Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод організації еластичних обчислень в автономній системі

(тема)

Виконав

студент II курсу, групи СПм-22-2

Настиченко Т.А.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Кучук Н.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Настиченко Тетяна Андріївна
(прізвище, ім’я, по батькові)

1. Тема роботи Метод організації еластичних обчислень в автономній системі

затверджена наказом по університету від “ 06 ” листопада 2023 р. № 1299 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи 1) Файли формату *.py, *.pb, *.txt, які містять послідовність виконуваних функцій. 2) Операційна система –Windows 10, 11.
3) Модулі asyncio та threading.

4. Перелік питань, що потрібно опрацювати у роботі
1) Аналітичний огляд.
2) Розробка системи для еластичних обчислень.
3) Програмна реалізація системи.
4) Тестування розробленої системи для еластичних обчислень.
5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 16

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	07.11.23-13.11.23	
2	Вибір та обгрунтування методики дослідження	14.11.23-20.11.23	
3	Вибір інструментальних засобів	21.11.23-23.11.23	
4	Розробка системи для еластичних обчислень.	24.11.23-06.12.23	
5	Тестування розробленої системи	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.23-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кучук Н.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 78 с., 22 рис., 3 табл., 2 дод., 15 джерел.

ХМАРНІ ОБЧИСЛЕННЯ, ЕЛАСТИЧНІ ОБЧИСЛЕННЯ, АСИХРОННІСТЬ, БАГАТОПОТОЧНІСТЬ, МЕРЕЖІ ПЕТРІ.

Метою кваліфікаційної роботи є розробка моделі для організації еластичних обчислень.

У ході виконання кваліфікаційної роботи було проведено дослідження можливості організації еластичних обчислень у системі з декількома процесорами. У ході роботи використано напрацювання у сфері технології інтернету речей. Зокрема, було розглянуто можливості використання хмарних обчислень та запропоновано використання моделі туманних обчислень для забезпечення автономності, а також задля досягнення передбачуваної затримки передачі.

ABSTRACT

Master’s thesis: 78 pages, 22 figures, 3 tables, 2 appendices, 15 sources.

CLOUD COMPUTING, ELASTIC COMPUTING, ASYNCHRONY, MULTITHREADING, PETRI NET

The goal of the work is to develop a model for the organization of elastic calculations.

The object of the work is the process of developing a model for the organization of elastic calculations.

The purpose of the work is to develop a model for the organization of elastic calculations and its implementation. The object of the work is the process of developing a model for the organization of elastic calculations and its implementation The possibility of organizing elastic calculations in a system with several processors was studied. In the course of the work, the development in the field of Internet of Things technology was used. In particular, the possibilities of using cloud computing were considered and the use of a fog computing model was proposed to ensure autonomy, as well as to achieve a predictable delay in the transmission of tasks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

ВМ – віртуальна машина

IaaS – Infrastructure as a service

IoT – Інтернет речей

SaaS – Software as a service

IDE - Integrated development environment

PaaS – Platform as a service

NAT – Network Address Translation

ВСТУП

Сьогодні інформація є одним з найважливіших ресурсів не тільки для підприємств, які конкурують за вплив над ринком, а навіть для цілих держав. При цьому обробка та зберігання цієї інформації потребує значного обсягу обчислювальної потужності. Тому вимоги та вартість її обробки зростають із разючою швидкістю. Великі обсяги потужностей потрібні в сферах освіти, науки, медицини, навіть у повсякденному житті. Тому все гостріше постає питання про необхідність створення економічних та ефективних систем обробки даних.

Розробка будь-якого програмного забезпечення на даний час проходитиме в умовах чітких обмежень за рахунок бюджету компанії-замовника, що сильно обмежує можливості розгортання інфраструктури для вдалої роботи. Коли замовник замовляє розробку продукту, компанія-виконавець повинна чітко зрозуміти обсяг навантаження системи та фінансові можливості клієнта, оскільки таким чином можна зрозуміти в яких межах можливо проводити розробку продукту і чи взагалі можливо розробити додаток, який буде відповідати умовам доступності, відмовостійкості та ефективності використання ресурсів. Ось тут і стоїть питання використання туманних обчислень, чи релевантно закуповувати самому обладнання, наймати людей, чи краще використати для цього хмару?

Частіше, програмісти при створенні додатку вибирають другий варіант. Такий варіант не підходить тоді, коли потрібна велика швидкість обчислень, або пропускна здатність системи недостатня для відправки даних. Але, в першому варіанті, самого лише обладнання недостатньо. Наприклад, є декілька систем, кожна з яких займається вирішенням свого завдання. У такому разі може виникнути ситуація, коли потужності бракує для виконання необхідних обчислень вчасно. Вирішенням проблеми може бути делегування завдання іншій системі, який має у своєму розпорядженні більше обчислювальних потужностей, яких вистачить для проведення всіх потрібних

обчислень у термін, проте навіть тоді не завжди потужностей вистачатиме, а також вона вже може бути зайнята виконанням будь-якого іншого важливого завдання. Чи можна вирішити проблему, використовуючи лише ті ресурси, які є? Так, і рішенням у подібній ситуації може стати об'єднання кількох систем в одну потужнішу обчислювальну мережу.

У зв'язку з цим виникає нова проблема: як відбуватиметься утворення обчислювальної мережі? Найочевиднішим рішенням буде організація обчислювальної мережі в ручному режимі, тобто якась людина повинна буде включати в разі потреби обрані системи в єдину обчислювальну мережу. Однак це створює лише більше проблем: буде потрібний навчений персонал, якому доведеться витрачати час на об'єднання, замість вирішування інших завдань. Також можна написати ПЗ, яке в автоматичному режимі об'єднуватиме системи, коли необхідно, проте в цьому випадку виникає проблема нестачі ресурсів через високу нагрузку в одній або декількох систем, що унеможливить їх об'єднання.

Звідси слід знайти спосіб, за допомогою якого відбуватиметься автономне об'єднання кількох систем в єдину обчислювальну мережу, для подальшого виконання будь-якої задачі. При цьому варто враховувати те, що така мережа має вміти змінюватися з часом, тобто до неї повинні вміти підключатися нові процесори, а також відключатися старі. Така концепція, коли пул ресурсів не статичний, а може й змінюватися з часом, називається еластичними обчисленнями. На цю концепцію добре лягає така технологія як інтернет речей, де теж постійно з'являються нові пристрої та відключаються старі.

У кваліфікаційній роботі необхідно визначити спосіб, яким можна об'єднати множину систем в єдину мережу, яка може проводити паралельні обчислення на всіх системах, що входять до цієї мережі. При цьому системи можуть додаватися в обчислювальну мережу, так і виходити з неї, т.к. для підтримки всіх систем в активному стані потрібно значна кількість енергії та ресурсів. Тобто потрібно організувати еластичні обчислення - обчислення у

яких ресурси можуть додаватися та звільнятися за потребою.

Проаналізувати варіанти реалізації із використанням інтернету речей, математичний апарат мереж Петрі для побудови моделі. Реалізувати отримані знання для створення моделі еластичних обчислень. Написати програму для моделювання еластичних обчислень. Зробити висновок щодо ефективності вивчених методів на основі отриманих результатів моделювання.

1 ОГЛЯД СУЧАСНОГО СТАНУ З ВИКОРИСТАННЯ ЕЛАСТИЧНИХ ОБЧИСЛЕНЬ В ТУМАННИХ ТЕХНОЛОГІЯХ

1.1 Інтернет речей

Інтернет речей – концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами в автоматичному режимі, за допомогою використання стандартних протоколів зв'язку. Окрім давачів, мережа може мати виконавчі пристрої, вбудовані у фізичні об'єкти і пов'язані між собою через дротові чи бездротові мережі. Ці взаємопов'язані пристрої мають можливість зчитування та приведення в дію, функцію програмування та ідентифікації, а також дозволяють виключити необхідність участі людини, за рахунок використання інтелектуальних інтерфейсів. Типичная система IoT работает посредством сбора и обмена данными в режиме реального времени.

Система IoT складається з трьох компонентів: смарт-пристрій, додаток IoT та графічний інтерфейс користувача. Розглянемо кожен компонент быльш детально.

Смарт-пристрій – це пристрій, такий як телевізор, камера відеоспостереження або тренажер, якому було надано обчислювальні можливості. Такий пристрій збирає дані зі свого середовища, введення користувача або шаблонів використання і передає дані через Інтернет до програми IoT і з нього.

Додаток IoT – це набір сервісів та ПЗ, які поєднують дані, отримані від різних пристроїв IoT. Така програма використовує технологію машинного навчання або штучного інтелекту для аналізу цих даних та прийняття обґрунтованих рішень.

Ці рішення передаються назад на пристрій IoT, а потім IoT інтелектуально реагує на вхідні дані.

Графічний інтерфейс користувача Пристроєм IoT або парком пристроїв можна керувати через графічний інтерфейс користувача. Загальні приклади включають мобільний додаток або веб-сайт, які можна використовувати для реєстрації та керування смарт-пристроями.

Можна скористатися напрацюваннями з цієї галузі, для вирішення поставленого завдання, а також для прогнозування майбутніх проблем, які можуть виникнути зі збільшенням кількості систем в мережі.

1.2 Туманні обчислення

За останнє десятиліття IT-індустрія зазнала серйозних змін. Найбільш кардинальним рішенням стала поява хмарних технологій у сфері зберігання даних, обчислень, мережевої взаємодії. Але у сучасних реаліях навіть його можливостей уже замало.

Як показала практика, хмарні технології мають низку обмежень. І вони негативно впливають на зручність роботи:

- необхідність у наявності інтернет-з'єднання;
- затримка
- проблема нестачі смуги пропускання
- повільна реакція та проблеми масштабування

Сьогодні із найбільшими обмеженнями зіткнувся інтернет речей. Вирішити такі проблеми мають туманні обчислення. Це передова концепція у хмарній архітектурі, що передбачає обробку даних на периферійних пристроях мережі – ПК, смартфонах, персональних гаджетах та ін., виводячи їх із хмари.

Термін туманні обчислення (або розмивання) був придуманий CISCO в 2014 році, тому він є новим для більшості людей. Туманні та хмарні обчислення взаємопов'язані між собою.

Визначення може звучати так: туманне обчислення – це розширення хмарних обчислень, що складається з кількох граничних вузлів

безпосередньо підключених до фізичних пристроїв. Такі вузли фізично набагато ближчі до пристроїв порівняно з централізованими центрами обробки даних, тому вони здатні забезпечувати миттєві з'єднання.

Значна обчислювальна потужність периферійних вузлів дозволяє самостійно виконувати обчислення великого обсягу даних, не відправляючи їх у віддалений сервер.

Модель туманних обчислень показана на рисунку 1.1.

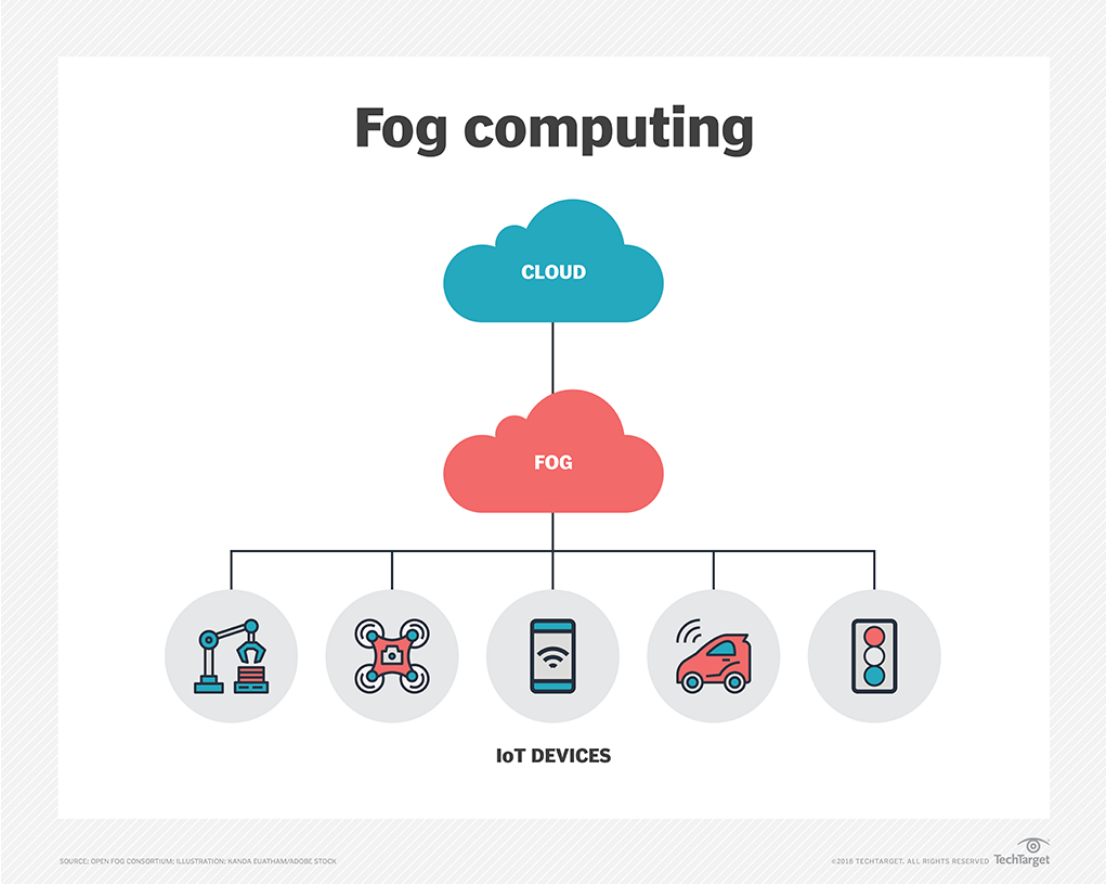

Рисунок 1.1 – Модель туманних обчислень

Туманні обчислення також включають хмарні обчислення – невеликі та досить потужні центри обробки даних, розташовані на граничному сегменті мережі.

Їхньою метою є підтримка ресурсомістких додатків інтернету речей, які потребують низького часу затримки.

Основна відмінність між туманними та хмарними обчисленнями полягає в тому, що хмара є централізованою системою, а туман є розподіленою децентралізованою інфраструктурою.

Туманні обчислення є посередником між обладнанням та віддаленими серверами.

Туманні обчислення визначають, яка інформація буде надіслана на сервер, і яку інформацію можна буде локально редагувати.

Таким чином, туман – це інтелектуальний шлюз, який розвантажує хмари, забезпечуючи більш ефективну обробку та аналіз даних.

Слід зазначити, що туманна мережа не є окремою архітектурою і не замінює хмарні обчислення, а скоріше доповнює їх максимально наближаючись до джерела інформації.

Нова технологія, можливо, вплине на інтернет речей, вбудовані рішення штучного інтелекту і 5G, оскільки вони, як ніколи раніше, вимагають швидкої та безперебійної роботи.

Переваги туманних обчислень:

- низький час відгуку (туман географічно ближчий до користувачів і здатний забезпечити миттєвий відгук);
- немає проблем із пропускною спроможністю (частина інформації агрегується в різних точках, а не вирушає в один центр по одному каналу);
- неможливість втрати з'єднання (через множину з'єднаних каналів)
- висока безпека (оскільки дані обробляються величезною кількістю вузлів у складній розподіленій системі);
- покращений інтерфейс користувача (миттєвий відгук і відсутність простоїв радують користувачів);
- енергетична ефективність (периферійні вузли використовують високоефективні протоколи, такі як Bluetooth, Zigbee або Z-хвиля).

Недоліки туманних обчислень:

- система туманних обчислень складніша (туман – додатковий шар у системі обробки та зберігання даних);

- додаткові витрати (компанії мають купувати периферійні пристрої-роутери, маршрутизатори, шлюзи);
- обмежений масштаб (на відміну хмари).

1.3 Порівняння туманних та хмарних обчислень

Традиційна архітектура централізованої хмари Cloud та обмеженість смуги пропускання мереж рівня ядра не дає можливість надсилати туди великі обсяги даних від кінцевих датчиків та сенсорів для обчислень та аналізу.

Fog Computing, з іншого боку, дає можливість пристроям Fog-вузлів на кордоні мережі виконувати деяку обробку даних у місці їх отримання та використання, щоб знизити затримку обміну та обсягу даних, що посилаються в Cloud, а також виконувати локальну аналітику.

Середній термін служби сервера у хмарних дата-центрах становить близько двох років. Для бізнес-додатків, які генерують велику кількість даних, такий частий апгрейд обладнання не тільки складний технічно, а й дуже дорогий. Замість інвестицій IT-департаментам організацій доводиться постійно займатися підтримкою відповідності своєї IT-системи поточним вимогам бізнесу. Однак, на межі мережі в невеликих розподілених дата-центрах компоненти серверної інфраструктури можуть служити до восьми років. Дослідження показують, що рішення, засновані тільки на централізованій хмарі Cloud, набагато витратніші, ніж на гібриній архітектурі Cloud+Fog.

Інша проблема, пов'язана з Cloud – швидкість обчислень та затримка обміну даними. При передачі в хмару швидкість обчислень знижується, а затримка обміну даними зростає. Однак, дані можуть мати різну цінність та призначення. Одні дані є найбільшою цінністю в момент їх збору. Через деякий час їхня цінність швидко падає. Інші дані призначені для накопичення певного обсягу для того, щоб аналітика на їх основі була релевантною.

Граничні обчислення та Fog-системи дозволяють аналізувати дані до того, як вони надсилаються до центральної хмари Cloud, в момент, коли їхня цінність максимальна. Наприклад, хакерську атаку або вторгнення в систему можна запобігти ефективніше, коли аналіз відбувається безпосередньо в місці атаки або вторгнення. На очікування пересилання даних до центрального дата-центру йде дорогоцінний час. Тому вилучення із загального, великого обсягу даних тієї частини, яку слід передати в Cloud, та відсікання непотрібних даних – дуже важливий процес, при якому цінні дані відокремлюються від малоцінних даних. Наприклад, безпілотний автомобіль генерує масу проміжних даних, які не потрібно зберігати в дата-центрі протягом тривалого часу. У таблиці 1.1 показано основні відмінності у вимогах до систем Cloud та Fog Computing.

Таблиця 1.1 – Відмінності у вимогах до систем Cloud та Fog Computing

	Вимоги	Cloud computing	Fog computing
1	Затримка даних	Висока	Низька
2	Джиттер затримки	Високий	Дуже низький
3	Розташування ПЗ послуги	В інтернеті	На кордоні мережі
4	Відстань між клієнтом та сервером	Багато переходів	Один перехід (hop)
5	Безпека	Важко поставити	Можна поставити
6	Перехоплення даних під час передачі	Висока ймовірність	Дуже низька ймовірність
7	Географічний розподіл	Централізоване	Місцеве
8	Число серверних вузлів	Декілька	Дуже багато
9	Підтримка рясності	Обмежена	Підтримується
10	Взаємодія у реальному часі	Підтримується	Підтримується
11	Коннективність «останньої милі»	Виділена лінія	Бездротова мережа

Fog computing також допомагає вирішити проблеми з даними промислових роботів. Переважна частина цих даних потрібна тільки в місці роботи робота, тому їхню обробку ефективніше і доцільніше проводити тут же. Передача цих даних до центральної хмари дуже затратна і часто технічно неможлива, а очікування команд з центру – це втрачений час.

У таблиці 1.2 показані основні проблеми Cloud та те, як вони вирішуються у Fog.

Таблиця 1.2 – Вирішення проблем хмарних обчислень за допомогою туманних обчислень

	Cloud computing	Fog computing
1	Дані та програми обробляються в централізованій хмарі Cloud, що при великому обсязі даних займає багато часу.	Замість того, щоб надсилати дані до центру, вони обробляються на межі мережі, тому час обробки суттєво знижується.
2	Проблема нестачі смуги пропускання, як результат посилки повного обсягу даних для обробки в централізовану хмару Cloud.	Найменша потреба в смузі пропускання мережі рівнів агрегації та ядра, за рахунок того, що біти даних агрегуються на вузлах Fog і частково там обробляються з видаленням надлишкових даних.
3	Повільна реакція та проблеми масштабування, як результат віддаленого розташування серверів.	Размещение небольших серверов, называемых «граничными серверами» (edge servers) неподалёку от пользователей в Fog-сети позволяет избежать времени ожидания отклика решить проблема масштабирования.

Можна сформулювати чотири основні причини необхідності Fog computing:

- Fog забезпечує обробку в реальному часі та управління кібер-фізичними системами CPS (cyber-physical system);
- Fog допомагає додаткам можуть відповідати вимогам користувачів;
- Fog надає середовище для пулінгу місцевих ресурсів (тобто гнучке їхнє перепризначення та перевикористання в місцевих системах);
- Fog дозволяє швидко виробляти інновації та масштабування із прийнятними витратами.

Можна зробити висновок про те, що Fog має переваги перед Cloud, проте не може повністю замінити централізовану хмару. Центральна хмара Cloud буде кращою у разі масивних та багато-потокових обчислень, потреба в яких залишається високою.

Fog і Cloud будуть взаємодоповнювати один одного і в той же час кожна з цих парадигм матиме свої переваги та недоліки.

Fog і тісно пов'язана з ним концепція граничних обчислень (Edge computing) гратимуть критичну роль у розвитку Інтернету Речів (IoT). Fog computing зростатиме за рахунок появи нових мережевих парадигм з вимогами швидкої обробки з меншою затримкою та джиттером.

Cloud computing буде служити цілям високопродуктивних обчислень, обробки великих обсягів різнорідних даних у ядрі штучного інтелекту, довгострокового зберігання даних, цінність яких або не зменшується, або з часом повільно падає.

1.4 Моделі обслуговування у хмарних обчисленнях

Виділяють три найбільш поширені моделі хмарних послуг:

- Infrastructure as a Service (IaaS) – інфраструктура як послуга.
- Platform as a Service (PaaS) – платформа як послуга.
- Software as a Service (SaaS) — програмне забезпечення як послуга.

Розглянемо кожну модель детально.

Інфраструктура як послуга – це надання обчислювальних ресурсів через хмару.

Як готове рішення клієнт може вибрати: сховище даних, віртуальний сервер, операційну систему та кількість ресурсів.

Модель IaaS наведена на рисунку 1.2.

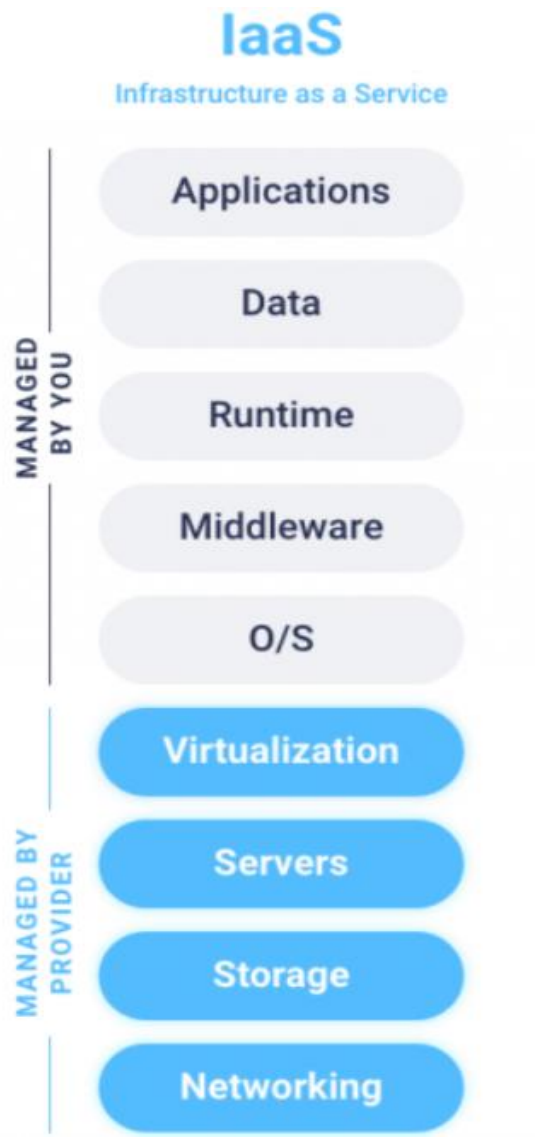

Рисунок 1.2 – Модель IaaS

IaaS часто використовують ті, хто хоче позбутися необхідності підтримувати власні локальні центри обробки даних. Покупка власного

серверного обладнання не потрібна, оскільки клієнт орендує його у провайдера IaaS і отримує у віртуальному вигляді через сервери хмар. Вони надаються організації через панель управління.За допомогою цього клієнти повністю контролюють всю інфраструктуру та можуть налаштувати її під потреби організації.

Користувачі IaaS самостійно керують програмами, операційними системами та спеціалізованим ПЗ, а провайдер підтримує роботу серверів, СГД та іншого фізичного обладнання.

IaaS – це найбільш гнучка модель хмарних послуг із простим процесом розгортання обладнання.

IaaS дозволяє підприємствам нарощувати обчислювальні ресурси за необхідності, замість того, щоб купувати дороге обладнання для власної інфраструктури.

Платформа як послуга (PaaS) надає середовище для розробників. Клієнти отримують доступ до платформи або набору інструментів для створення програм через Інтернет.

За допомогою послуг PaaS розробники можуть створювати все від простих мобільних додатків до складного програмного забезпечення для бізнесу.

Подібно до інших хмарних сервісів, PaaS дозволяє клієнтам користуватися сучасними потужними інструментами розробки, підтримку яких бере на себе провайдер. Платформа як послуга хороша тим, що одразу ж готова до роботи.

За допомогою PaaS підвищується швидкість розробки, тестування та доставки додатків. На готовій платформі команді розробників буде простіше та економічніше реалізовувати проекти будь-якого розміру та складності – витрати на розгортання платформи та проміжного ПЗ бере на себе провайдер. Хмарні технології дозволяють збільшувати/зменшувати ресурси за потреби. Декілька користувачів можуть отримати доступ до проекту через одну і ту ж платформу, яка в свою чергу може працювати з різними веб-службами та

базами даних.

Модель PaaS наведена на рисунку 1.3.

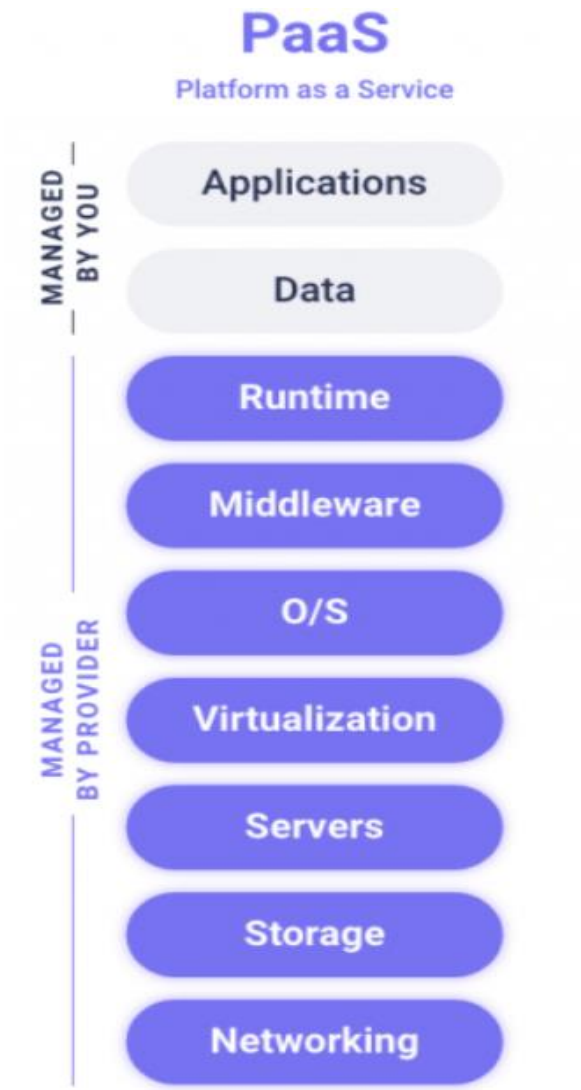

Рисунок 1.3 – модель PaaS

Програмне забезпечення як послуга – це надання клієнтам вже налаштованих програм для різноманітних бізнес-завдань через інтернет.

Як SaaS-рішень можуть надаватися CRM, ERP, ITSM-системи, таск-трекери та інше програмне забезпечення.

Віддалена, налаштування та обслуговування ПЗ провайдером надає компанії-замовнику більше часу для вирішення інших важливих питань та завдань. SaaS-рішення керуються централізовано і розміщуються на

віддаленому сервері. Виробник, а не користувач, несе відповідальність за налаштування необхідного обладнання та програмного забезпечення.

Модель IaaS наведена на рисунку 1.4.

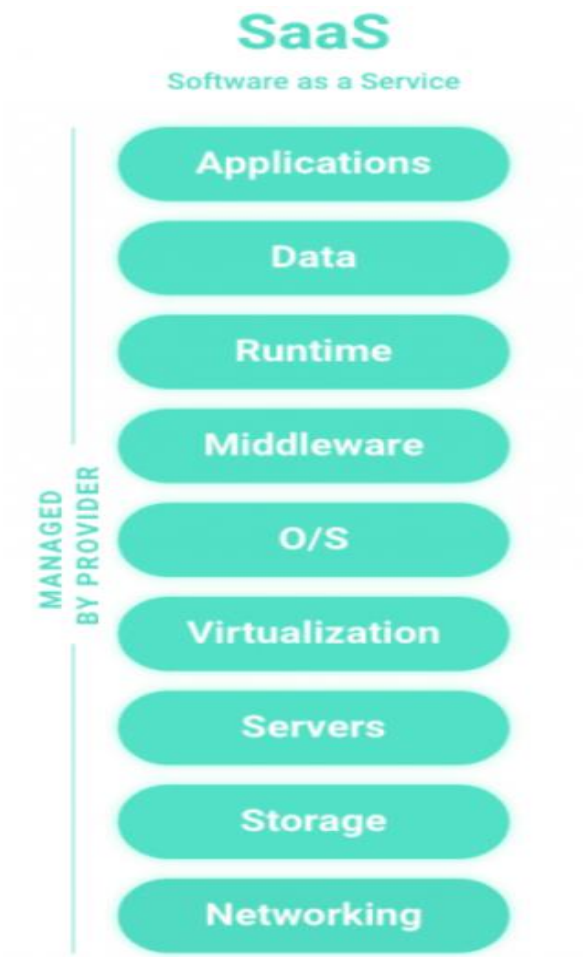

Рисунок 1. 4 – Модель SaaS

Найчастіше для роботи SaaS не потрібно завантаження та встановлення ПЗ на пристрій, більшість програм запускаються в браузері.

2 ОБГРУНТУВАННЯ ВИБОРУ ПРОГРАМНОГО ІСТРУМЕНТАРІЮ

2.1 Особливості еластичних обчислень

Еластичні обчислення – це можливість швидко нарощувати та звільняти ресурси процесорів, пам'яті та зберігання для задоволення змінних вимог без необхідності планувати завантаження та вживати заходів для обробки пікових навантажень. Засоби системного моніторингу, які зазвичай керують еластичними обчисленнями, без переривання операцій масштабують кількість виділених ресурсів відповідно до фактичної кількості, необхідної для обробки даних. Завдяки еластичності середовища компанії не потрібно оплачувати ресурси, що не використовуються або простоюють, а також турбуватися про придбання або обслуговування додаткових ресурсів та обладнання.

Еластичні обчислення мають безліч переваг, незважаючи на обмежене управління та проблеми, пов'язані з безпекою. Еластичні обчислення ефективніші, ніж звичайна IT-інфраструктура. Як правило, вони автоматизовані і не потребують цілодобового адміністрування з боку людини, тим самим забезпечуючи безперервну доступність служб без уповільнення чи переривання обслуговування.

У хмарних або туманних обчисленнях еластичність визначається як «ступінь, в якій система здатна адаптуватися до змін робочого навантаження шляхом надання та відключення ресурсів автономним чином, так що у кожний момент часу доступні ресурси максимально відповідають поточному попиту». Еластичність - це визначальна характеристика, яка відрізняє хмарні та туманні обчислення від запропонованих раніше обчислювальних парадигм, наприклад, таких як грид-обчислення. Динамічна адаптація потужності, наприклад, шляхом зміни використання обчислювальних ресурсів для задоволення робочого навантаження, що змінюється, називається «еластичними обчисленнями».

Проілюструємо еластичність обчислень на простому прикладі постачальника послуг, який хоче запустити веб-сайт у хмарі IaaS. На момент t0 веб-сайт непопулярний, для обслуговування всіх користувачів Інтернету достатньо однієї машини (зазвичай у хмарах, віртуальної машини). В момент t1 веб-сайт раптово стає популярним, наприклад, в результаті Slashdot-ефекту, або слэшдотингу, який виникає, коли популярний веб-сайт посилається на менший веб-сайт, що спричиняє значне збільшення трафіку (рисунок 2.1, час близько 19.30).

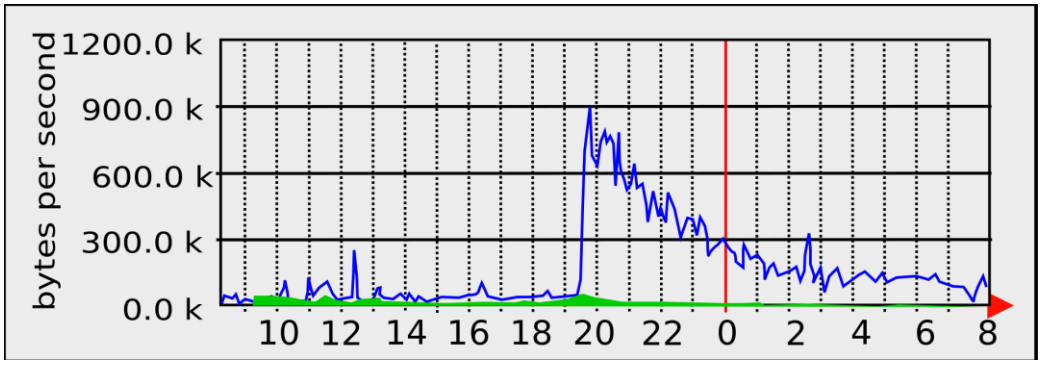

Рисунок 2.1 – Приклад появи Slashdot-ефекту

Такий стрибок перевантажує менший сайт, спричиняючи його сповільнення або навіть тимчасову недоступність. Як правило, менш надійні сайти не в змозі впоратися з величезним збільшенням трафіку та стають недоступними. Поширеними причинами є недостатня пропускна здатність даних, сервери, які не справляються з великою кількістю запитів, і квоти трафіку. Сайти, які підтримуються на спільних службах хостингу, часто виходять з ладу, коли стикаються з Slashdot-ефектом, який у таких випадках має той самий ефект, що й атака на відмову в обслуговуванні, хоча й випадково. Отже, в цьому випадку однієї віртуальної машини стає недостатньо для обслуговування всіх користувачів. Залежно від кількості веб-користувачів, які одночасно здійснюють доступ до веб-сайту, та вимог до ресурсів веб-сервера, може знадобитися, допустимо десять машин. Еластична

система повинна негайно виявити цю умову та виділити дев'ять додаткових машин із хмари, щоб оперативно обслуговувати всіх веб-користувачів.

На момент t2 (рисунок 2.1, час близько третьої години опівночі) сайт знову стає непопулярним. Десять комп'ютерів, які зараз виділені сайту, в основному простоюють, і однієї, максимум двох машин, буде достатньо для обслуговування кількох користувачів, які звертаються до сайту. Еластична система має негайно виявити цей стан, деініціалізувати вісім або дев'ять машин і передати їх у хмару.

Еластичність націлена на зіставлення обсягу ресурсів, виділених для служби, з обсягом ресурсів, які їй дійсно потрібні, щоб уникнути надлишкового чи недостатнього виділення ресурсів. Надлишкового забезпечення, тобто виділення більше ресурсів, ніж потрібно, слід уникати, оскільки постачальник послуг часто повинен платити за ресурси, що виділяються для послуги. У цьому випадку витрати постачальника послуг вищі за оптимальні, а прибуток знижується.

Недостатньої ініціалізації, тобто виділення меншої кількості ресурсів, ніж потрібно, при еластичних обчисленнях слід уникати, інакше служба не зможе надати своїм користувачам гарний сервіс. У прикладі, наведеному вище, недостатня підготовка веб-сайту може зробити його повільним або недоступним. Інтернет-користувачі зрештою відмовляються від доступу до нього, таким чином, постачальник послуг втрачає клієнтів. У довгостроковій перспективі дохід провайдера зменшуватиметься, що також знижує прибуток.

2.2 Проблеми, що виникають при еластичних обчисленнях

Одна потенційна проблема полягає в тому, що еластичність потребує часу. Хмарна віртуальна машина (ВМ) може бути придбана користувачем у будь-який час, практично відразу по потребі, проте підготовка придбаної віртуальної машини може тривати кілька хвилин. Час запуску віртуальної

машини залежить від чинників, як розмір образу, тип віртуальної машини, розташування центру обробки даних, кількість віртуальних машин тощо. У постачальників хмарних послуг різна продуктивність запуску віртуальних машин. Це означає, що будь-який механізм управління, розроблений для еластичних застосунків, повинен враховувати в процесі прийняття рішення час, необхідний для того, щоб дії з еластичності набули чинності, наприклад, надання іншій віртуальній машині конкретного компонента програми.

Еластичні програми можуть виділяти та звільняти ресурси (наприклад, віртуальні машини) на запит для певних компонентів програми. Це робить хмарні ресурси нестабільними, а традиційні інструменти моніторингу, які пов'язують дані моніторингу з конкретним ресурсом (наприклад, віртуальними машинами, такими як Ganglia або Nagios), не підходять для моніторингу поведінки еластичні аплікації. Наприклад, протягом терміну служби рівень сховища даних еластичного застосунка може додавати та видаляти віртуальні машини сховища даних через вимоги до вартості та продуктивності, варіюючи кількість використовуваних віртуальних машин. Таким чином, для моніторингу еластичних програм потрібна додаткова інформація, така, як зв'язування логічної структури програми з базовою віртуальною інфраструктурою. Це, у свою чергу, породжує інші проблеми, наприклад, як агрегувати дані з кількох віртуальних машин для отримання поведінки компонента програми, що працює поверх цих віртуальних машин, оскільки різні метрики, можливо, необхідно агрегувати по-різному (наприклад, використання ЦП може бути усереднено , мережа переклад може бути підсумована). При розгортанні застосунків у хмарних інфраструктурах (IaaS/PaaS) необхідно враховувати вимоги зацікавлених сторін, щоб забезпечити належну еластичність.

Незважаючи на те, що традиційно можна було б спробувати знайти оптимальний компроміс між вартістю і якістю або продуктивністю, для реальних користувачів хмари вимоги до поведінки складніші і націлені на кілька вимірів еластичності (наприклад, SYBL, рисунок 2.2).

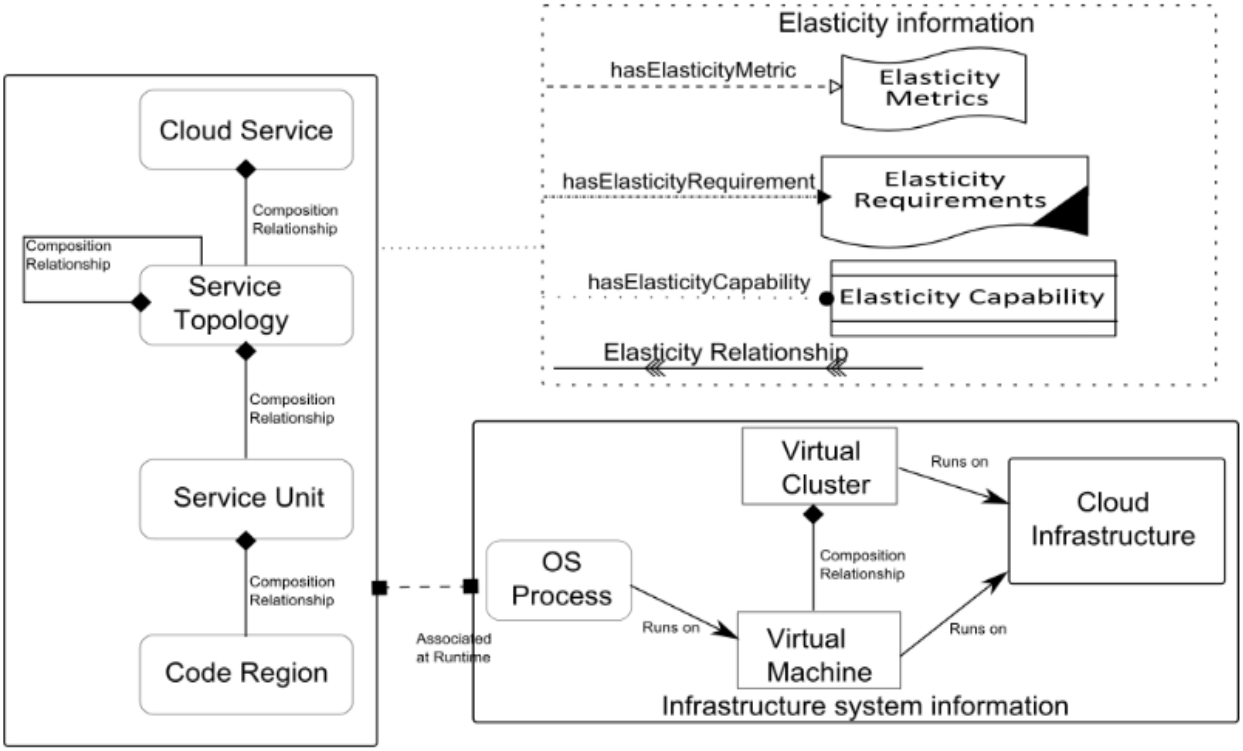

Рис. 2.2 – Схема багатовимірної еластичності SYBL

Хмарні програми можуть бути різних типів та складності, з кількома рівнями артефактів, розгорнутими на різних рівнях. При управлінні такими структурами необхідно враховувати безліч питань, одним з низки запропонованих підходів у цьому сенсі може бути rSYBL (рисунок 2.3).

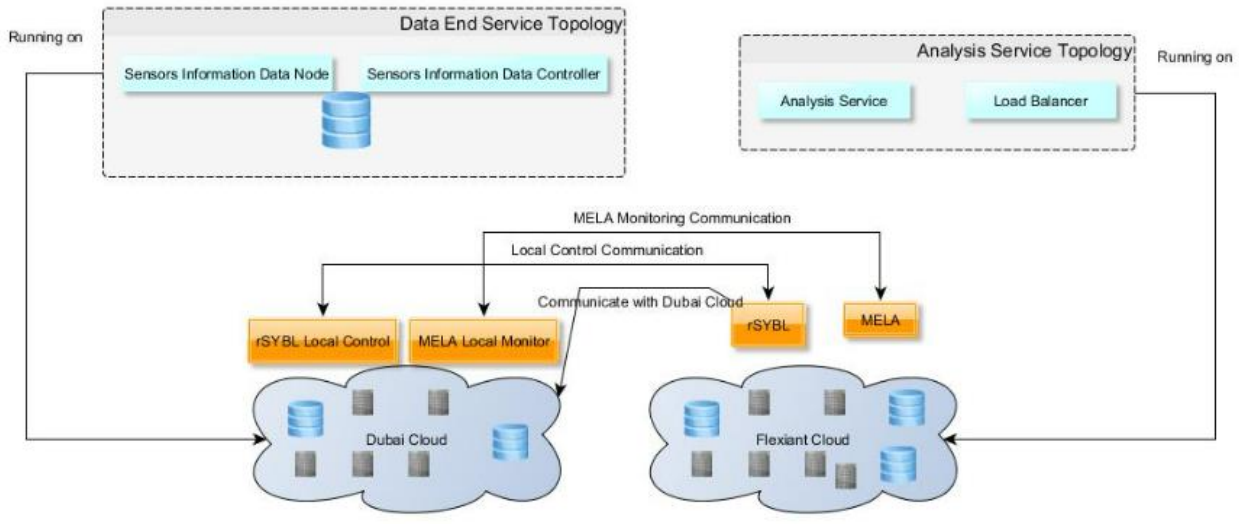

Рисунок 2.3 – Мультипідхід до багатовимірної еластичності rSYBL

Для багаторівневого керування системи керування повинні враховувати вплив керування нижнього рівня на більш високі та навпаки (наприклад, одночасне керування віртуальними машинами, веб-контейнерами або веб-службами), а також конфлікти, які можуть виникнути. між різними стратегіями управління із різних рівнів. Еластичні стратегії у хмарах можуть використовувати переваги теоретико-керуючих методів (наприклад, прогнозуюче управління було випробувано у хмарних сценаріях, продемонструвавши значні переваги порівняно з реактивними методами).

Контрольне вікно керуючої програми еластичними обчисленнями наведено на рисунку 2.4.

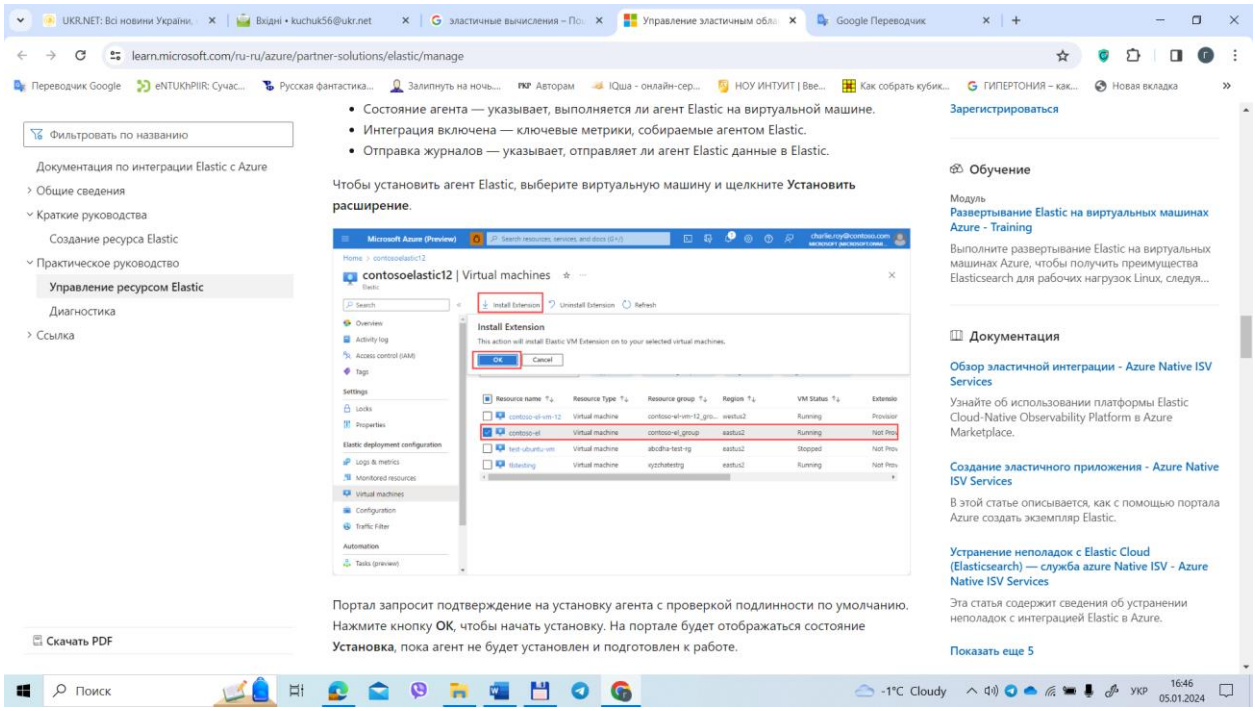

Рисунок 2.4 – Керування еластичними обчисленнями в MS Azure

Платформа Elastic Cloud-Native Observability Platform централізує журнали, метрики та аналітику трасування в одному місці. Це дозволяє простіше відслідковувати працездатність та продуктивність середовища Azure. Ця інформація допомагає також усунути несправності у службах. Завдяки пропозиції Elastic можна керувати рішенням через портал Azure.

Elastic впроваджується як рішення для моніторингу хмарних робочих навантажень за допомогою оптимізованого робочого процесу.

2.3. Вибір мови програмування

Мова програмування Python славиться своєю простотою і лаконічністю. Небагатослівний і зрозумілий синтаксис, схожий на псевдокод, а також сильна динамічна типізація сприяють навчанню новачків. В останні роки Python придбав свою популярність внаслідок ефективності в таких сферах розробки, як Machine Learning і Data Science завдяки своїй розширюваності і гнучкості, що так необхідно в даному сегменті програмування.

Інтерпретатор мови бере на себе всю роботу, звільняючи програміста від необхідності ручного управління пам'яттю. Практична неможливість отримати segmentation fault, а також зручна система винятків дозволяють оперативно налагоджувати програми. Ситуації, коли їх падіння, через виниклі помилки, вимагають глибокого дебаггінга, досить рідкі. Для обгрунтування вибору необхідно виокремити плюси мови програмування.

На Python можливо розробляти прості програми, знаючи тільки англійську мову, навіть без досвіду в програмуванні. У порівнянні з багатьма іншими мовами Python має зрозумілий синтаксис та підходить для Linux, Windows та Mac Os, також має реалізацію інтерпретаторів для мобільних пристроїв і непопулярних систем.

Використається для розробки веб-додатків, ігор, зручний для автоматизації, математичних обчислень, машинного навчання, в галузі інтернету речей. Існує реалізація під назвою Micro Python, оптимізована для запуску на мікроконтролерах (можна писати інструкції, логіку взаємодії пристроїв, організовувати зв'язок, реалізовувати розумний будинок).

У світі Python багато якісних бібліотек, так що не потрібно винаходити велосипед, якщо треба терміново вирішити завдання. Для навчання є багато

корисних книг, в першу чергу англійською мовою, звичайно, але і в перекладі також видана велика кількість літератури.

Python відрізняється суворою вимогою до написання коду (вимагає відступи), що можно віднести до переваг. З самого початку мова сприяє писати код організовано.

2.4. Вибір інтегрованого середовища розробки

Інтегроване середовище розробки (IDE) — комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм (рисунок 2.5).

Рисунок 2.5 – Узагальнена структура IDE

PyCharm – це інтегроване середовище розробки для Python, яка має повний комплект засобів, необхідних для ефективного програмування на Python. Розглянемо переваги PyCharm.

PyCharm має зручний редактор коду з усіма корисними функціями: підсвічуванням синтаксису, автоматичним форматуванням, доповненням і відступами. PyCharm дозволяє перевіряти версії інтерпретатора мови на сумісність, а також використовувати шаблони коду.

PyCharm дозволяє швидко робити рефакторинг коду, а також має зручний графічний відладчик. У PyCharm можна проводити інтегроване Unit тестування, використовувати інтерактивні консолі для Python, SSH, відладчика і баз даних.

PyCharm має велику колекцію плагінів, і його можна використовувати в зв'язці з різними трекерарами, такими як JIRA, Youtrack, Lighthouse, Redmine, Trac і так далі. PyCharm крос-платформне середовище розробки: можна використовувати на Linux, Windows і Mac OS. PyCharm можна назвати однією з кращих IDE для Python. Залежно від своїх можливостей і потреб можна вибрати або платну професійну версію, або безкоштовну версію для спільноти.

2.5 Система контролю версій

Незалежно від мови або напрямку розробки, файли коду регулярно додаються, видаляються і змінюються. Деякі з них можуть містити сотні рядків коду, а інші тисячі. Поки проєкт складається з пари-трійки файлів, його розробка не створює ніяких складнощів. З ростом кодової бази з'являються певні питання, які предсталені нижче.

Як не втратити файли з вихідним кодом?

Як захиститися від випадкових виправлень і вилучень?

Як скасувати зміни, якщо вони виявилися некоректними?

Як одночасно підтримувати робочу версію і розробку нової?

Система контролю версій вирішує ці питання за допомогою спеціальних програм, які відстежують зміни коду.

Git – одна із таких програм. Це розподілена система контролю версій,

яка дає можливість розробникам відстежувати зміни в файлах і працювати над одним проєктом спільно з колегами (рис. 2.6).

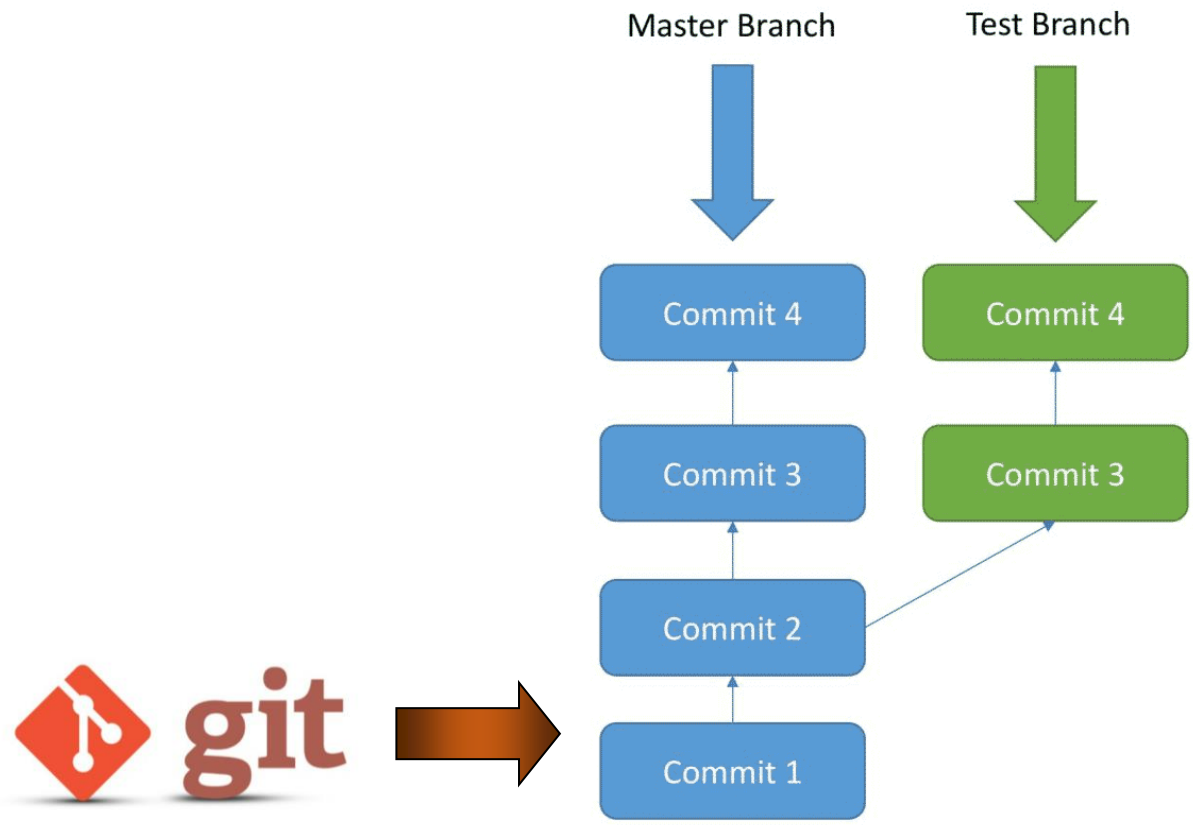

Рисунок 2.6 – Схема функціонування Git

Її розроблено в 2005 році Лінусом Торвальдсом, творцем Linux задля того, щоб інші розробники мали можливість вносити свій вклад в ядро Linux. Git відомий своєю швидкістю, простим дизайном, підтримкою нелінійної розробки, повною децентралізацією і можливістю ефективно працювати з великими проєктами.

Підхід Git до зберігання даних схожий на набір знімків мініатюрної файлової системи. Кожен раз, коли користувачем зберігається стан проєкту в Git, система запам'ятовує, як виглядає кожен файл в цей момент, і зберігає посилання на цей знімок. Переваги Git:

- безкоштовний і open-source, є можливість безкоштовного користування і внесення будь-яких змін у свій вихідний код;

- невеликий і швидкий, виконує всі операції локально, що збільшує його швидкість, крім того, Git локально зберігає весь репозиторій у невеликий файл без втрати якості даних;

- резервне копіювання, Git ефективний в зберіганні бекапів, отже випадків втрачання даних при використанні Git практично не існує;

- просте розгалуження, в інших системах контролю версій створення гілок – стомлююча та трудомістка задача, адже увесь код копіюється в нову гілку, у Git управління гілками реалізовано просто і ефективно.

Разом з Git можна використовувати і GitHub.

GitHub – сервіс онлайн-хостингу репозиторіїв, що має всі функції розподіленого контролю версій і функціональність управління вихідним кодом - все, що підтримує Git і навіть більше. Зазвичай він використовується разом з Git і дає розробникам можливість зберігати їхній код онлайн, а потім взаємодіяти з іншими розробниками в різних проектах.

Також GitHub може похвалитися контролем доступу, багтрекінгом, управлінням завданнями та вікі для кожного проекту. Мета GitHub – сприяти взаємодії розробників.

До проекту, завантаженого на GitHub, можна отримати доступ за допомогою інтерфейсу командного рядка Git та Git-команд. Також є й інші функції, такі як документація, запити на прийняття змін (pull requests), історія коммітів, інтеграція з безліччю популярних сервісів, email-повідомлення, емодзі, графіки, вкладені списки завдань, система @згадувань, схожа на ту, що в Twitter тощо.

Проект GitHub зберігається в репозиторії (repository) – колекції всіх змін створюваного коду. Якщо працювати над проектом одному – потрібно створити новий репозиторій. Якщо у проекті кілька розробників – кожен із них клонуватиме репозиторій початкового роробнику проекту.

Усередині репозиторію зміни коду зберігаються у вигляді гілок та комітів, так, як це наведено на приклади рисунку 2.7.

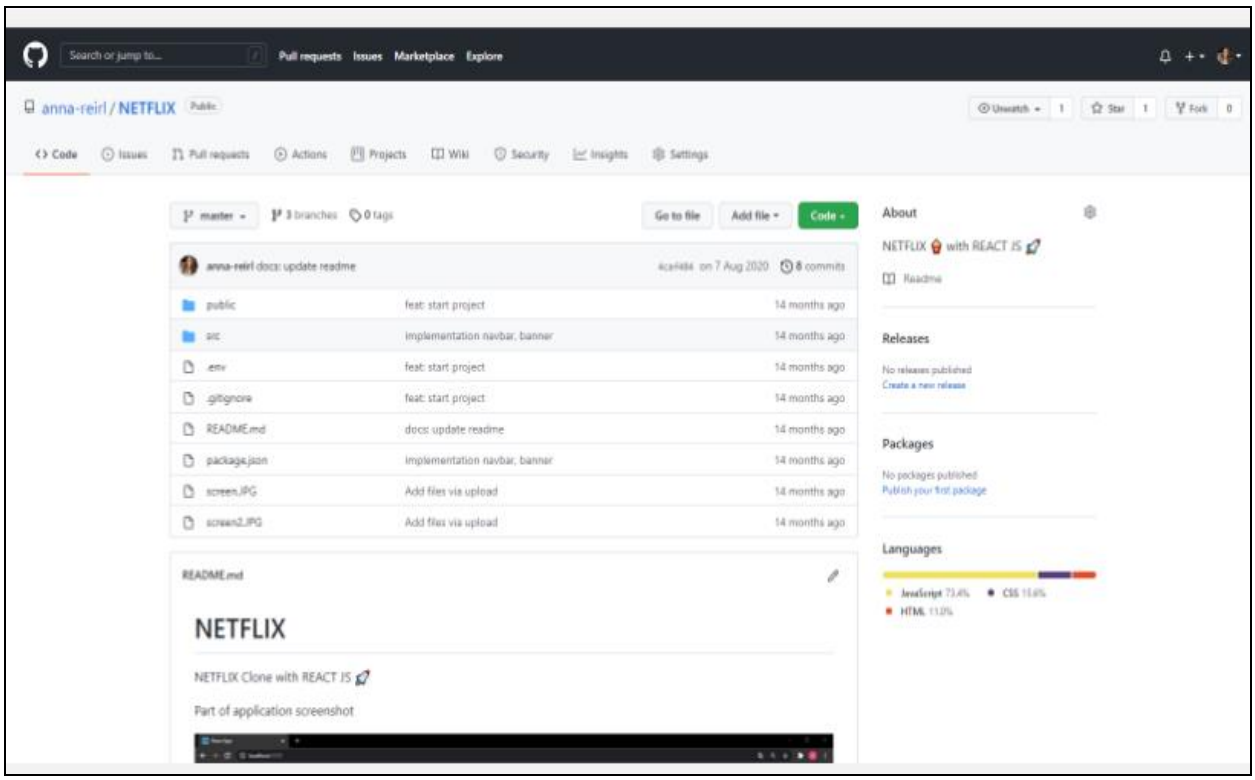

Рисунок 2.7 – Вікно репозиторію

Коміт (commit) – основний об'єкт розробки, у якому зберігаються зміни коду за итерацию. По суті, це список з усіма актуальними змінами та посилання на попередню версію комміту. Кожен комміт має атрибути: ім'я, дату створення, автор і коментарі до поточної версії (наприклад, «Створив сторінку courses.html» при розробці сайтів з відеокурсами).

Гілка (branch) – покажчик на коміт із певними змінами. Наприклад, два розробники взяли коміт, і кожен з них зробив свої зміни в коді, створивши за новим комітом («Створив сторінку courses.html з особистим кабінетом» та «Створив сторінку courses.html з вільним доступом на курси»). Так, у проекті з'явилися дві гілки з різним кодом: розробник може вибрати, над яким коммітом йому працювати далі.

Основною гілкою проекту, як правило, вважається гілка main або master – розробники створюють нові гілки на її основі. Також можна створити необмежену кількість гілок, щоб вносити нові зміни, не заважаючи основному проекту.

Часто розробники роблять паралельні зміни коду. Наприклад, один розробник працює над зовнішнім виглядом сайту, а інший займається розміщенням контенту на ньому. Після закінчення роботи гілки кожного з них можна об'єднати в одну, щоб створити коміт із усіма внесеними розробниками змінами.

Для цього Git використовують функцію pull request (pr). Pull request – це заявка на злиття коду з різних гілок. У процесі злиття Git створить коміт і покаже всі зміни у файлі коду: додані до розгалуження рядки підсвічуються зеленим кольором, видалені червоним. Так кожен із розробників та менеджер проекту побачать, що сталося з кодом після спільної роботи над коммітом. Перед остаточним злиттям (merge) усі розробники повинні переглянути зміни коду (code review) та прийняти їх.

3 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МОДЕЛІ ЕЛАСТИЧНИХ ОБЧИСЛЕНЬ

3.1 Мережі Петрі

При реалізації було вирішено скористатися мережами Петрі через їх можливість моделювати динамічні дискретні системи (зокрема асинхронні паралельні процеси).

Мережі Петрі – математичний апарат для моделювання динамічних дискретних систем. Вперше описані Карлом Петрі у 1962 році.

Мережа Петрі є дводольним орієнтованим графом, що складається з вершин двох типів – позицій і переходів, з'єднаних між собою дугами. Вершини одного типу неможливо знайти з'єднані безпосередньо. У позиціях можуть розміщуватись мітки (маркери), здатні переміщатися по мережі.

Подією називають спрацювання переходу, у якому мітки з вхідних позицій цього переходу переміщуються у вихідні позиції. Події відбуваються миттєво, чи разночасно, і під час деяких умов. Приклад роботи мережі Петрі наведений на рисунку 3.1.

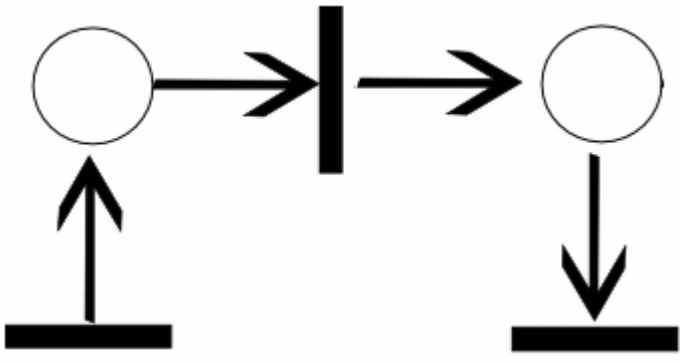

Рисунок 3.1 – Приклад роботи мережі Петрі

Як і стандартні UML діаграми, BPMN та EPC, мережі Петрі надають можливість графічно ілюструвати процеси, що включають вибір, ітерації та одночасне виконання. Але на відміну від даних стандартів, мережі Петрі

мають чітке математичне формулювання і за ними стоїть розвинена математична теорія.

Введемо всі необхідні поняття та визначення.

Мережі Петрі - інструмент дослідження систем. Теорія мереж Петрі уможливлює моделювання системи математичним уявленням її у вигляді мережі Петрі. Передбачається, що аналіз мереж Петрі допоможе отримати важливу інформацію про структуру та динамічну поведінку системи, що моделюється. Ця інформація буде корисна для оцінки моделюваної системи та вироблення пропозицій щодо її вдосконалення та зміни.

Мережа Петрі складається з чотирьох елементів:

- множина позицій P;
- множина переходів T;
- вхідна функція I;
- вихідна функція O.

Вхідна і вихідна функції пов'язані з переходами і позиціями. Вхідна функція I відображає перехід $\mathrm{t_j}$ в множину позицій $I(\mathrm{t_j})$, що називаються вхідними позиціями переходу.

Вихідна функція O відображає перехід $\mathrm{t_j}$ в множину позицій $\mathrm{O(t_j)}$, що називаються вихідними позиціями переходу.

Структура мережі Петрі визначається її позиціями, переходами, вхідною та вихідною функціями.

Визначення. Мережа Петрі C є четвіркою $C = (P, T, I, O)$.

$P = (\mathrm{p_1}, \mathrm{p_2}, \ldots, \mathrm{p_n})$ - кінцева множина позицій, $n \geq 0$,

$T = (\mathrm{t_1}, \mathrm{t_2}, \ldots, \mathrm{t_m})$- кінцева множина переходів, $m \geq 0$.

Безліч позицій і безліч переходів не перетинаються, $P \cap T = \emptyset$.

$I : T \to P$ - є вхідною функцією,

$O: T \to P$ – є вихідною функцією.

$$C = (P, T, I, O)$$

$$P = (\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \mathrm{p}_4, \mathrm{p}_5)$$

$$I(\mathrm{t}_1) = \{\mathrm{p}_1\}, I(\mathrm{t}_2) = \{\mathrm{p}_2\}, I(\mathrm{t}_3) = \{\mathrm{p}_4\},$$

$$I(\mathrm{t}_4) = \{\mathrm{p}_4\}, I(\mathrm{t}_5) = \{\mathrm{p}_2\}, I(\mathrm{t}_6) = \{\mathrm{p}_5\},$$

$$O(\mathrm{t}_1) = \{\mathrm{p}_2\}, O(\mathrm{t}_2) = \{\mathrm{p}_3\}, O(\mathrm{t}_3) = \{\mathrm{p}_4\}$$

$$O(\mathrm{t}_4) = \{\mathrm{p}_1\}, O(\mathrm{t}_5) = \{\mathrm{p}_5\}, O(\mathrm{t}_6) = \{\mathrm{p}_1\}$$

Представлена структура мережі Петрі представлена у вигляді четвірки, що складається з множини P позицій, множини T переходів, вхідної функції $I : T \to P$ та вихідної функції $O: T \to P$.

Визначення. Потужність множини P є число n, а потужність множини T є число m. Довільний елемент P позначається символом $\mathrm{p}_i, i = 1, \dots, n$, а довільний елемент T символом $\mathrm{t}_j, j = 1, \dots, m$.

Визначення. Позиція p_i є вхідною позицією переходу t_j у разі, якщо $\mathrm{p}_i \in I(\mathrm{t}_j)$. p_i є вихідною позицією t_j, якщо $\mathrm{p}_i \in O(\mathrm{t}_j)$.

В значній мірі, теоретична робота по мережах Петрі ґрунтується на формальному визначенні мереж Петрі, викладеному вище. Тим не менш для ілюстрації понять теорії мереж Петрі набагато зручніше графічне уявлення мережі Петрі.

Теоретико-графовим поданням мережі Петрі є орієнтований двудольний мультиграф.

Визначення. Граф G мережі Петрі є дводольний орієнтований мультиграф G = (V, A), де $V = (v_1, v_2, \dots, v_n)$ множина вершин, $A = (\alpha_1, \alpha_2, \dots, \alpha_m)$ -спрямовані дуги, $\alpha_i = (v_j, v_k)$, де $v_j, v_k \in V$ (рисунок 3.1). Множина V може бути розбита на дві непересічні підмножини P і T таких, що $P \cup T = V$, $P \cap T = \emptyset$, та для будь-якої спрямованої дуги $\alpha_i \in A$, якщо $\alpha_i = (v_j, v_k)$, тоді або $v_j \in P$ $v_k \in T$, або $v_j \in T$ і $v_k \in P$.

Приклад графу мережі Петрі наведений на рисунку 3.2.

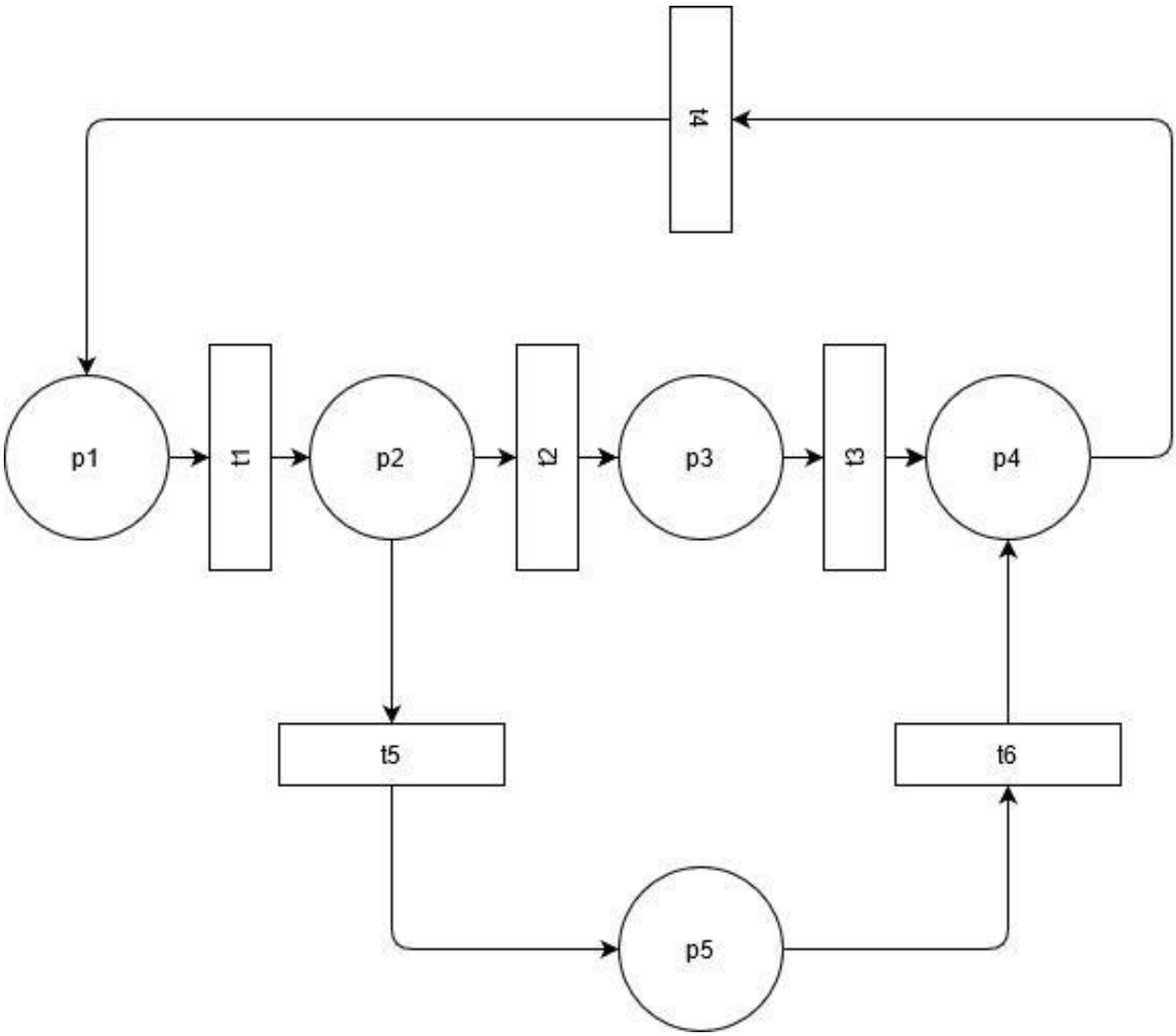

Рисунок 3.2 – Граф мережі Петрі

При перетворенні мережі Петрі на граф, визначимо $V = P \cup T$. А визначимо як безліч спрямованих дуг, таких, що для всіх $\mathrm{p_i} \in P.$ и $\mathrm{t_j} \in T.$

$$\#((\mathrm{p_i}, \mathrm{t_j}), A) = \#(\mathrm{p_i}, I(\mathrm{t_j}))$$

$$\#((\mathrm{t_j}, \mathrm{p_i}), A) = \#(\mathrm{p_i}, O(\mathrm{t_j}))$$

Тоді G = (V, A) є граф мережі Петрі, еквівалентний структурі мережі Петрі. Перетворення з графа на мережу Петрі відбувається аналогічно.

3.2. Організація еластичних обчислень

Еластичні обчислення вимагають реалізації моделі IaaS (потрібно динамічно додавати нові та видаляти старі ресурси з інфраструктури) і Paas (потрібно динамічно виділяти та звільняти ресурси в системі). Для цього розберемо основні засади реалізації моделі хмарних обчислень.

Є деяка робота (Job), яку можна розбити на набір завдань (Task), які необхідно виконати, причому виконання деяких із них можуть йти паралельно різних працівниках(worker) - одиниці обчислень, у своїй деякі завдання може бути виконані лише на деяких працівниках, таким чином потрібен менеджер завдань, який розподілятиме завдання між доступними працівниками, цьому завдання потрібно розподілити так, щоб встигнути виконати їх до настання їхнього дедлайну. Таким чином досягається еластичність обчислень, тобто ресурси виділяються якщо з наявними виділеними ресурсами завдання не вкладається в дедлайни та звільняються, якщо вони є у надлишку.

Звідси випливає необхідність оцінки часу виконання, для того щоб укластися в дедлайни завдань, які необхідно виконати. Цю оцінку можна зробити статичною, тобто знайти максимальне час необхідний виконання завдання, проте при такому підходу виходить, що на виконання завдання завжди виділяється максимум ресурсів, незалежно від того, за скільки часу насправді виконується робота на конкретному працівнику, що призводить до неоптимального використання їх ресурсів. Тому варто зробити оцінку адаптивною, тобто обчислювати скільки часу йде на виконання завдання на кожному працівнику, та за цими даними прогнозувати, скільки завдання займе часу наступного разу.

Цей підхід більш оптимальний у плані використання ресурсів, проте складніший у реалізації, і навіть вводить додатковий оверхед, що необхідний для обчислення часу виконання. Однак, виграш ресурсів у результаті оптимального їх використання переважує оверхед, тому на практиці

використовується саме адаптивна оцінка часу. Розглянемо цей метод докладніше.

Для адаптивної оцінки знадобиться побудувати модель знань, керуючись якою, і будуть виділятися додаткові працівники у разі якщо поточної кількості ресурсів не вистачає для вкладання в дедлайн або навпаки, забиратися зайві, якщо їх надлишок. При побудові цієї моделі використовуються дані про час виконання для реальних працівників, після аналізу яких існуюча модель буде доповнена. Також ці дані будуть використані для симуляції обчислювальної потужності працівника в майбутньому.

Модель хмарних обчислень із базою знань наведена на рисунку 3.3.

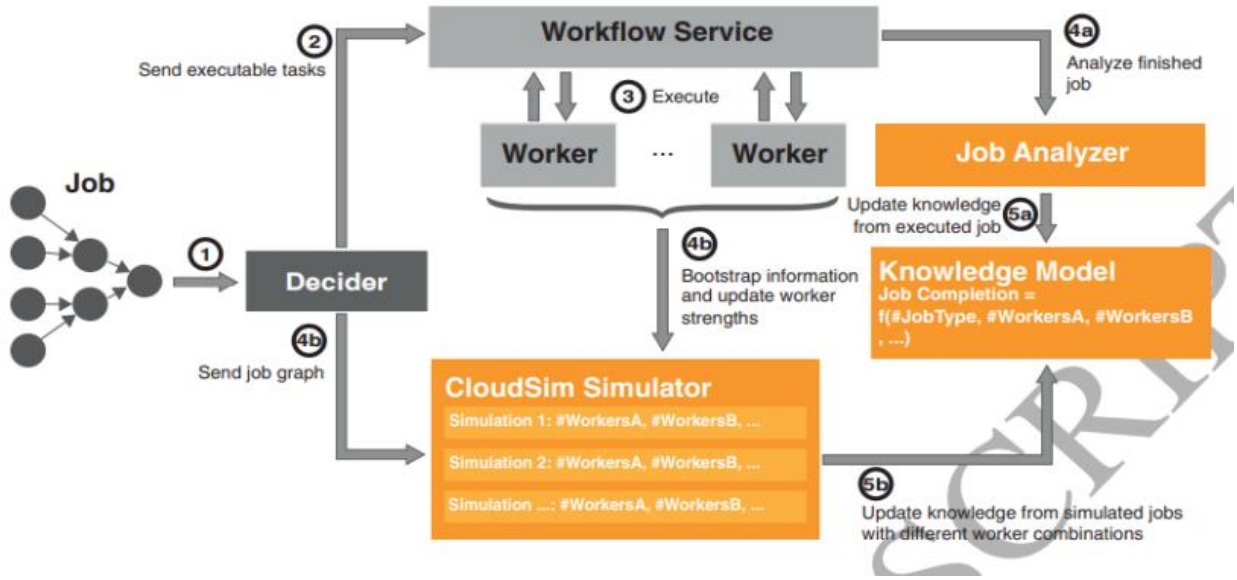

Рисунок 3.3 – Модель хмарних обчислень із базою знань

Зупинимося докладніше на моделі виділення ресурсів для виконання завдань. У найпростішому вигляді виконання завдання можна подати, як показано на рисунку 3.4.

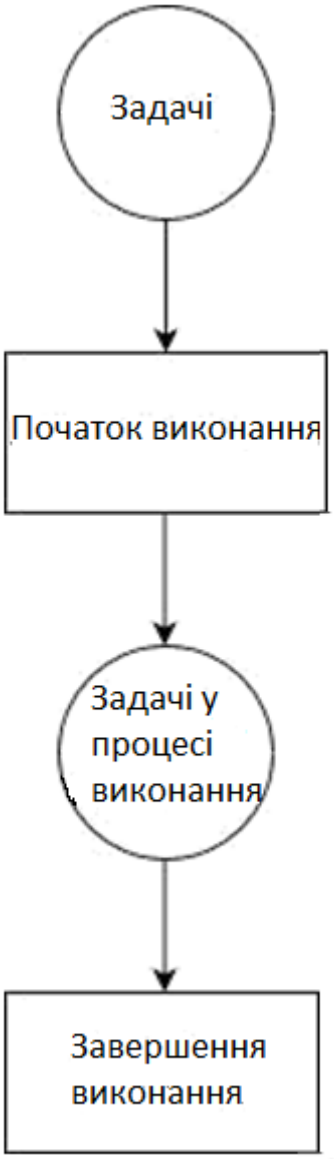

Рисунок 3.4 – Просте подання виконання задачі

У цій схемі не враховується необхідність виділення ресурсів для виконання програми, як на рисунку 3.5, а також можливість наявності пріоритету у завдань, що зображено на рисунку 3.6, у відповідності з яким завдання з більш низьким пріоритетом повинні отримувати ресурси для виконання лише після того, як отримали завдання з вищим пріоритетом.

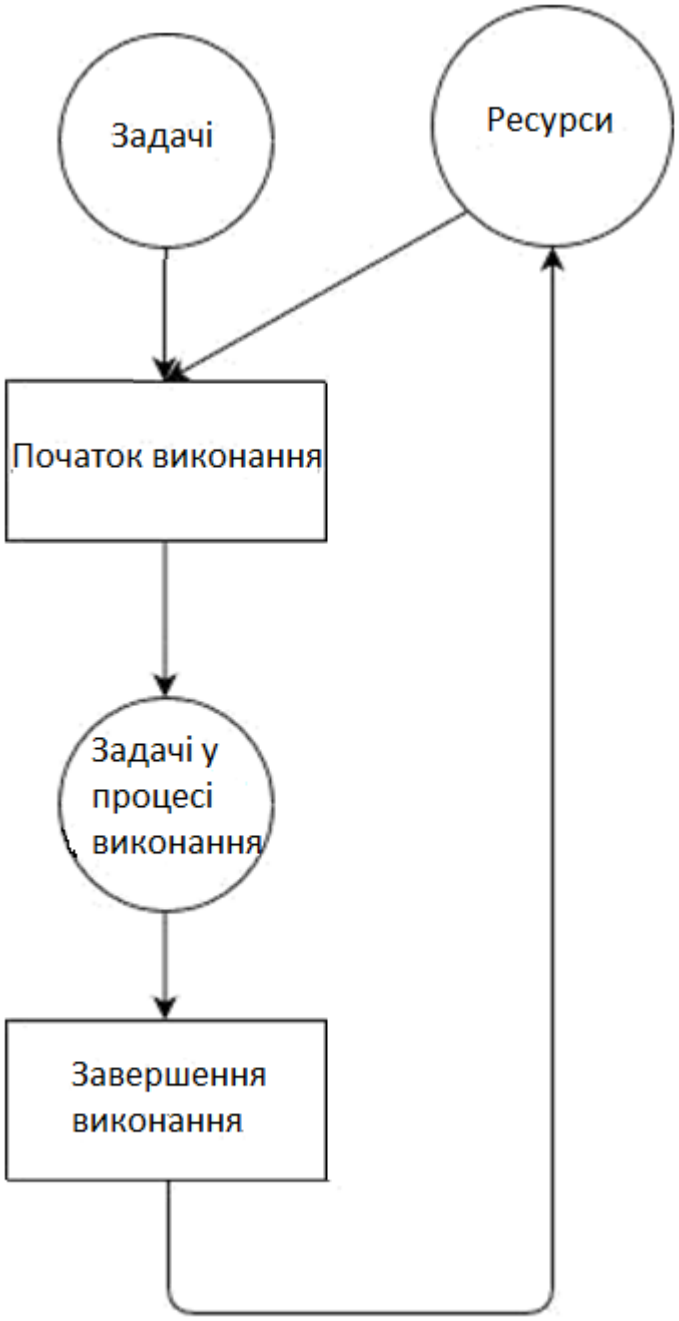

Рисунок 3.5 – Подання виконання задачі, що враховує ресурси

Для виконання завдання можуть бути потрібні різні типи ресурсів (пам'ять, процесори), але для простоти уявлення вони будуть об'єднані в один тип.

Рисунок 3.6 – Подання, при якому враховують пріоритет завдань

Також завдання можуть бути різних типів і вимагати для виконання типи ресурсів, що не перетинаються, як це показано на рисунку 3.7.

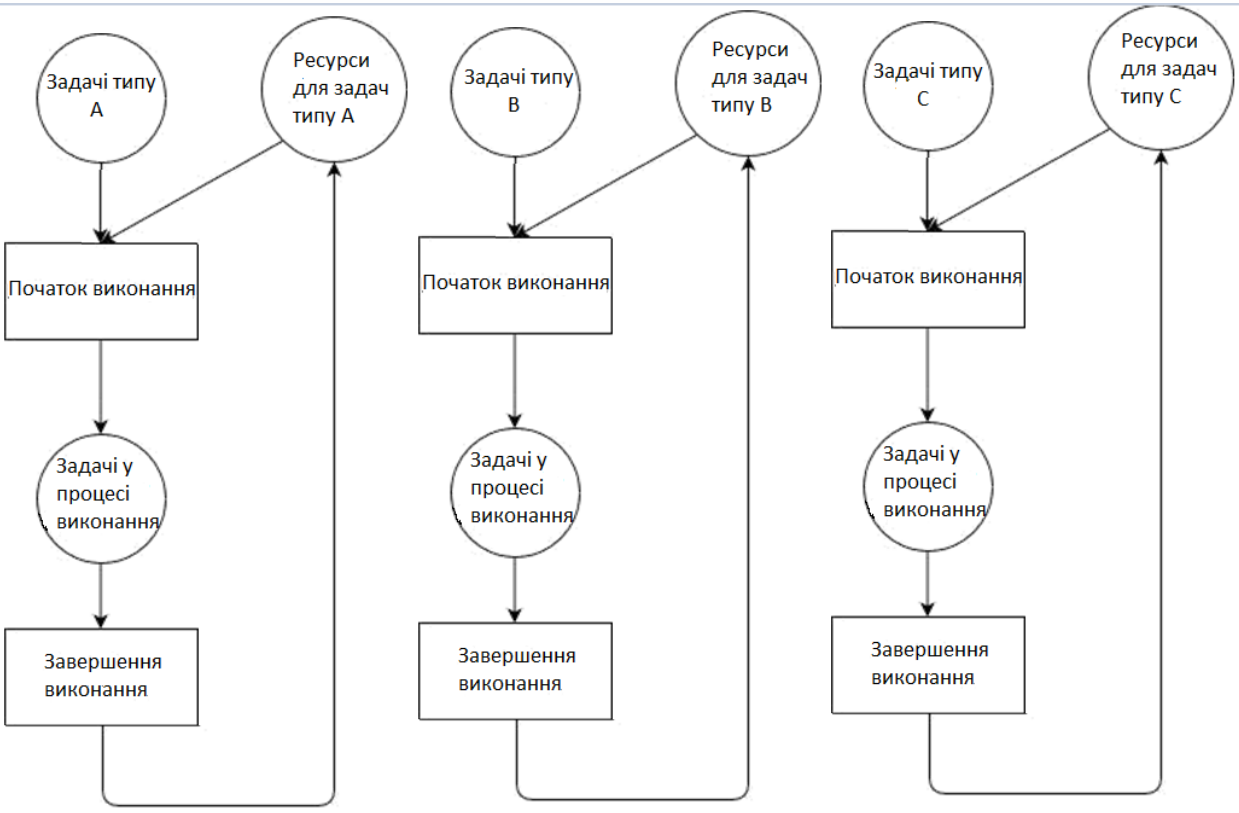

Рисунок 3.7 – Подання з різними типами задач

Розглянемо просте уявлення з ресурсами. У даному випадку не враховується що:

- ресурси можуть входити та виходити з обчислювальної мережі;
- ресурс виділений для виконання завдання, може вийти з ладу в час виконання завдання;
- можливість додавання нових та виходу старих ресурсів з обчислювальної мережі;
- перехід ресурсів у сплячий режим через надлишок;
- перехід ресурсів зі сплячого режиму в активний при їх нестачі.

Всі ці нюанси враховуються на мережі Петрі, представленої на рисунку 3.8. Пріоритетні завдання та різні типи завдань не були представлені щоб зберегти читання схеми.

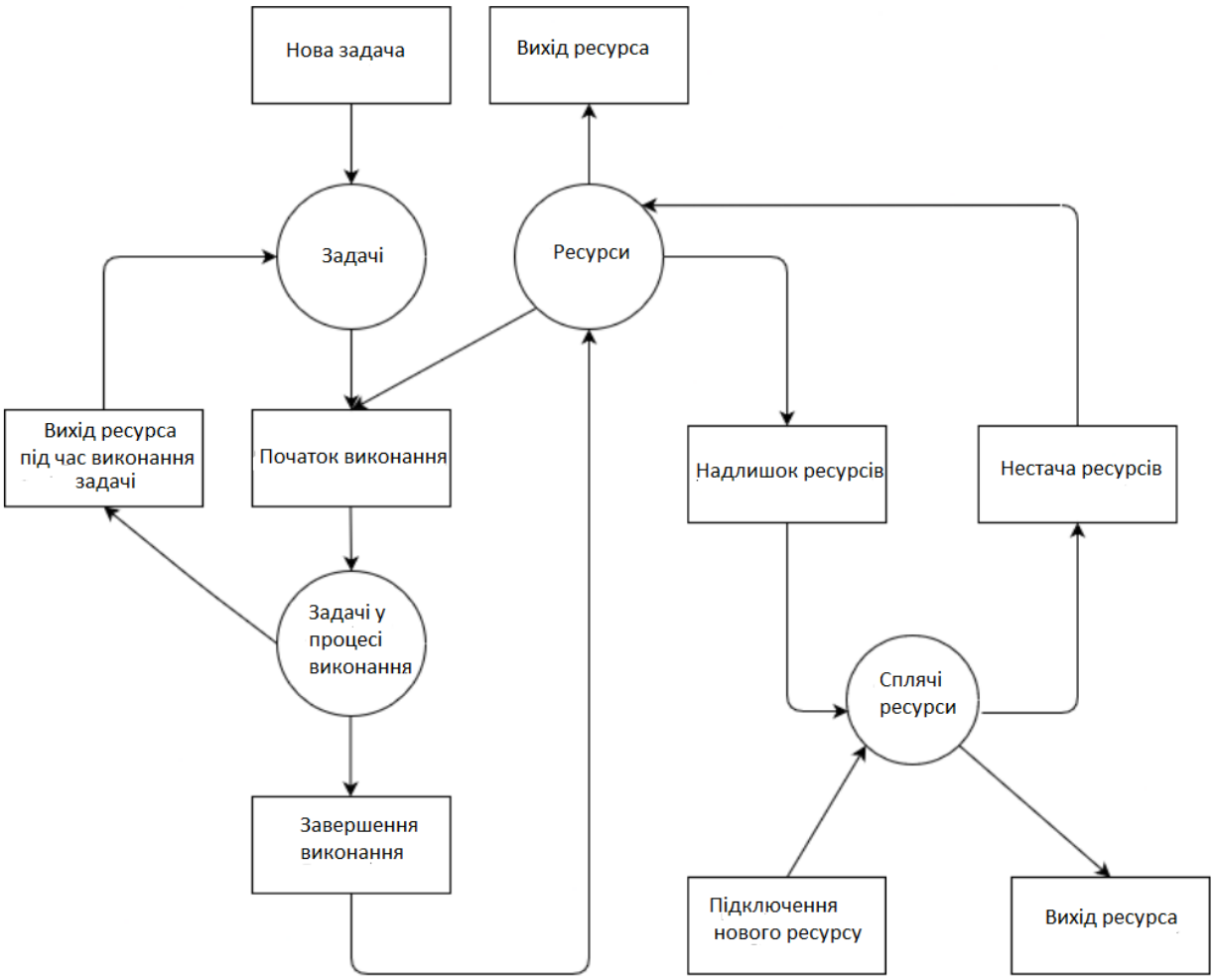

Рисунок 3.8 – Покращене подання виконання задач

3.3 Результати імітаційного моделювання

Для реалізації еластичних обчислень було вирішено розробити програму на мові python з використанням бібліотеки flask, для організації еластичних обчислень в межах однієї NAT мережі. Для перевірки роботи в NAT мережу були підключені 2 пристрої (ноутбук та телефон), після чого їм на виконання почали подаватися завдання. З кодом програми можно ознайомитися в додатку до кваліфікаційної роботи.

Були отримані дані виконання та з них побудовані графіки кількості невиконаних завдань, кількості вузлів мережі та кількість активних вузлів у мережі протягом 10 хвилин (рисунки3.9 – 3.11).

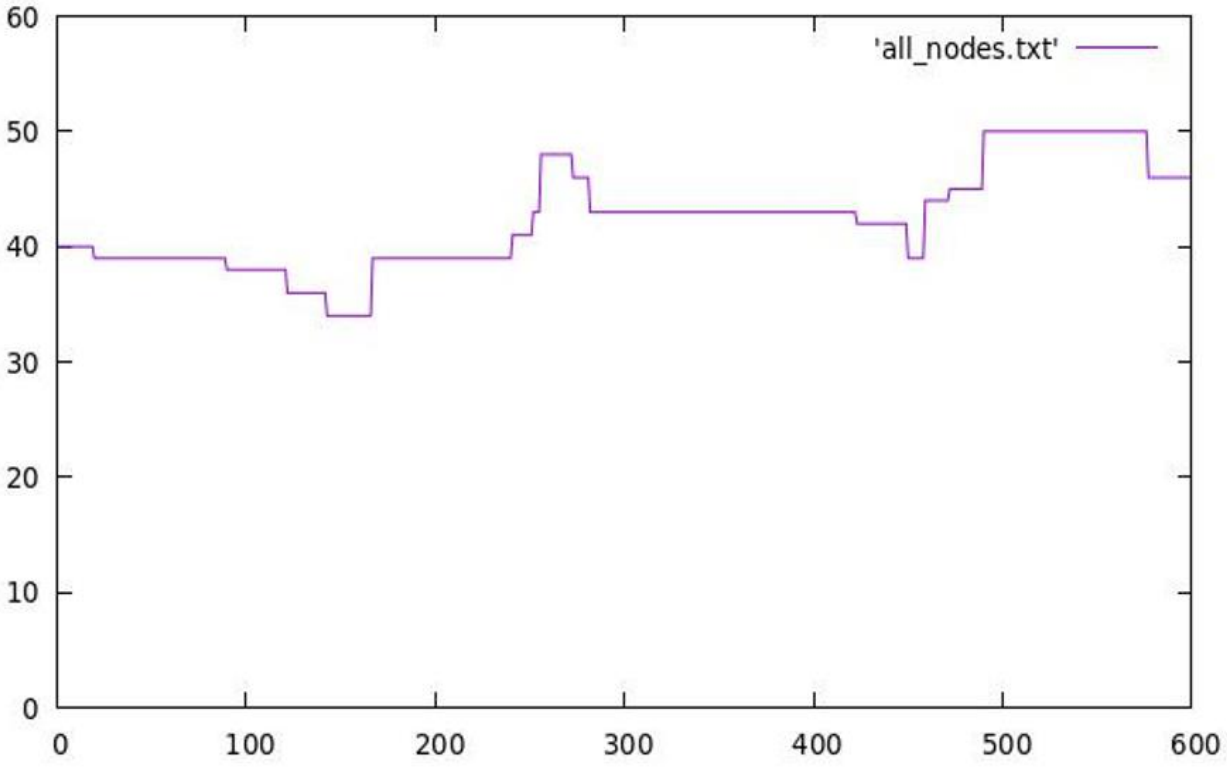

Рисунок 3.9 – Кількість вузлів у мережі

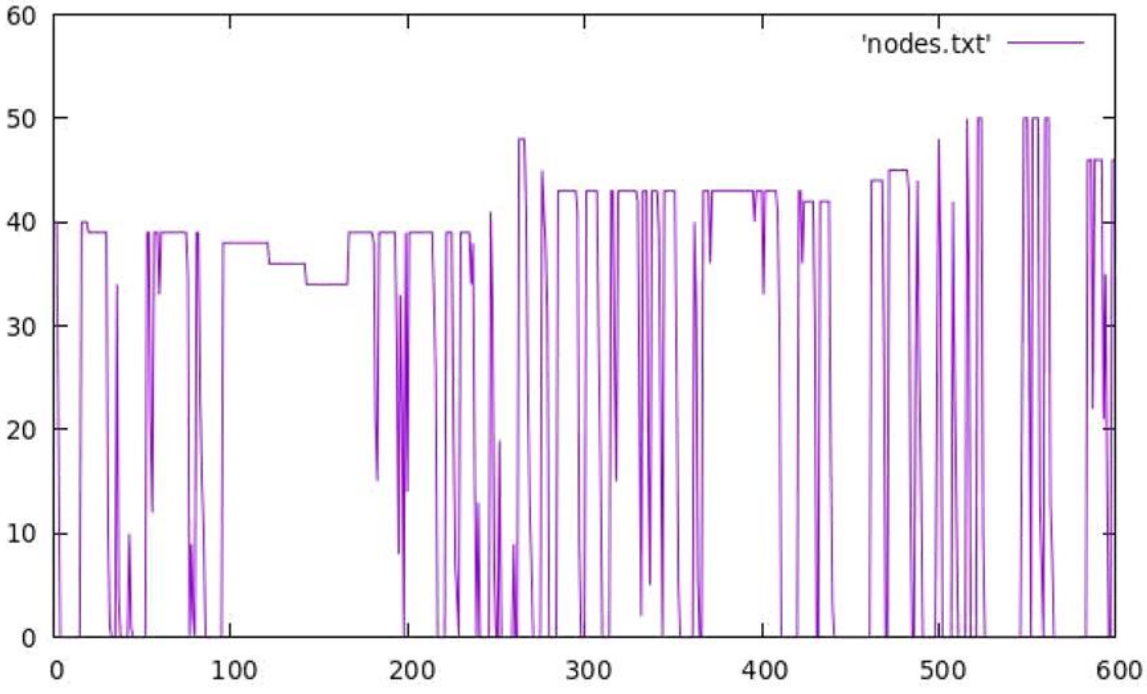

Рисунок 3.10 – Кількість активних вузлів

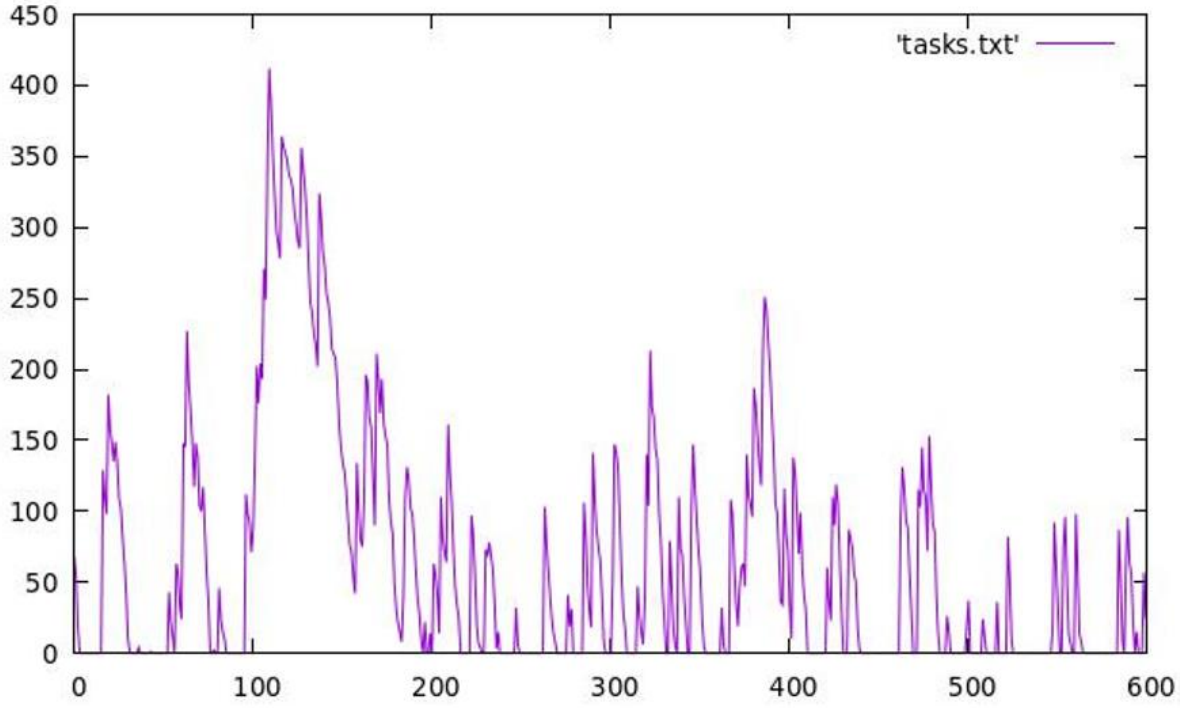

Рисунок 3.11 – Кількість невиконаних завдань

Кожен пристрій має чергу завдань, які можуть додаватися нові завдання, які будуть асинхронно виконані процесами на пристрої. Реалізовані різні типи завдань, які можуть виконуватися тільки відповідними процесами на вузлах.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто методи організації еластичних обчислень з використанням інтернету речей та була обрана модель туманних обчислень. Була написана програма на мові python з використанням вебфреймворку flask, що надає користувачеві платформу (PaaS), яка може об'єднувати кілька пристроїв у єдину обчислювальну мережу, з подальшим додаванням та виходом з неї вузлів (ресурсів). На цій платформі користувач може запустити свою програму, виконання якого автоматично розподіляється між доступними вузлами мережі, і у разі, якщо вузол відключився з мережі під час виконання, перекладає завдання на інший вузол мережі.

Були зроблені виміри продуктивності та побудований графік використання ресурсів, на якому видно, що ресурси використовуються оптимально, тобто ресурси виходять зі сплячого режиму, коли вже працюючих ресурсів не вистачає для виконання завдань.

У еластичних обчислень досить великий потенціал в майбутньому, бо на ринок виводиться дедалі більше нових та складних програм. І якщо організувати еластичні обчислення не лише в рамках одної системи, а й об'єднати кілька різних систем у єдину мережу, то можна буде отримати значно більшу обчислювальну потужність, ніж є на будь-якій окремо взятій системі. Потрібно зауважити, що обчислювальні можливості в хмарі будуть вищими, але завдяки локальності (близькій розташованості вузлів один від одного), така мережа матиме значно меншу затримку перед початком виконання завдання, дозволяючи їй обробляти завдання набагато швидше, що особливо важливо у випадках, які потребують швидкої відповіді. В хмару можна буде надсилати вже оброблені дані або особливо складні завдання, які не можна вирішити у рамках туманної мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Настиченко Т.А., Кучук Н.Г. Використання еластичних обчислень в туманних технологіях. Проблеми інформатизації. Тези доповідей 11 міжнародної науково-технічної конференції, Т.1, Баку – Бельсько-Бяла – Харків, 15-16 листопада 2023.– С.73.

2. Burger, A., Cichiwskyj, C., Schmeißer, S., & Schiele, G. (2020). The Elastic Internet of Things - A platform for self-integrating and selfadaptive IoT-systems with support for embedded adaptive hardware. Future Generation Computer Systems.

3. Juarez, F., Ejarque, J., & Badia, R. M. (2018). Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Future Generation Computer Systems, 78, 257–271.

4. Питерсон Дж. Теория сетей Петри и моделирование систем / Дж. Питерсон; пер. с англ. под ред. В.А. Горбатова – М.: Мир, 1984. – 264 с.

5. De Coninck, E., Verbelen, T., Vankeirsbilck, B., Bohez, S., Simoens, P., & Dhoedt, B. (2016). Dynamic auto-scaling and scheduling of deadline constrained service workloads on IaaS clouds. Journal of Systems and Software, 118, 101–114.

6. A. Itzkovitz and A. Schuster. Distributed shared memory: Bridging the granularity gap, in Proceedings of the First ACM Workshop on Software Distributed Shared Memory (WSDSM, 1999.)

7. A. Snavely and D. M. Tullsen. Symbiotic job scheduling for a simultaneous multithreading processor. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 234–244, Nov. 2020.

8. A.V. Bogdanov, M. Dmitriev, Ye Myint Naing, Eucalyptus Open-source Private Cloud Infrastructure, GRID 2010, Proceedings of the 4th International Conference Dubna, June 28- July 3, 2010. Page: 57-63.

9. A.V.Bogdanov, A.A. Lazarev, La Min Htut, Myo Tun Tun, Building User Access System in Grid Environment, Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 4th Intern. Conference, Dubna, 2019, Pages: 63-69.

10. Youpoung P, Muenchaisri P. Fault reparation time estimation based on exponential distribution function and software instability metric In: 6th International Conference on Electronics Information and Emergency Communication (ICEIEC),; Beijing. 2016; pp. 18-24. http://dx.doi.org/10.1109/ICEIEC.2016.7589678

11. Galkin AM, Simonina OA, Yanovsky GG. Multi-service IP Network QoS Parameters Estimation in Presence of Self-similar Traffic.Next Generation Teletraffic and Wired/Wireless Advanced Networking NEW2AN 2006. Lecture Notes in Computer ScienceBerlin, Heidelberg: Springer 2006; 4003. http://dx.doi.org/10.1007/11759355_23

12. Norros I. storage model with self-similar input. Queueing Syst 1994; 16(3-4): 387. http://dx.doi.org/10.1007/BF01158964

13. Pleskanka N. Algorithms for obzhgovovuvannyacherng in bloodless places", M.I.Kirik, NMPleskanka // Zbirniknaukovyhprac, - Institute of problems modulyuvannya in energetics. G.P.Pukhova. Kiev. Vypusk 2014; 70: 159-62.

14. Sonia Ben Rejeb. ZièdChoukair, and Sami Tabbane"The CAC Model and QoS Management in Wireless Multi-service Network"(IJCSE). Int J Comput Sci Eng 2010; 02(02): 333-9.

15. Roger H, et al. Network management system using model-based intelligence. U.S. Patent No. 5,504,921, 2022 April; 02