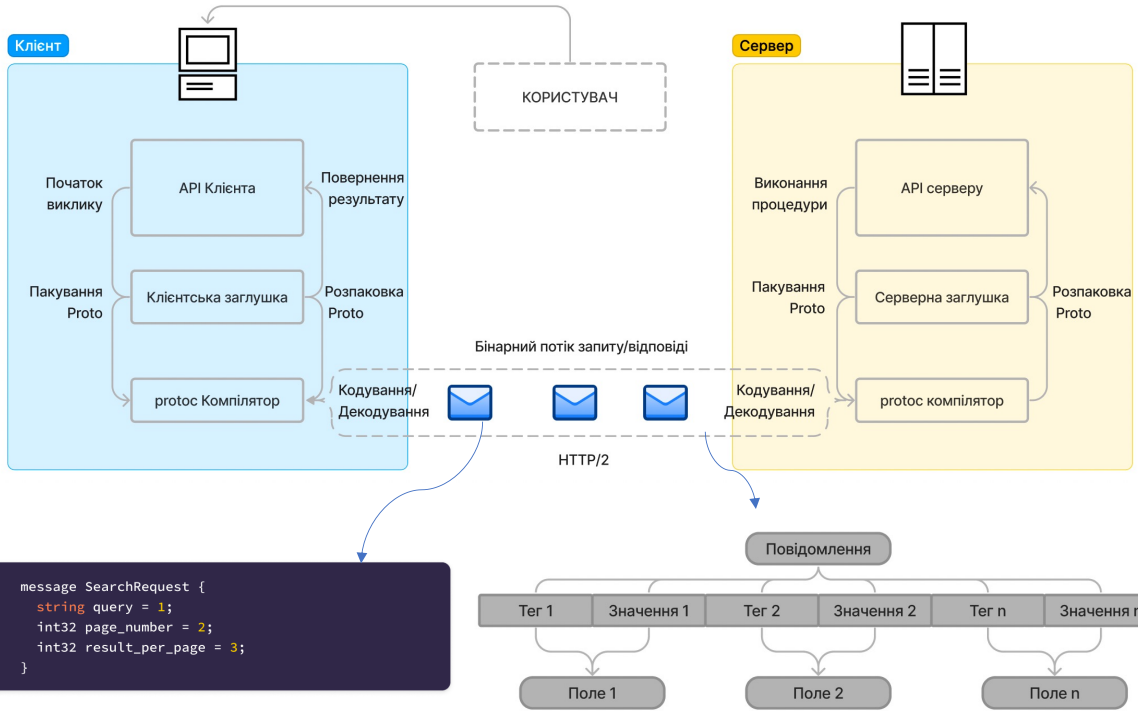
Для безперервних областей

$$\bar{x} = \frac{\int x dA}{\int dA}$$

де, x – відстань до регіону,
A – площа регіону

Рівень запиту. Внутрішня модель роботи gRPC

6




```
message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

Формат повідомлень (незакодований)

Будова потоку повідомлень

Модель розгортання власної мережі оркестратора

7

Схема алгоритму кластеризації

Мережа окестратора – множина клітин D, розміщених у вигляді решітки, де виділені сектора, які містят клітини $d_{(i,j)}$:

$$d_{(i,j)} \in D.$$

Визначення секторів мережі оркестратора за алгоритмом К-середніх

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|, \text{де}$$

J – цільова функція, k – число кластерів, n - число випадків, x – випадок i, c -центроїд кластеру j, вираз $\left\| x_i^{(j)} - c_j \right\|$ – функція дистанції (Евклідова відстань між точкою даних xi та центроїдом кластера cj).

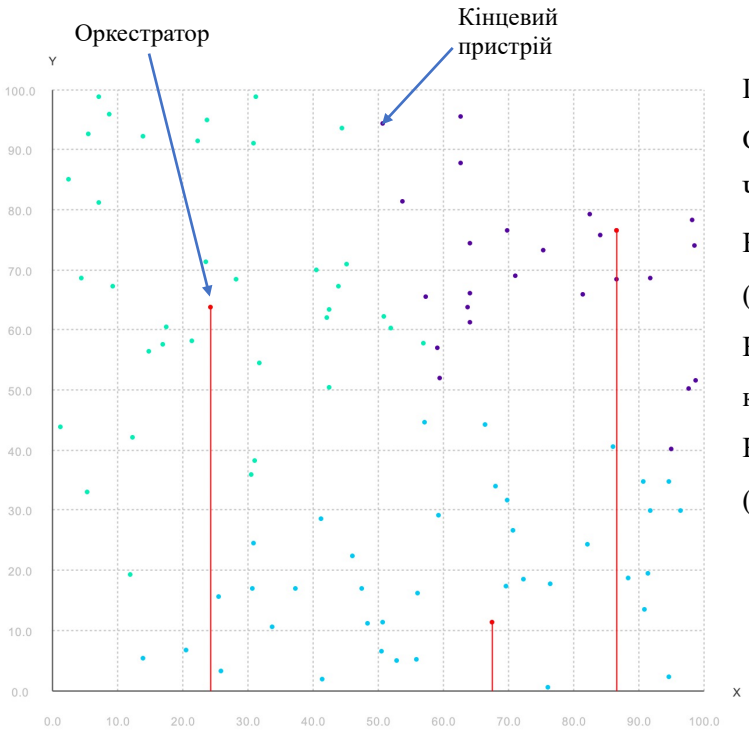
Визначення дистанції між клітинами, в яких знаходяться обчислювальні прилади

$$D1 = |x_i - c_1|$$
$$D2 = |x_i - c_2|$$
$$Dn = |x_i - c_n|$$

Рівень обробки завдань. Результати емуляції потоку завдань 8

Алгоритм обчислень	Нечітка логіка	Циклічне розподілення(Round Robin)
% виконаних завдань	98,3195	81.3933
Запущені, але не виконані ізза високої затримки (%)	0	18.6067
К-сть незавершених завдань	92	1055
Енергоспоживання(загальне)	89.4506 Вт/г	81.8768 Вт/г
Використання мережі (Загальне, с)	69.3818 секунд	92.6843 секунд
Середня швидкість передачі даних	451.557 Мбіт/с	348.088 Мбіт/с
Трафік	3878 Мбайт	3825.25 Мбайт

Результати моделювання мережі оркестратора 9



Приклад моделювання:

Обрана к-сть кластерів(**К**) – 3

Число пристроїв (**x**) – 100

К-сть пристроїв кластеру 1 (зелений колір**, n1**) – 36 пристроїв

К-сть пристроїв кластеру 2 (Синій колір, **n2**) – 40

К-сть пристроїв кластеру 3 (Фіолетовий колір, **n3**) -24

ВИСНОВКИ

1. Проаналізовано існуючі методики проектуваня сучасних систем обробки інформації
2. Досліджено алгоритми оркестрації та транспортування даних.
3. Розглянуто дослідження з використанням захищеної методики передачі даних між компонентами системи та реалізацій протоколів захисту
4. Розроблено модель системи з використанням окестраторів, для проміжної обробки даних з кінцевих пристроїв до шлюзу додатку та протоколом екстреної передачі даних.
5. Проведено емулювання роботи оркестратора в високонавантажених системах з виставленою конфігурацією ключових характеристик кінцевих пристроїв та мережі, в якій вони знаходяться.
6. Проведено моделювання мережі оркестратора завдань.

ДЯКУЮ ЗА УВАГУ!