ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«ВИКОРИСТАННЯ ПЕРИФЕРІЙНИХ ОБЧИСЛЕНЬ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА НАДІЙНОСТІ КОМП'ЮТЕРНИХ МЕРЕЖ»**

на здобуття освітнього ступеня магістр
за спеціальності 123 Комп'ютерна інженерія
(код,найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олексій ЗАМКОВИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр.КСДМ-62
Олексій ЗАМКОВИЙ
(ім'я, ПРІЗВИЩЕ)

Керівник: В'ячеслав ЧЕРЕВИК
к.т.н., доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти «Магістр»

Спеціальність 123 Комп'ютерна інженерія
Освітньо-професійна програма Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерної інженерії
Наталія ЛАЩЕВСЬКА
(ім'я, ПРІЗВИЩЕ)
"____" _____2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Замковому Олексію Юрійовичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Використання периферійних обчислень для підвищення продуктивності та надійності комп'ютерних мереж
керівник роботи В'ячеслав ЧЕРЕВИК к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "19" 10 2023 р. №145
2. Строк подання кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи:
3.1. Інтернет ресурси стосовно периферійних обчислень.
3.2. Інтернет ресурси стосовно граничних обчислень інтернету речей.
3.3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити):
4.1. Аналіз існуючих рішень.
4.2. Виконання граничних обчислень на наявних в мережі пристроях інтернету речей.

4.3. Універсальна програма виконання граничних обчислень на пристроях інтернету речей.

4.4. Тестування виконання граничних обчислень на безсистемних пристроях інтернету речей

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання "19" жовтня 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	**Примітка**
1.	Підбір технічної літератури	.2023р. .2023р.	Виконано
2.	Аналіз існуючих рішень	.2023р. .2023р.	Виконано
3.	Виконання граничних обчислень на наявних в мережі пристроях інтернету речей	.2023р. .2023р.	Виконано
4.	Універсальна програма виконання граничних обчислень на пристроях інтернету речей	.2023р. .2023р.	Виконано
5.	Тестування виконання граничних обчислень на безсистемних пристроях інтернету речей	.2023р. .2023р.	Виконано
6.	Оформлення роботи, висновки	.2023р. .2023р.	Виконано
7.	Розробка демонстраційного матеріалу, доповідь	.2023р. .2023р.	Виконано

Здобувач вищої освіти Олексій ЗАМКОВИЙ

(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи В'ячеслав ЧЕРЕВИК

(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступеня магістр: 81 стор., 30 рис., 25 джерел.

Мета роботи – підвищення ефективності системи Інтернету Речей шляхом зменшення часу передачі та обробки інформації на пристроях Інтернету Речей за рахунок реалізації граничних обчислень.

Об'єкт дослідження – периферійні обчислення.

Предмет дослідження – пристрої Інтернету Речей.

Короткий зміст роботи: В роботі розглянуто існуючі підходи до організації граничних обчислень для пристроїв Інтернету Речей, запропоновано підхід до організації граничних обчислень безпосередньо на пристроях Інтернету Речей для різних типів систем, що використовуються пристроями наявних в локальній мережі користувача. Розглянуто приклад реалізації універсальної програми виконання граничних обчислень мовою Python та представлені методичні рекомендації до вибору організації граничних обчислень для конкретної системи. Продемонстровано приклад реалізації універсальної програми виконання граничних обчислень на пристроях Інтернету Речей побудованих на базі плати NodeMCU та представлені результати тестування ефективності такої системи.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНІ МЕРЕЖІ, ГРАНИЧНІ ОБЧИСЛЕННЯ, ІНТЕРНЕТ РЕЧЕЙ, ПЕРИФЕРІЙНІ ОБЧИСЛЕННЯ, PYTHON, ARDUINO, C/C++

ABSTRACT

The text part of the qualification work for obtaining a master's degree: 81 pages, 30 figures, 25 sources.

The purpose of the work is increasing the efficiency of the Internet of Things system by reducing the time of information transmission and processing on Internet of Things devices due to the implementation of marginal calculations.

The object of research is peripheral computing.

The subject of research is Internet of Things devices.

Summary of the work: The paper examines the existing approaches to the organization of boundary computations for Internet of Things devices, proposes an approach to the organization of boundary computations directly on Internet of Things devices for various types of systems used by devices available in the user's local network. An example of the implementation of a universal program for performing limit calculations in the Python language is considered, and methodological recommendations for choosing the organization of limit calculations for a specific system are presented. An example of the implementation of a universal program for performing marginal calculations on Internet of Things devices built on the basis of the NodeMCU board is demonstrated, and the results of testing the effectiveness of such a system are presented.

KEY WORDS: COMPUTER NETWORKS, EDGE COMPUTING, INTERNET OF THINGS, PERIPHERAL COMPUTING, PYTHON, ARDUINO, C/C++

ЗМІСТ

ВСТУП

Щороку кількість користувачів глобальної мережі Інтернет росте. В той же час збільшується кількість пристроїв, що потребують підключення до Інтернету. Також з розвитком комп'ютерної графіки та сінематографа, збільшується трафік, користувачів Інтернету, які споживають мультимедійний контент.

Одними із найбільш суттєвих споживачів Інтернету є пристрої Інтернету Речей (IoT). Більшість таких пристроїв розповсюджуються на комерційній основі, проте також існує можливість для будь-якої людини з хоча б мінімальним технічним досвідом знайти в Інтернеті потрібний проект з відкритим кодом (OSP) і на основі нескладних інструкцій створити свою власну копію пристрою чи навіть цілу систему пристроїв.

Проблема виникає при сильній віддаленості користувачів від центральної обчислювальної інфраструктури провайдерів послуг, або від хмарного обчислювального центру для саморобних пристроїв. В такому випадку трафік користувачів може долати великі відстані, інколи цілі континенти і тим самим перевантажувати комунікаційні вузли інформацією, яка мала б обчислюватися ближче до користувачів.

Для вирішення проблеми сучасні розробники намагаються дотримуватися парадигми периферійних обчислень, тобто переносити частину обчислень як найближче до користувачів. Периферійні обчислення найбільш відомі як Edge Computing, або граничні обчислення. Проте через відносну новизну не всі користувачі і розробники сприймають цю парадигму позитивно. Адже при створенні пристроїв IoT з вбудованим граничним обчислювальним пристроєм, або при необхідності встановленні граничного обчислювального сервера у локальну мережу користувача, витрати на таку систему зростають. А у випадку

розташування граничних обчислювальних серверів на території постачальника послуг, зростають витрати на утримання додаткових дата центрів (ДЦ), які виконують ці граничні обчислення, і складність виходу на нові ринки, тим самим знову таки збільшуючи вартість такої системи і створюючи додаткові складності для користувачів на віддалених територіях.

Проте, незважаючи на підвищену вартість, побудова системи пристроїв IoT, що базуються на граничних обчисленнях має ряд значних переваг. По перше, безпека таких систем збільшується, адже у зловмисників з'являється менше можливостей перехопити дані по дорозі від користувача до ДЦ. Додатково, за рахунок зменшення відстані до ДЦ, зменшується час Інтернет відгуку, який забезпечує швидку реакцію пристроїв IoT, що може бути критичним у випадку самокерованих автомобілів, або систем безпеки. І, звичайно, зменшується навантаження на глобальні Інтернет шляхи, тим самим дозволивши зменшити темп побудови нових каналів, що має сприятливу дію на екологію та дозволяє зменшити вартість обслуговування.

В магістерській роботі розглянуто головні особливості побудови систем IoT за парадигмою граничних обчислень. Протестовано виконання граничних обчислень на двох малопотужних пристроях IoT. Розроблено та імплементовано універсальну систему виконання граничних обчислень мовою Python, що дозволяє виконувати обчислення на самостійно розроблених, або комерційних пристроях IoT на операційні системі Windows або Linux. Створено список рекомендацій по застосуванню граничних обчислень.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз існуючих рішень виконання граничних обчислень

В роботі розглянуто основні підходи до організації граничних обчислень на пристроях Інтернету Речей, виокремлено основні підходи та проведено їх детальний аналіз. Виведено переваги та недоліки розглянутих систем.

1.1.1 Виконання обчислень пристроями Інтернету Речей

Вибуховий ріст і збільшення обчислювальної потужності пристроїв Інтернету Речей (IoT) призвели до безпрецедентного зростання обсягів даних. І вони будуть продовжувати зростати, оскільки введення мереж 5G дає значний поштовх до збільшення кількість підключених пристроїв. Раніше ідея хмари та штучного інтелекту полягала в автоматизації та пришвидшенні інновацій, шляхом отримання ефективного розуміння даних. Але безпрецедентний масштаб та складність даних, що передаються пристроями IoT, випереджають можливості мережі та інфраструктури.

Надсилання всіх даних, створених пристроєм, до централізованого центру обробки даних (ДЦ) або до хмари спричиняє проблеми з пропускною здатністю та затримкою передачі даних. Граничні обчислення пропонують більш ефективну альтернативу — обробка та аналіз даних переноситься ближче до місця, де вони створені. Оскільки дані не повинні пересилатися через мережу до ДЦ, затримка значно зменшується. Граничні обчислення дозволяють швидко та всебічно аналізувати дані, створюючи можливість для глибшого розуміння, меншого часу відгуку та покращеного досвіду роботи для клієнтів.

Від підключених транспортних засобів до інтелектуальних ботів на заводському поверсі, кількість даних від пристроїв IoT, що генеруються в нашому світі значно вища ніж будь-коли раніше, проте більшість цих даних взагалі не використовуються, або використовується не повністю. Наприклад, дослідження McKinsey & Company показало, що офшорна нафтова бурова установка генерує дані більш ніж з 30000 датчиків — але менше одного відсотка цих даних потрібне для прийняття рішень. Граничні обчислювальні технології використовують додаткові обчислювальні пристрої для аналізу даних наближеного до реального часу. Ця розширена здатність аналітики в сучасних пристроях може сприяти підвищенню швидкодії та безпеки пристроїв IoT.

Існує декілька підходів до виконання хмарних та граничних обчислень в пристроях IoT, до основних належать:

- традиційні хмарні обчислення — виконання обчислень даних, що генерують пристрої IoT, на віддаленому ДЦ. Має найнижчу вартість і складність для розробників та виробників пристроїв IoT, проте забезпечує найгірші часові показники;

- граничні обчислення на спільній території — виконання обчислень даних на граничному сервері, що є спільним для декількох користувачів послуг, які є незалежними один від одного. Має відносно вищу вартість і складність для розробників та виробників пристроїв IoT, але забезпечує пришвидшену передачу та обробку даних;

- граничні обчислення на території кінцевого користувача — виконання обчислень даних на граничному сервері, що знаходиться безпосередньо в локальній мережі користувача. Має значно вищу вартість для кінцевого користувача і відносно вищу складність масштабування, але забезпечує найвищу захищеність і швидкість передачі даних;

- граничні обчислення безпосередньо на пристрої Інтернету Речей — виконання обчислень даних на граничному пристрої, що вбудовується

безпосередньо в пристрій IoT. Має найвищу вартість, оскільки потребує додаткових фінансових вкладень для кожного пристрою IoT, але забезпечує повну захищеність і майже миттєву швидкість передачі даних;

- гібридна схема виконання граничних обчислень — комбінація підходів 1-4, що забезпечує найвищі показники продуктивності. Потребує значних фінансових вкладень і має високу складність розробки та підтримки, але забезпечує високу модульність системи за рахунок комбінації граничних серверів різного рівня.

1.1.2 Традиційні хмарні обчислення

Традиційно задля економії складні обчислення переносяться з пристроїв Інтернету Речей (IoT) на хмару виробника. Це дозволяє використовувати менш потужну апаратну складову окремих пристроїв компенсуючи це потужним сервером, що обслуговує безліч користувачів “за запитом”, тобто працює без простоювання при цьому забезпечуючи низький час відгуку для кінцевого споживача. Це мало сенс, так як користувачам більшості пристроїв IoT не потрібно постійно користуватися обчислювальними потужностями, і якщо б пристрої IoT комплектувались більш потужною обчислювальною системою більшість часу вона б не використовувалась. Але з ростом масштабу IoT збільшується трафік цих одиничних пристроїв тим самим перевантажуваючи глобальні вузли зв’язку.

Типова структура традиційних хмарних обчислень наведена на рисунку 1.1. За цією схемою, всі обчислення виконуються в дата центрі (ДЦ), що знаходиться в місті С. Для користувачів з міста С це не створює проблеми, але користувачі, що знаходяться в містах А та В вимушені пересилати дані до міста С. При цьому вони не тільки значно підвищують трафік між цими містами, спричинюючи збільшення часу мережевого відгуку для інших жителів міста, але й завдаються додаткового ризику,

що їх дані можуть перехопити на шляху до ДЦ і в їхньому місті і в місті С.

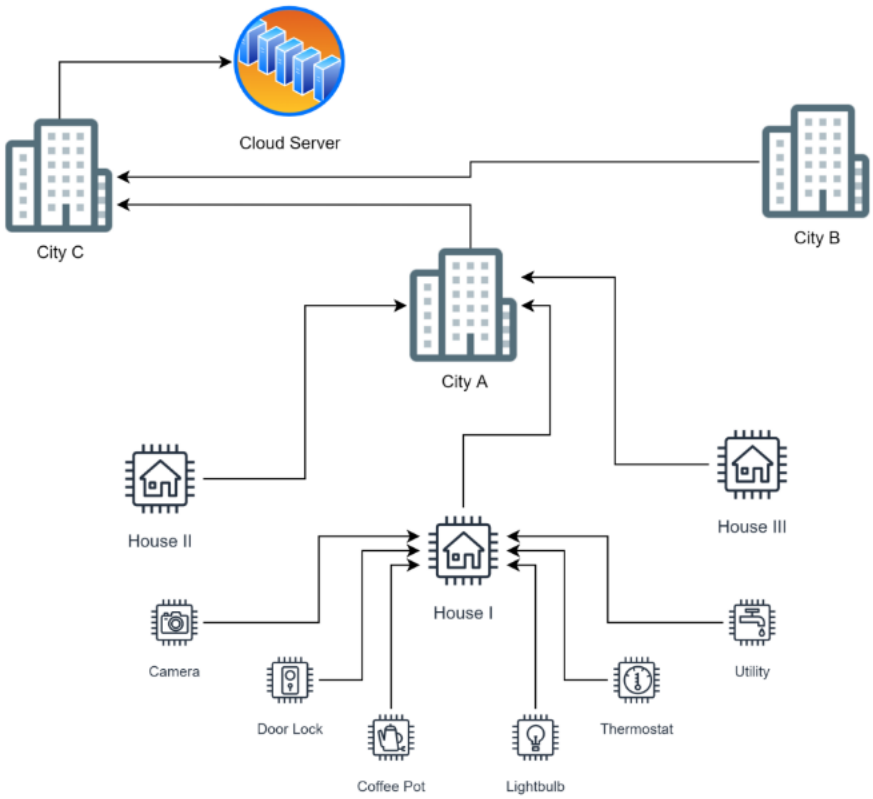

Рисунок 1.1 – Типова структура традиційних хмарних обчислень

З першого погляду така схема легко піддається масштабуванню, адже при збільшені кількості користувачів, провайдер хмарних послуг може просто збільшити обчислювальні потужності дата центру без необхідності проведення додаткових операцій таких як побудова нових дата центрів. Проте це не вирішує проблеми перевантаження транспортного рівня мережі і за цією моделлю ця проблема тимчасово вирішується тільки шляхом збільшенням кількості транспортних каналів,

що веде до суттєвих фінансових втрат і збільшення вуглецевого сліду мережі передачі даних.

Звичайно, постачальники хмарних послуг може мати декілька місць розташування своїх дата центрів і запити клієнтів будуть направлені до найближчого з них. Наприклад, у випадку, показаному на рисунку 1.1, дата центр знаходиться у місті С і запити клієнтів що поступають з цього міста долають відносно невелику відстань. Проте навіть у цьому випадку, запити клієнтів міста С можуть мати збільшену затримку під час пересилки в мережі через трафік, що надходить з міст А та В.

До цієї схеми відноситься використання послуг хмарних сервісів, як, наприклад, Amazon Web Services, що пропонує набір сервісів, які використовуються напряму пристроями IoT (рис. 1.2).

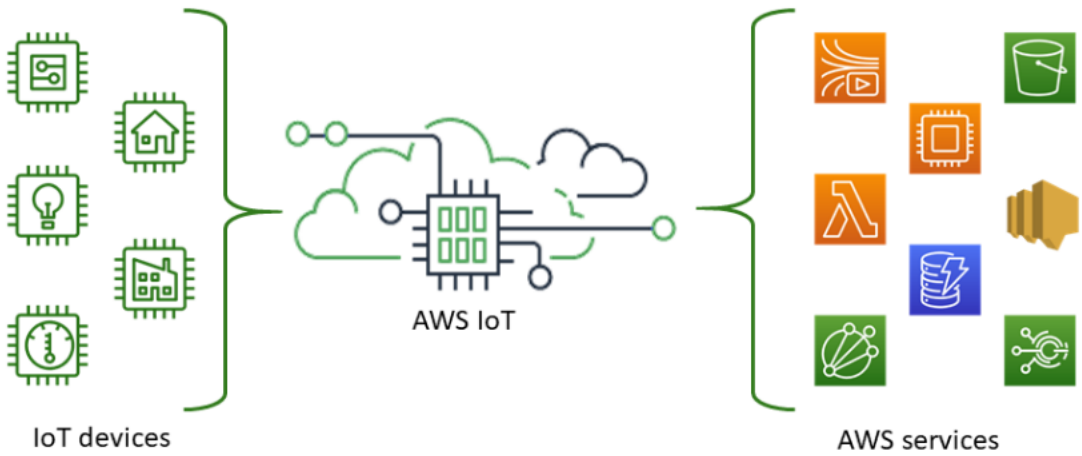

Рисунок 1.2 – Структура Amazon Web Services для пристроїв IoT

Amazon Web Services для пристроїв IoT (AWS IoT) дозволяють обрати найбільш зручні та сучасні технології для більшості завдань з якими стикаються розробники. Для допомоги в контролі та моніторингу

пристроїв IoT у використані, ядро AWS IoT підтримує наступні протоколи:

- MQTT – Черга повідомлень і передача телеметрії (Message Queuing and Telemetry Transport);

- MQTT over WSS – MQTT з використанням захищених веб-сокетів (Message Queuing and Telemetry Transport over Web Socket Secure);

- HTTPS – Захищений протокол передачі гіпертексту (Hypertext Transfer Protocol Secure);

- LoRaWAN – Широкосмугова мережа далекої дії (Long Range Wide Area Network).

Менеджер повідомлень ядра AWS IoT підтримує пристрої та клієнтів, що використовують протоколи MQTT і MQTT over WSS для прийняття та відправки повідомлень. Також підтримуються пристрої та клієнти, що використовують протокол HTTPS для відправки повідомлень. Ядро AWS IoT для LoRaWAN дозволяє підключати та керувати безпровідними пристроями LoRaWAN і заміняє потребу в розробці та підтримці мережевого серверу LoRaWAN. Хмарні сервіси AWS включають розподілене, широкомасштабне зберігання даних і обчислювальні сервіси, що підключені до Інтернету. До них входять:

- сервіси підключення та керування пристроями IoT;

- обчислювальні сервіси Amazon Elastic Compute Cloud і AWS Lambda;

- сервіси баз даних Amazon DynamoDB.

Пристрої IoT можуть отримувати хмарні послуги AWS за допомогою наступних технологій:

- Wi-Fi та Широкосмуговий Інтернет;

- Широкосмугові стільникові дані;

- Вузькосмугові стільникові дані;

- LoRaWAN.

До переваг традиційної структури хмарних обчислень відносять:

- низька початкова вартість;

- легкість імплементації.

Недоліки традиційної структури хмарних обчислень включають:

- вразливість даних до перехоплення під час передачі від пристрою клієнта до дата-центру;
- висока вартість масштабування системи;
- неефективне використання мережевих ресурсів.

1.1.3 Граничні обчислення на спільній території

Для зменшення часу мережевого відгуку, деякі постачальники послуг хмарних обчислень будують додаткові дата центри у місцях скупчення великої кількості клієнтів. Це наближає обчислення та зберігання даних до пристроїв, що його потребують, а не змушує покладатися на централізований дата центр, що може бути розташований за тисячі кілометрів. Це спричинено тим, що дані, особливо ті, що збираються в режимі реального часу, можуть страждати від проблем із затримкою створеною довгими шляхами передачі, які можуть вплинути на продуктивність програми. Крім того, за великої кількості пристроїв клієнтів в наближеному географічному положенні, компанії можуть заощадити гроші здійснюючи обробку локально, зменшуючи обсяг даних, що відправляється в централізований дата центр.

На рисунку 1.3 подано приклад структури, за якою постачальник послуг створює граничні обчислювальні центри в містах A, B та C. Таким чином, обчислення даних клієнтів відбувається безпосередньо в місті їхнього розташування, а на головний ДЦ в місті C відправляється лише ті дані, обробка яких не можлива на граничному сервері. При цьому виконання частини обчислень локально значно зменшує трафік між цими містами, спричинюючи зменшення часу мережевого відгуку не тільки для даних, що обробляються на граничному сервері, але й для даних, що відправляються на централізований ДЦ. Також, це зменшує кількість можливих місць перехоплення даних зловмисниками, оскільки більша

частина даних не покидають міста в якому знаходиться пристрій клієнта.

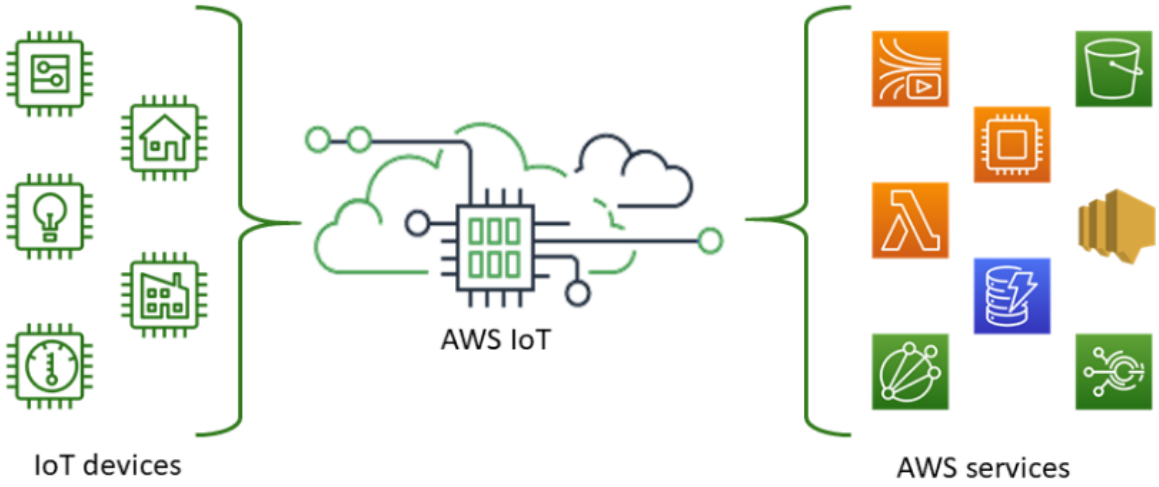

Рисунок 1.3 – Типова структура граничних обчислень на спільному граничному сервері

Така схема гарно піддається масштабуванню, оскільки при збільшені кількості користувачів, провайдер хмарних послуг може побудувати додаткові граничні обчислювальні центри на базі географічного положення користувачів, або просто збільшити обчислювальні потужності існуючих центрів. Проте це не повністю вирішує проблему перевантаження транспортного рівня мережі і вразливості даних до перехоплення даних зловмисником під час їх транспортування, але у випадку великої кількості граничних ДЦ, ступінь цих проблем може бути незначною.

Деякі постачальники хмарних послуг, пропонують вже готові апаратні рішення для виконання таких граничних обчислень, як, наприклад, AWS Outposts, що мають готові програмні рішення для інтеграції граничного серверу у систему пристроїв Інтернету Речей (рис.

1.4).

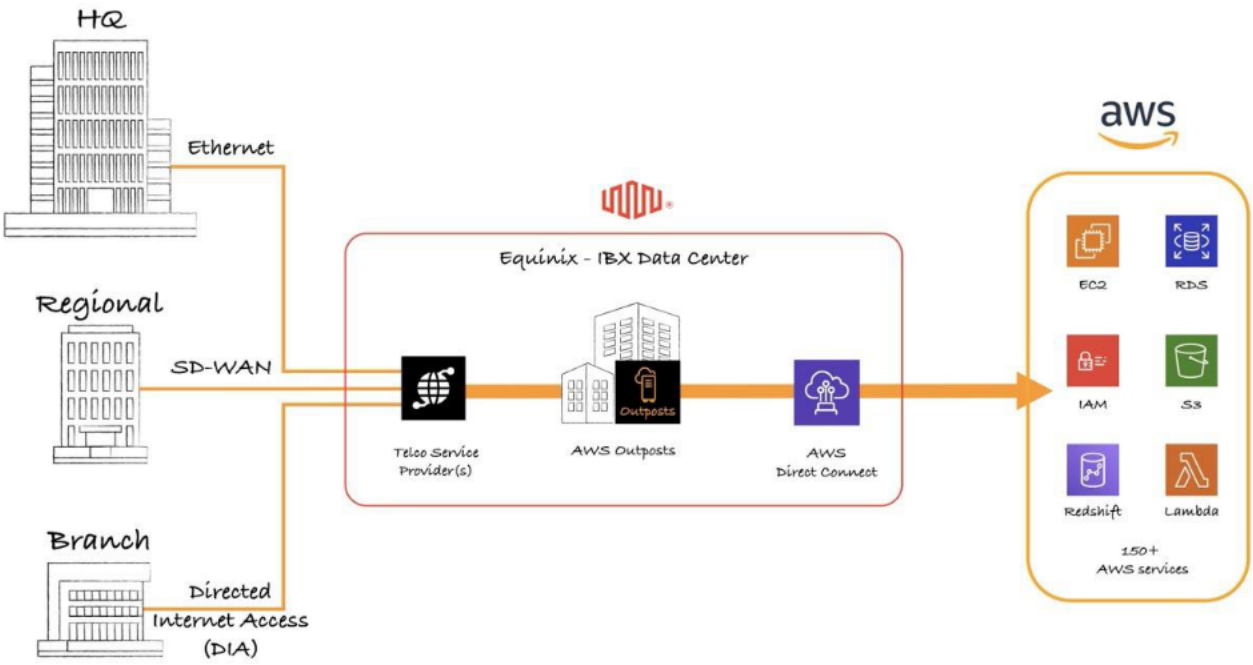

Рисунок 1.4 – Структура Amazon Web Services Outposts

AWS Outposts це повноцінний сервіс, що забезпечує таку саму AWS інфраструктуру, AWS сервіси, програмний інтерфейс керування і інструменти для будь якого місця розташування граничних серверів. Вони доступні як 42U системи, що можуть розширюватися від 1 до 96 стійок для створення пулів обчислення та зберігання даних.

До переваг граничних обчислень на спільному граничному сервері відносять:

- стала вартість масштабування системи;
- контроль роботи системи розробниками системи;

- легкість підключення нових клієнтів.

Недоліки граничних обчислень на спільному граничному сервері включають:

- вразливість даних до перехоплення під час передачі від пристрою клієнта до граничного ДЦ;

- початкова складність розробки;

- неефективне використання мережевих ресурсів.

1.1.4 Граничні обчислення на території кінцевого користувача

За наявності великої кількості пристроїв IoT, або потреби в обробці значних обсягів даних, деяким клієнтам може бути доцільно встановити локальний пристрій виконання граничних обчислень безпосередньо на своїй території. Це значно зменшує обсяг даних, що покидає локальну мережу клієнта, тим самим збільшуючи безпеку конфіденційних даних та значно зменшує мережевий відгук для пристроїв IoT, але при цьому значно збільшує ціну такої системи.

На рисунку 1.5 наведено приклад структури, за якою клієнт встановлює граничний обчислювальний пристрій безпосередньо в локальній мережі. Таким чином, обчислення даних клієнту відбувається безпосередньо на території клієнтів, відправляючи на головний ДЦ в місті С лише ті дані, обробка яких не можлива, або ускладнена на локальному граничному сервері.

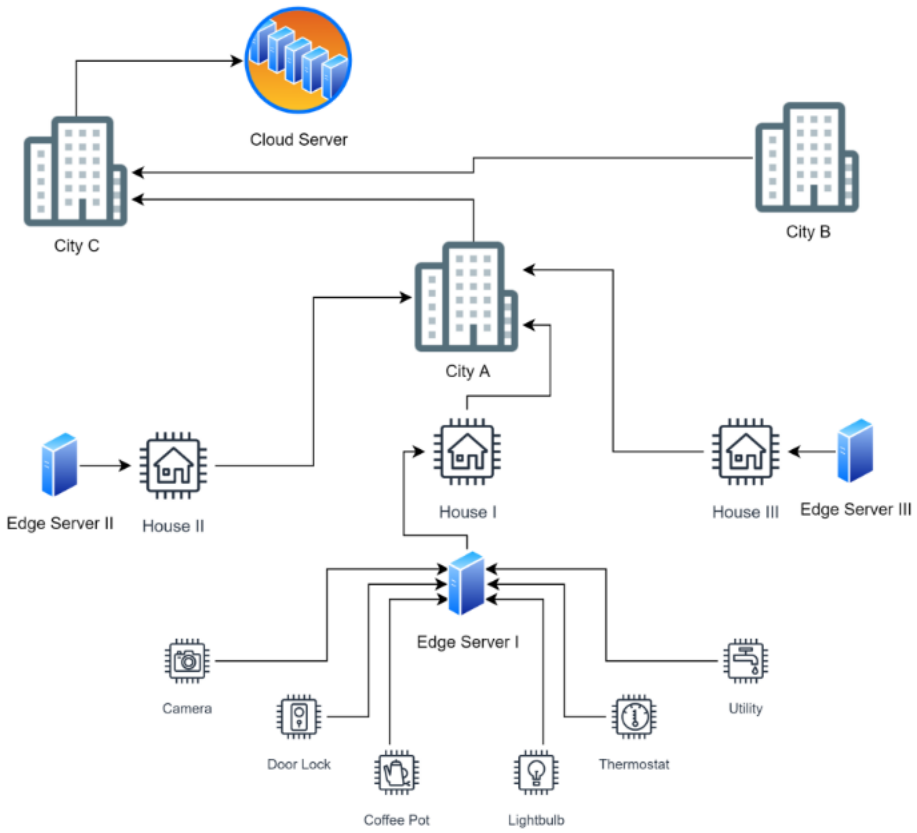

Рисунок 1.5 – Типова структура граничних обчислень на території кінцевого користувача

Виконання частини обчислень локально значно зменшує трафік, що проходить через глобальну мережу, спричинюючи зменшення часу мережевого відгуку для всіх користувачів мережі. Додатково, це зводить до мінімуму ризик перехоплення даних зловмисниками, оскільки більша частина даних не покидають локальної мережі в якій знаходиться пристрій клієнта.

Прикладом постачальників послуг, що надають можливість

локалізувати граничні обчислення є Google Cloud Edge, що пропонує розробникам програмні засоби організації граничних обчислень безпосередньо в локальній мережі користувача, які наведено на рисунку 1.6.

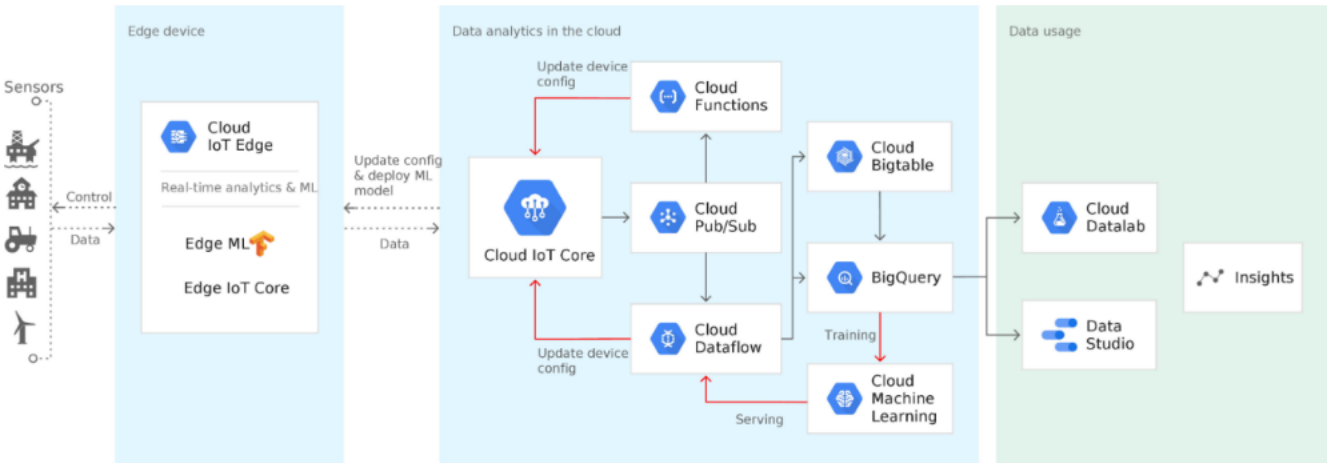

Рисунок 1.6 – Структура Google Cloud Edge для пристроїв IoT

Google Cloud Edge надає набір програмних засобів, які можуть бути використані для організації граничних обчислень на машині користувача, а не у централізованому ДЦ. Ці програмні засоби дозволяють перетворити робочу станцію користувача, що працює на базі Linux, на локальний граничний сервер. Вони складаються з трьох основних компонентів:

- Edge Connect — програмний засіб для надійного підключення граничних пристроїв до хмари та для програмних і мікропрограмних оновлень;

- Edge ML — програмний засіб для виведення моделей машинного навчання моделей TensorFlow Lite;

- Edge TPU — апаратний засіб, що розширює основні обчислювальні можливості системи для більш ефективного виконання моделей

TensorFlow Lite на граничному сервері.

До переваг граничних обчислень на території отримувача послуг відносять:

- конфіденційність даних користувача;
- ефективне використання глобальних мережевих ресурсів;
- часткова незалежність клієнтів.

Недоліки граничних обчислень на території отримувача послуг включають:

- висока вартість для кінцевого користувача;
- складність розробки та підтримки системи.

1.1.5 Граничні обчислення безпосередньо на кінцевому пристрої Інтернету Речей

Деякі пристрої IoT вимагають постійного зв'язку з обчислюваним сервером, або створюють величезні обсяги даних, передача яких перевантажує канали передачі локальної мережі. У цьому випадку вводиться так званий граничний пристрій, що входить безпосередньо до складу пристрою IoT і виконує граничні обчислення без передачі даних мережею. Таким чином, система досягає максимально короткого часу відгука та максимального захисту даних і при цьому значно зменшує обсяг даних, що передаються як локальною так і глобальною мережею, проте при цьому значно збільшуючи її вартість.

На рисунку 1.7 зображено приклад структури, за якою клієнт встановлює граничний обчислюючий пристрій безпосередньо в пристрої IoT. Таким чином, граничне обчислення даних відбувається безпосередньо на пристроях які це потребують найбільше, при цьому дані, що рідко використовуються, але не є важливими для користувача, відправляються на головний ДЦ в місті С. Не зважаючи на те, що деякі з пристроїв все ще відправляють запити на централізований ДЦ, через незначні обсяги цих даних, все рівно значно зменшується трафік, що

проходить через локальну та глобальну мережу, спричинюючи зменшення часу мережевого відгуку для всіх користувачів мережі і повна його мінімізація для пристроїв IoT в яких встановлені граничні обчислювальні пристрої. Додатково, це мінімізує ризик перехоплення даних зловмисниками, оскільки важливі дані не покидають пристрою клієнта.

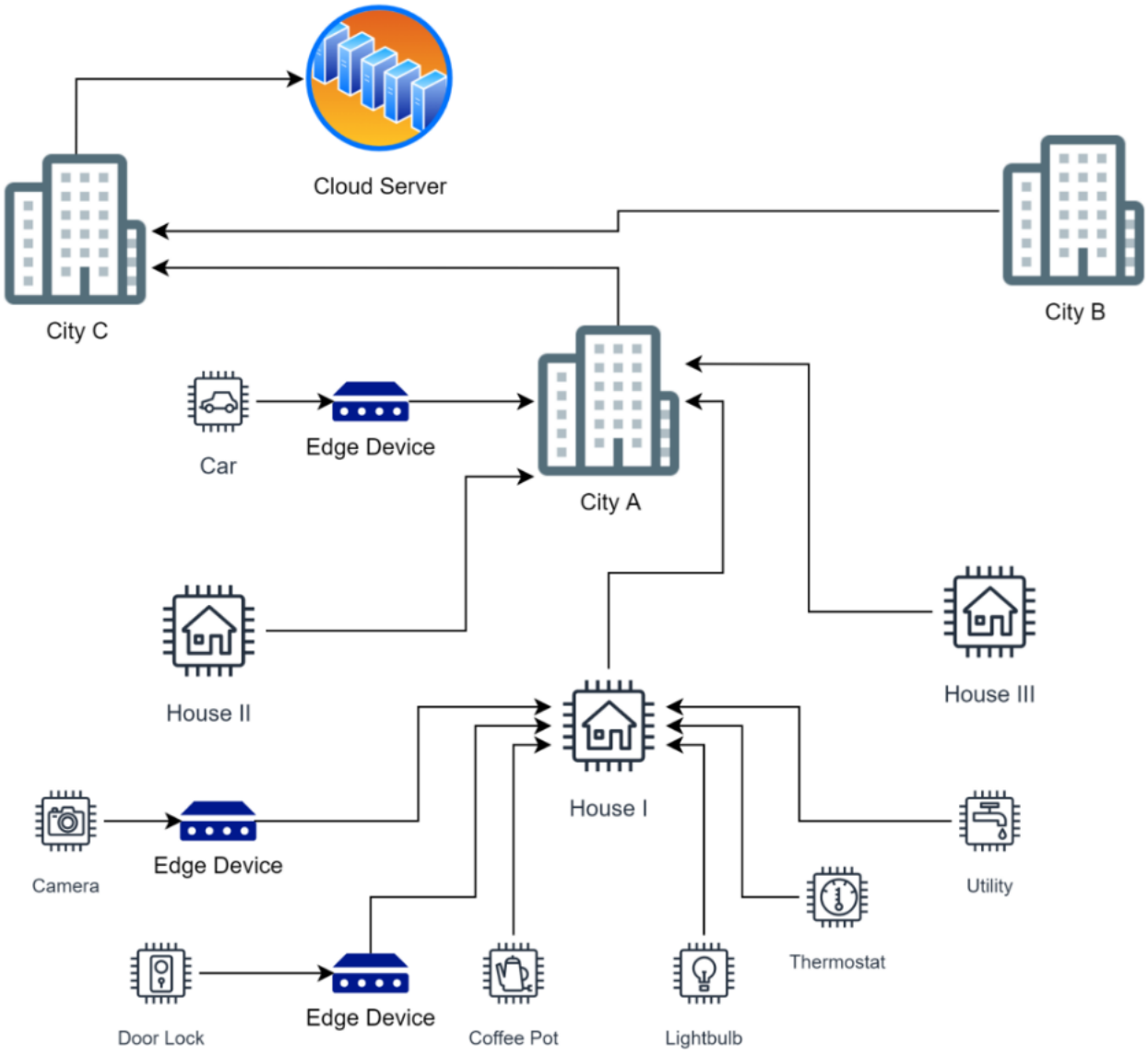

Рисунок 1.7 – Типова структура граничних обчислень на пристрою отримувача послуг

Така система може бути побудована на базі AWS IoT Greengrass, а апаратною частиною граничного пристрою може слугувати плата Nvidia Jetson. AWS IoT Greengrass – це IoT сервіс з відкритим кодом який допомагає створювати, розгортати та керувати програмним забезпеченням граничного обчислювального пристрою. Він використовується на мільйонах пристроїв у будинках, на заводах, у транспортних засобах та на підприємствах, обчислюючи локально дані, які вони генерують, виконуючи прогнози на основі моделей машинного навчання (ML), фільтруючі та агрегуючі дані пристрою та передаючи лише необхідну інформацію в хмару.

AWS IoT Greengrass дозволяє швидко та легко створювати інтелектуальне програмне забезпечення для пристроїв IoT. Цей сервіс забезпечує локальну обробку, обмін повідомленнями, управління даними, ML та пропонує заздалегідь побудовані компоненти для прискорення розробки додатків. AWS IoT Greengrass також забезпечує безпечний спосіб безперебійного підключення граничних пристроїв до будь-якої служби AWS, або до сторонніх служб. Після завершення розробки програмного забезпечення AWS IoT Greengrass дозволяє віддалено керувати програмним забезпеченням, не потребуючи оновлення мікропрограми.

До переваг граничних обчислень на пристрої отримувача послуг відносять:

- висока конфіденційність даних користувача;
- ефективне використання мережевих ресурсів;
- незалежність роботи пристроїв від перебоїв зв'язку з мережею.

Недоліки граничних обчислень на пристрої отримувача послуг включають:

- висока вартість;
- неефективне використання ресурсів граничного пристрою.

1.1.6 Гібридна схема виконання граничних обчислень

Найбільш продуктивною є схема виконання граничних обчислень, що об'єднує засоби наведенні в пунктах 1.1.2-1.1.5, проте така система має високу вартість і підвищену складність розробки, оскільки алгоритм роботи системи повинен передбачати ефективне розподілення обчислень на різних етапах граничних обчислень. На рисунку 1.8 наведений приклад такої структури.

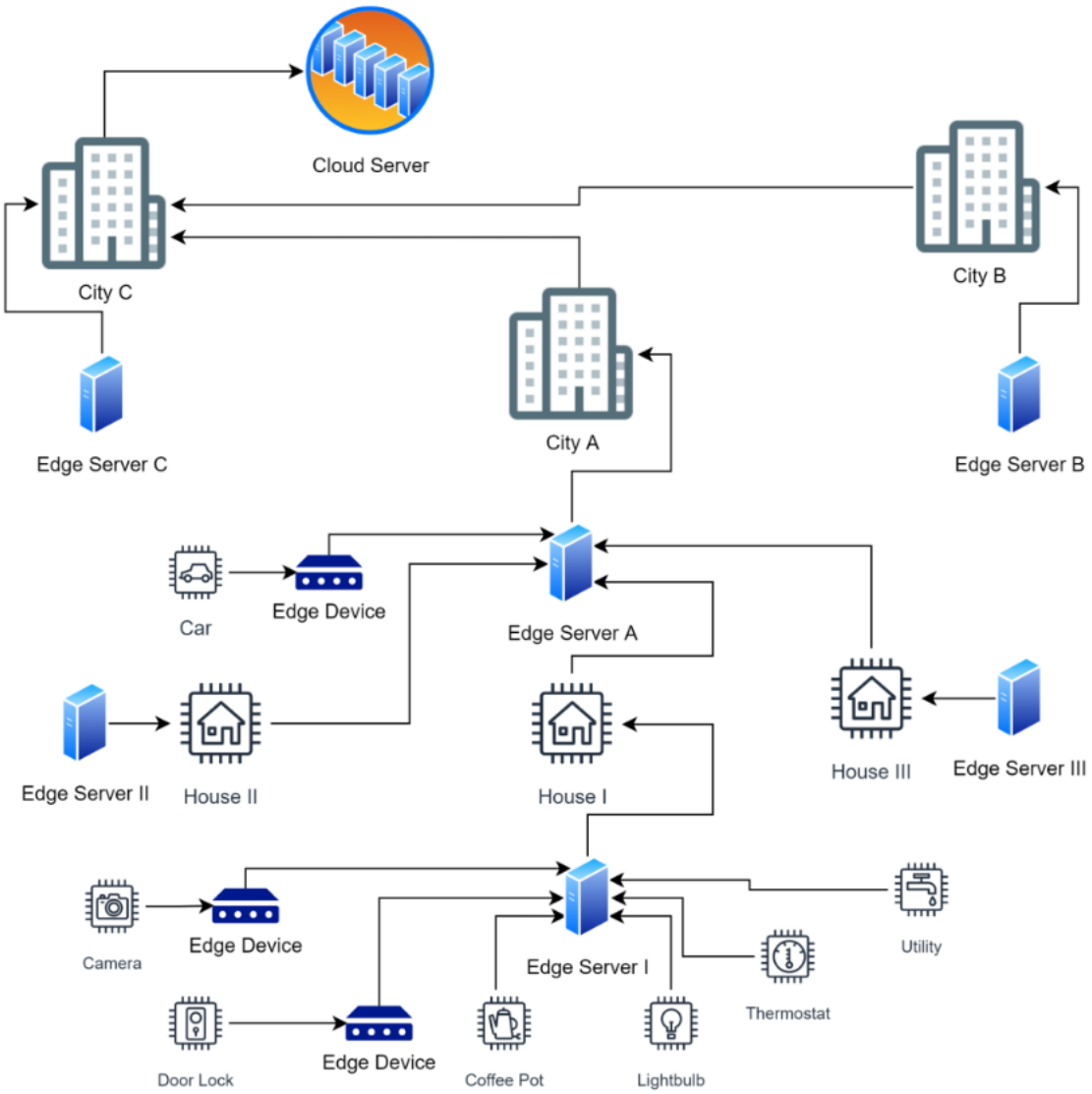

Рисунок 1.8 – Гібридна структура виконання граничних обчислень

Сценарій використання ресурсів передбачає введення граничних пристроїв для найбільших споживачів обчислювальних ресурсів та пристроїв чутливих до перебоїв мережі. Після цього вводяться граничні пристрої в локальні мережі, які виконують обчислення задач більш низького навантаження і пріоритетності. Далі вводяться граничні пристрої на спільній території для виконання задач найнижчого пріоритету та обсягу даних. Така система має гнучку структуру і може адаптуватися до змін стану мережі, проте потребує значних фінансових та часових вкладень.

1.2 Аналіз існуючих сімейств пристроїв Інтернету Речей

В роботі розглянуто основні підходи до створення пристроїв Інтернету Речей, виокремлено основні моделі їх роботи та проведено детальний аналіз особливостей реалізації цих моделей.

1.2.1 Пристрої Інтернету Речей

Інтернет Речей (IoT) — це новітня парадигма, що передбачає використання зв’язку між електронними пристроями та датчиками через Інтернет для полегшення життя людей. IoT використовує інтелектуальні пристрої та Інтернет для надання інноваційних рішень проблем та питань, пов’язаних з різними бізнесами, урядовими та приватними установами по всьому світу. IoT поступово стає важливим аспектом нашого життя, який поступово змінює світ навколо нас. В цілому, це інновація, яка об’єднує велику кількість розумних систем, інтелектуальних пристроїв та датчиків (рис. 1.9). Більше того, це дозволяє використовувати квантові та нано

технології для досягнення швидкості зберігання, зондування та обробки, які раніше не можна було уявити. Також були проведені масштабні наукові дослідження, щоб продемонструвати потенційну ефективність та придатність перетворень світу завдяки IoT. Це дає поштовх до створення нових інноваційних бізнес- планів, при цьому забезпечуючи безпеку та сумісність.

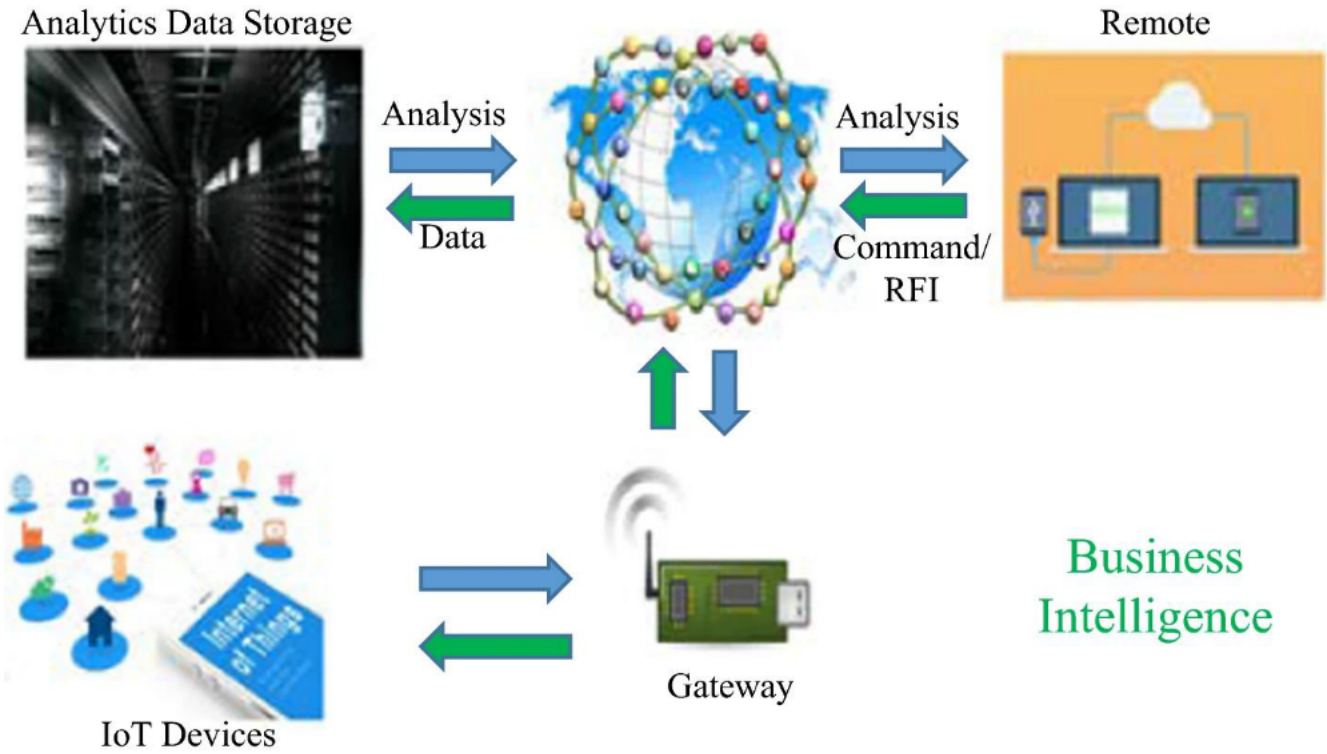

Рисунок 1.9 – Загальна архітектура IoT

IoT пристрої та технології вже давно проникають в життя людей. Одним з результатів розвитку IoT є концепція розумних домашніх систем

та побутових приладів, що складаються з Інтернет-пристроїв, систем автоматизації будинків та системи надійного управління енергією. Крім того, ще одним важливим досягненням IoT є система Smart Health Sensing. Вона складається з невеликого інтелектуального обладнання та пристроїв для підтримки та моніторингу стану здоров’я людей. Ці пристрої можна використовувати як у приміщенні, так і на відкритому повітрі для перевірки та моніторингу різних проблем зі здоров'ям та рівня фізичної підготовки, або кількості спалених калорій у фітнес-центрі тощо. Крім того, вони використовуються для моніторингу критичних станів здоров'я в лікарнях та травматологічних центрах. Таким чином це змінює основу медичної сфери, полегшивши роботу лікарів і зменшивши кількість неочікованих проблем зі здоров'ям за допомогою високих технологій та інтелектуальних пристроїв. Більше того, дослідники IoT активно досліджують способи підвищення якості життя інвалідів та людей старшого віку. IoT демонструє кардинальну ефективність у цій галузі та надає можливість забезпечення нормального життя таких людей. Оскільки ці пристрої та обладнання є економічно вигідними з точки зору вартості розробки та легко доступні в межах звичайного цінового діапазону, більшість людей мають до них доступ. Ще одним важливим аспектом життя людей є транспорт. IoT надає нові можливості та підходи, щоб зробити його більш ефективним, комфортним та надійним. Пристрої IoT вже зараз контролюють рух на різних сигналізованих перехрестях у великих містах у межах систем Розумного Міста. Крім того, на ринки випускаються транспортні засоби із заздалегідь встановленими системами, які здатні розпізнати майбутні затори на карті та можуть запропонувати інший маршрут із низьким рівнем заторів. Таким чином, IoT має незлічимі сценарії застосування у різноманітних аспектах життя та технології, як з точки зору вдосконалення існуючих технологій, так і створенню абсолютно нових.

IoT також показав своє значення та потенціал в економічному та

промисловому зростанні регіонів, що розвиваються. Крім того, на ринку торгівлі та фондових бірж це розглядається як революційний крок. Однак безпека даних під час пересилки є важливою проблемою та однією з головних питань для вирішення. Інтернет складається з безлічі незалежних вузлів, деякі з яких є під контролем або заражені шкідливим програмним забезпеченням зловмисниками і тим самим передаючі дані в мережі, завжди існує ризик втрати конфіденційності, або підміни даних під час транспортування. Однак IoT надає свободу у використані будь яких можливих рішень для вирішення питань безпеки даних та інформації, в тому числі згадані раніше граничні обчислення. Отже, найважливішою проблемою IoT у торгівлі та економіці є безпека, тому розробка безпечного шляху обробки даних є гарячою темою в галузі Інтернету Речей.

Всі пристрої IoT можна поділити на дві основні категорії:

- пристрої IoT, що працюють без операційної системи;
- пристрої IoT, що працюють на базі сучасних операційних систем.

Окремо варто виділити пристрої, що розробляються кінцевим виробником. Деякі компанії пропонують придбати плати та набори для розробки, які дозволяють легко створити свій власний пристрій IoT. Більшість таких плат, дозволяють розробнику запрограмувати пристрій як до першої, так і до другої категорії, проте більш дешеві плати, підтримують тільки безсистемну розробку через обмеження в ресурсах.

1.2.2 Сімейства пристроїв Інтернету Речей, що працюють без операційної системи

Пристрої Інтернету Речей, що працюють без операційної системи можна розділити на дві головні групи:

- безсистемні пристрої (baremetal);
- пристрої системи реального часу (RTOS).

Baremetal пристрої, це пристрої, що виконують одну програму. Іншими словами, безсистемний пристрій, це фізичний пристрій, повністю призначений для запуску одного спеціального додатка, який, наприклад, може бути «програмою управління термостатом». Це схоже за принципом роботи на перші комп'ютери, проте сучасні комп'ютери зазвичай при запуску завантажують операційну систему яка здатна завантажувати, керувати та розподіляти системні ресурси на інші програми. Це робиться таким чином, щоб програми, не заважали один одному, або щоб їхній вплив був зведений до мінімуму. Це добре працює для персональних комп'ютерів (ПК), оскільки користувачі очікують можливість запускати кілька програм одночасно. Проте, це не єдиний спосіб запускати програми, наприклад, програма термостата, яка вимірює температуру через датчик та включає нагрівач, коли температура падає під певною заданою температурою і надсилає користувачу повідомлення про досягнення бажаної температури, не потребує цього рівня складності. В цьому випадку використовується baremetal пристрої IoT. Це пов’язано з тим, що такий простий пристрій не виграє від накладних витрат, які можуть виникнути при використанні операційної системі (ОС) або RTOS, оскільки він може спокійно виконувати свою програму в одному потоці і не потребує функціоналу, який пропонують ці більш складні рішення.

Деякі виробники пропонують baremetal пристрої, які можуть бути вільно запрограмовані кінцевим користувачем, як, наприклад, плати Arduino. Ці плати можуть бути використані при створенні різноманітних пристроїв Інтернету Речей за допомогою безлічі модулів розширення та готових програмних проектів, які створює спільнота розробників.

Наступним етапом від baremetal пристроїв є пристрої під управлінням RTOS. Програмне забезпечення пристроїв RTOS має порівняно просту структуру, проте на відміну від безсистемних пристроїв, може одночасно запускати та зупиняти різні підпроцеси. Проте RTOS все ще ближче до безсистемних пристроїв ніж до ОС. Найбільш суттєвою

відмінністю є захист пам'яті та віртуалізація. В ОС високого класу пам'ять додатків відокремлюється, і ОС гарантує, що один процес (запущена програма) не може пошкодити пам'ять іншого процесу. Це робиться за допомогою віртуалізації пам'яті. Це дає можливість ОС міняти місцями невикористану пам’ять на диску, коли вона не використовується. Проте віртуалізація відсутня в RTOS. Цей рівень управління ресурсами дуже корисний при застосуванні в загальній системі, як ПК, де користувач не знає багато про схему використання програми. Однак на невеликих IoT пристроях це інакше. Записуючи програму на такий пристрій, користувач, яким зазвичай виступає розробник, зазвичай знає про:

- скільки програм та процесів буде запущено;
- який обсяг пам’яті потрібний цим програмам і процесам;
- ці процеси не заважають один одному.

Тому, працюючи з невеликим пристроєм IoT, розробники зазвичай вибирають RTOS. Це пов’язано з тим, що пристрої IoT повинні керувати стеком TCP/IP, що зазвичай вимагає безлічі переривань в системі. Щоб уникнути управління цими складностями самостійно, додатки, які раніше використовували безсистемно, оновлюються до RTOS або повноцінної ОС.

При цьому в деяких випадках, як, наприклад, NodeMCU розробнику надаються програмні засоби, що дозволяють керувати стеком TCP/IP без використання RTOS, що дозволяє заощадити пам'ять та ресурси на малопотужних пристроях Інтернету Речей.

Також, розробник може уникнути необхідності керування стеком TCP/IP в малопотужних датчиках та пристроях IoT, шляхом використання в мережі граничного пристрою-посередник, який буде виконувати мережеві дії за пристрій IoT, підключаючись до нього в будь який зручний спосіб (рис. 1.10), тим самим зменшуючи вартість кінцевої системи.

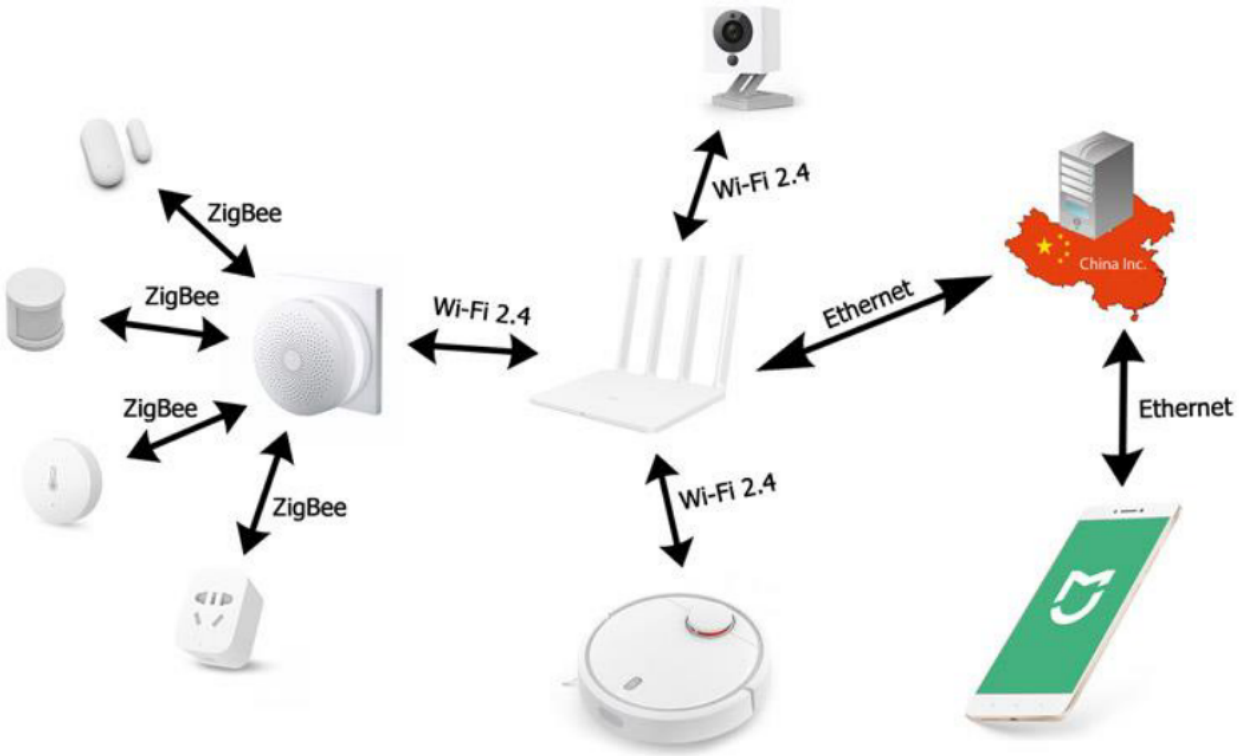

Рисунок 1.10 – Використання Xiaomi Gateway у якості посередника між пристроями IoT підключеними через ZigBee та мережею

1.2.3 Сімейства пристроїв Інтернету Речей, що працюють на базі сучасних операційних систем

Пристрої, що використовують Baremetal і RTOS мають деякі подібності, такі як зменшення накладних витрат, ефективне використання пам'яті, проте проектування пристроїв під керуванням ОС вимагає зовсім

іншого підходу. Традиційна операційна система інколи вважається занадто важкою для простих пристроїв IoT, однак ОС, такі як Linux або Windows, є необхідним варіантом для додатків IoT, які потребують розширених функціональних можливостей із потужними мікропроцесорами, великою пам'яттю та запуском складного програмного забезпечення. Однак більш доцільно використовувати пристрої з ОС для шлюзів IoT, серверів хмарних обчислень та граничних серверів, де існує необхідність обробляти великі потоки даних з різних вузлів або кінцевих пристроїв. Однією з досить відомих і широко використовуваних у спільноті IoT ОС є Linux – який часто є складовою частиною вбудованих систем. Одним з таких прикладів вбудованого Linux, який використовується для роботи над IoT, є Raspberry Pi. Raspberry Pi - це невелика плата, яка спочатку була розроблена для залучення дітей шкільного віку до розробки програмного забезпечення, проте тепер користується великою повагою у спільноті IoT і часто використовується у таких галузях, як промислова IoT. У вбудованому Linux програми запускаються або підтримуються ядром - головною частиною операційної системи, що контролює майже все в системі. Тому кожна взаємодія між апаратним та програмним забезпеченням проходить через ядро.

Вбудований Linux або інший тип операційної системи є найкращим варіантом для розробників, яким потрібна ОС, яка легко масштабується, вимагає менше кодування та має підтримку великої спільноти. Однак вищезгадане ядро, як правило, поширюється разом з великою кількістю допоміжного програмного забезпечення, яке поставляється попередньо встановленим і для розробника інколи важко зрозуміти, які програми працюють, яке їхнє завдання і чи можуть вони становити вразливість для безпеки системи, особливо, якщо вони використовуються або налаштовані неправильно. Наприклад, серед камер спостереження було виявлено багато вразливостей не в програмному забезпеченні, встановленому постачальником, а в додатковому програмному

забезпеченні ОС.

1.3 Особливості організації виконання граничних обчислень для пристроїв Інтернету Речей

Граничні обчислення можуть відбуватися, як на наближеному граничному сервері так і безпосередньо на пристрої IoT. Виконання обчислень на так званому граничному пристрої, що встановлюється безпосередньо в пристрій IoT досягає максимальної захищеності та швидкості обробки даних, проте вимагає значних фінансових витрат і при цьому не завжди використовує повний потенціал граничного пристрою, оскільки більшість задач не вимагає постійного обчислення. З іншої сторони граничний сервер забезпечує спільний для всіх пристроїв, що знаходяться в одній локальній мережі дозволяє більш ефективне використання обчислювальних можливостей системи, але в той же час створює не значну небезпеку перехоплення даних, за умови, що зловмисник зумів підключитися до локальної мережі системи, там може призводити до мінімальної затримки обробки даних, у випадку коли два, або більше, пристроїв IoT намагаються використати одні й ті самі ресурси. Граничний сервер, що використовується одночасно кількома різними системами показує найкращі показники з точки зору ефективності користування спільними ресурсами проте збільшує небезпеку перехоплення даних зловмисником і створює затримку передачі даних до граничного серверу, яка може бути значною у випадку сильної віддаленості системи клієнта від граничного дата центру.

Пристрої IoT можуть будуватися як на базі сучасної операційної системи, так і на базі без системних плат. Baremetal плати чудово підходить для розробників IoT, які хочуть мати повний контроль над

роботою пристрою IoT, мають невеликі або обмеженні завдання, а також мають потрібні знання для самостійного кодування та обслуговування стеку IoT. З іншого боку, RTOS є найкращим варіантом, якщо розробнику потрібна багатозадачність, або паралельна обробка даних. В той же час, системи під управлінням ОС є ідеальним рішенням, якщо пристрій IoT має запас обчислювальних ресурсів і може відмовитися від повного контроля над системними ресурсами в обмін на програмну інфраструктуру для побудови потужних програмних засобів, але при цьому має можливість детально вивчити набір попередньо встановлених програм для того, щоб захистити систему від вразливостей.

Поєднуючи різні типи пристроїв IoT, можна створити систему, яка буде виконувати граничні обчислення безпосередньо використовуючи вже існуючі апаратні ресурси, проте так система буде вимагати унікального підходу до кожного з розглянутих типів пристроїв Інтернету Речей.

Дипломний проект націлений на дослідження можливостей перенесення обчислювальних задач граничних серверів на інші пристрої Інтернету Речей, що знаходяться в одній комп'ютерній мережі. Запропонованим логічним кроком розвитку структури граничних обчислень, розглянутих в пунктах 1.1.2- 1.1.5, є побудови мережі граничних обчислень, яка використовує складові пристрої IoT для виконання розподілених обчислень. Проте через різноманіття типів апаратної та програмної складової пристроїв IoT, що було розглянуто в пунктах 1.2.1-1.2.3, неможливо обійтися одним програмним забезпеченням, оскільки безсистемні пристрої потребують написання унікального програмного забезпечення, що було розглянуто в пункті 1.2.2, тому вони потребують створення узагальненого керівництва для розробників, що будуть імплементувати систему граничних обчислень в програмне забезпечення цих пристроїв. Проте пристрої, що працюють під керівництвом OS, мають можливість виконувати універсальну програму,

оскільки підтримують багатозадачність і потребують менше уваги розробників до процесів низького рівня системи.

Проаналізовано пристрої Інтернету Речей і при цьому доведено, що граничні обчислення все ширше використовуються пристроями IoT для вирішення проблем надмірного трафіку та затримки виконання обчислень. Показано, що в останні роки значне збільшення кількості пристроїв Інтернету Речей призводить до надмірного навантаження на транспортні шляхи мережі Інтернет. Розглянуто різні моделі побудови систем виконання граничних обчислень і різні підходи до розробки програмного забезпечення пристроїв IoT.

Проведений аналіз показав, що залишаються невирішеними наступні задачі:

- використання вільних обчислювальних ресурсів пристроїв Інтернету Речей;

- мінімізація трафіку, що передається з пристроїв IoT локальної мережі;

- оптимізація процесу розробки і впровадження систем виконання граничних обчислень з використанням пристроїв IoT.

2 ВИКОНАННЯ ГРАНИЧНИХ ОБЧИСЛЕНЬ НА НАЯВНИХ В МЕРЕЖІ ПРИСТРОЯХ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Спосіб взаємодії пристроїв Інтернету Речей

Використовуючи наявні обчислювальні ресурси пристроїв IoT, можна створити мережу пристроїв, які будуть виконувати розподіленні граничні обчислення у вільний від основних задач час. Запропоновану система виконання граничних обчислень на мережі пристроях Інтернету Речей, можна реалізувати трьома способами:

- пристрої під'єднуються напряму. За цим підходом всі пристрої в мережі знають про один одного і завдання граничного обчислення розділяється на всі пристрої в мережі (рис 2.1). Недоліком цього підходу є те, що пристрої не знають про стан навантаження один одного і можливі випадки коли створюється затримка виконання граничного обчислення через один пристрій, що має спочатку виконати більш пріоритетну задачу.

За цією схемою система може підтримувати будь яку кількість пристроїв;

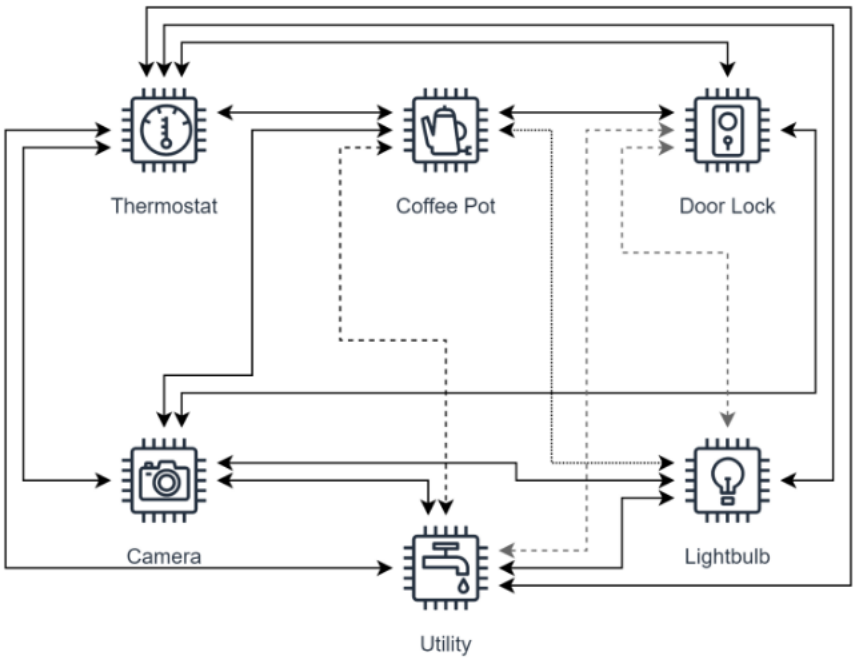

Рисунок 2.1 – Пряме підключення пристроїв IoT між собою

- всі пристрої під'єднуються до одного, який може в тому числі виступати шлюзом системи, що розподіляє завдання на інші пристрої системи. За цим підходом один пристрій системи регулярно моніторить стан пристроїв за допомогою службових запитів і розподіляє задачу з урахуванням навантаження пристроїв (рис 2.2). Відмінність цього метода від використання граничного серверу полягає в тому, що пристрій який вводиться в мережу є недостатньо потужним для самостійного виконання граничних обчислень, але в той же час дозволяє використати незадіяні апаратні та системні ресурси наявних в системі пристроїв IoT. Недоліком цього підходу є те, що система потребує встановлення додаткового пристрою, єдиною задачею, якого є розподілення завдань

граничного обчислення. При цьому, кількість пристроїв системи визначається потужністю додаткового пристрою, оскільки недостатня кількість пристроїв IoT в системі може зробити використання додаткового пристрою в якості граничного серверу більш ефективним, а завелика кількість пристроїв в системі може призвести до ситуації, коли більшість пристроїв IoT не задіяні до виконання граничних обчислень через обмеженість ресурсів додаткового пристрою;

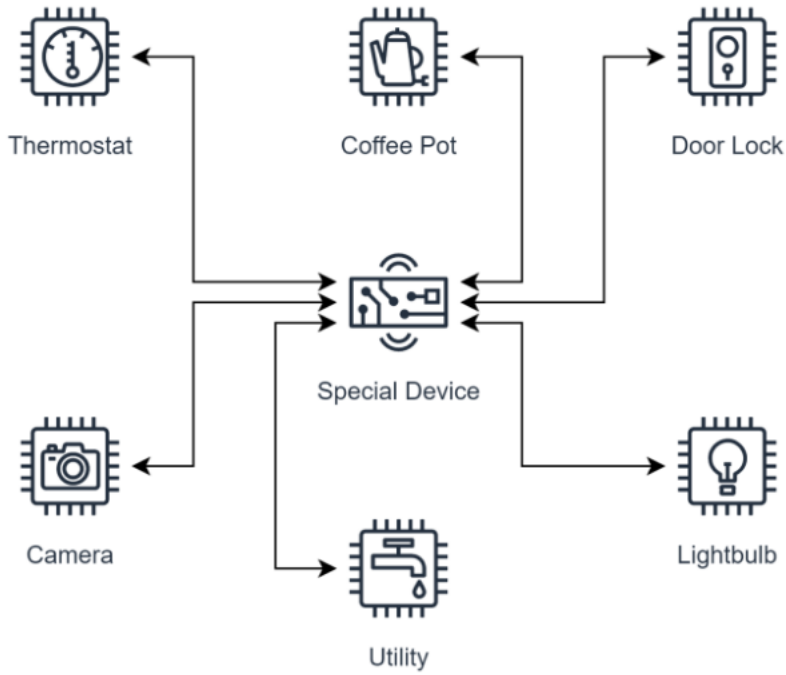

Рисунок 2.2 – Підключення пристроїв IoT через пристрій-посередник

- пристрої приєднуються напряму, але при цьому в систему вводиться пристрій, який регулярно моніторить стан мережі і перед початком створення завдання для граничних обчислень пристрої можуть дізнатися про стан мережі від цього пристрою (рис 2.3). Головним

недоліком такої системи є її надлишковість, оскільки пристрої мають створити додаткове з'єднання до пристрою, що моніторить стан мережі. У випадку, якщо ці функції виконує один з пристроїв IoT, можливі ситуації коли виконання обчислень затримується через навантаженість цього пристрою, у тому числі основними задачами. Тому, зазвичай задачі моніторингу передаються на потужній пристрій, який рідко використовується, або так само як і в другому підході вводиться новий пристрій для виконання цих функцій.

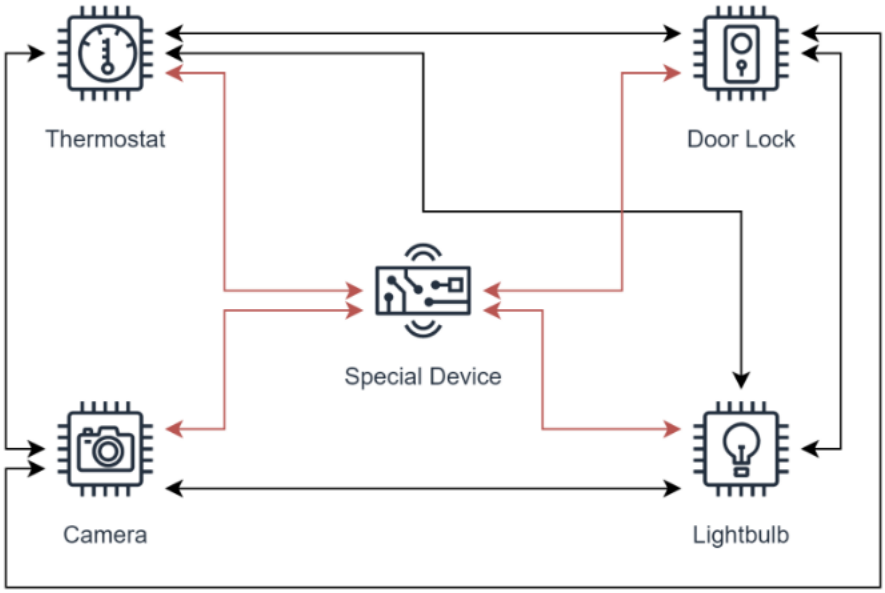

Рисунок 2.3 – Підключення пристроїв IoT напряму з використанням додаткового пристрою для контролю та моніторингу навантаження системи

У загальному випадку перший спосіб є найбільш універсальним і не потребує додаткових дій від кінцевого користувача. Тому для побудови системи виконання граничних обчислень на мережі пристроїв Інтернету

Речей, варто використати саме його, оскільки це дозволить вільно змінювати конфігурацію системи не вимагаючи додаткових дій від користувача, або розробника. Проте, потрібно передбачити незалежність основних пристроїв від зміни конфігурації мережі, оскільки другий та третій спосіб, дозволяють досягти більш швидку обробку даних для системи з наперед визначеною кількістю пристроїв IoT.

2.2 Алгоритм роботи системи

Програмне забезпечення (ПЗ) baremetal пристроїв IoT (рис. 2.4) не має вбудованих засобів багатозадачності і складається з функції роботи мікроконтролера, що знаходиться у нескінченному циклі та функції ініціалізації мікроконтролера, яка в деяких випадках може бути відсутня, або інтегрована в функцію роботи мікроконтролера. Тому важливо передбачити, такий алгоритм роботи мікроконтролера, що забезпечує стабільне виконання основних функцій мікроконтролера та мінімізує затримку виконання задач граничного обчислення, що надходять на пристрій IoT. Цього можна досягти двома способами:

- використати переривання для розподілу пріоритету задач мікроконтролера. При надходженні запиту на виконання граничного обчислення або тригерування виконання основних функцій пристрою, система обирає наступну задачу до виконання в залежності від пріоритету встановленого розробником, або перериває поточну задачу для виконання задачі більшого пріоритету. Для цього способу, необхідна апаратна підтримка переривань платою пристрою IoT та розробка додаткового ПЗ для керування пріоритетами задач. Перевагою цього способу є набагато вища швидкодія та надійність пристрою для виконання своїх основних задач та граничного обчислення;

- послідовне виконання основних задач пристрою IoT та граничного обчислення. У цьому випадку розробка додаткового ПЗ не потрібна, проте

час виконання кожної з задач повинен бути не великим, оскільки пристрій IoT не зможе виконувати задачі по мірі їх надходження і користувач повинен буде очікувати закінчення попереднього завдання.

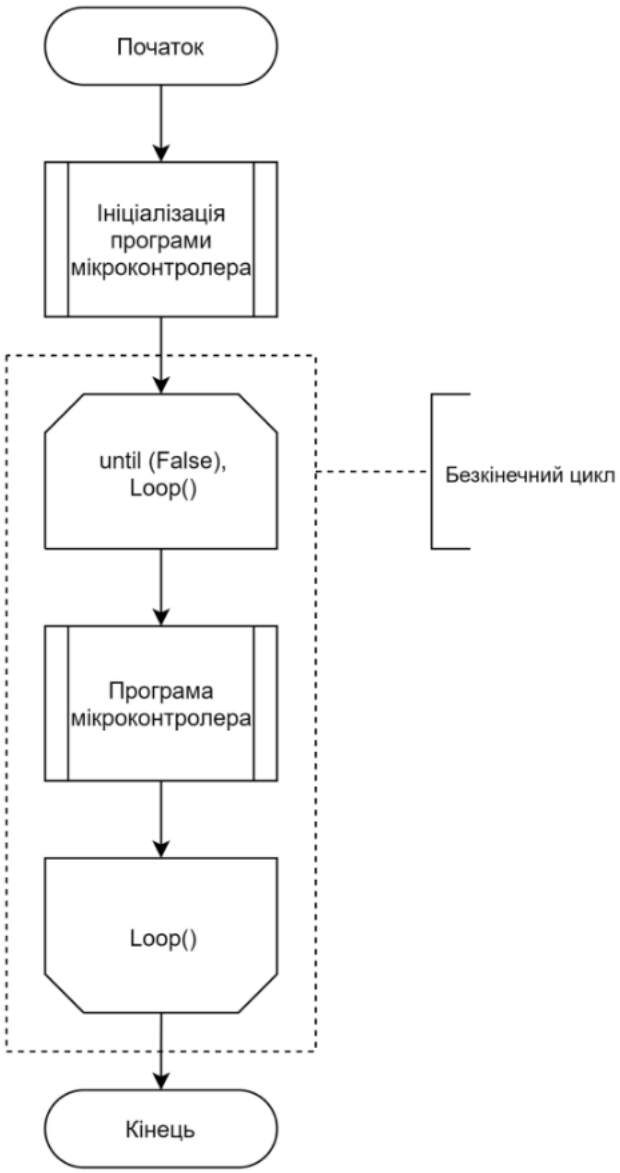

Рисунок 2.4 – Схема виконання програм baremetal пристроїв IoT на прикладі програми-скетчу Arduino

У випадку пристроїв IoT, що використовують RTOS пристрої мають вбудований в ядро планувальник завдань (рис. 2.5), що бере на себе задачі розподілення апаратних ресурсів пристрою і тому немає необхідності в синхронізації роботи модуля граничного обчислення з основною програмою. Тому програма, що використовується в такому випадку, розробляється без урахування роботи інших програм і таким чином, розробник може створити одну програму, яка буде виконуватися на пристроях IoT різного призначення. Проте програма все ще буде залежати від апаратної складової пристрою, оскільки різні плати можуть підтримувати відмінні інструкції.

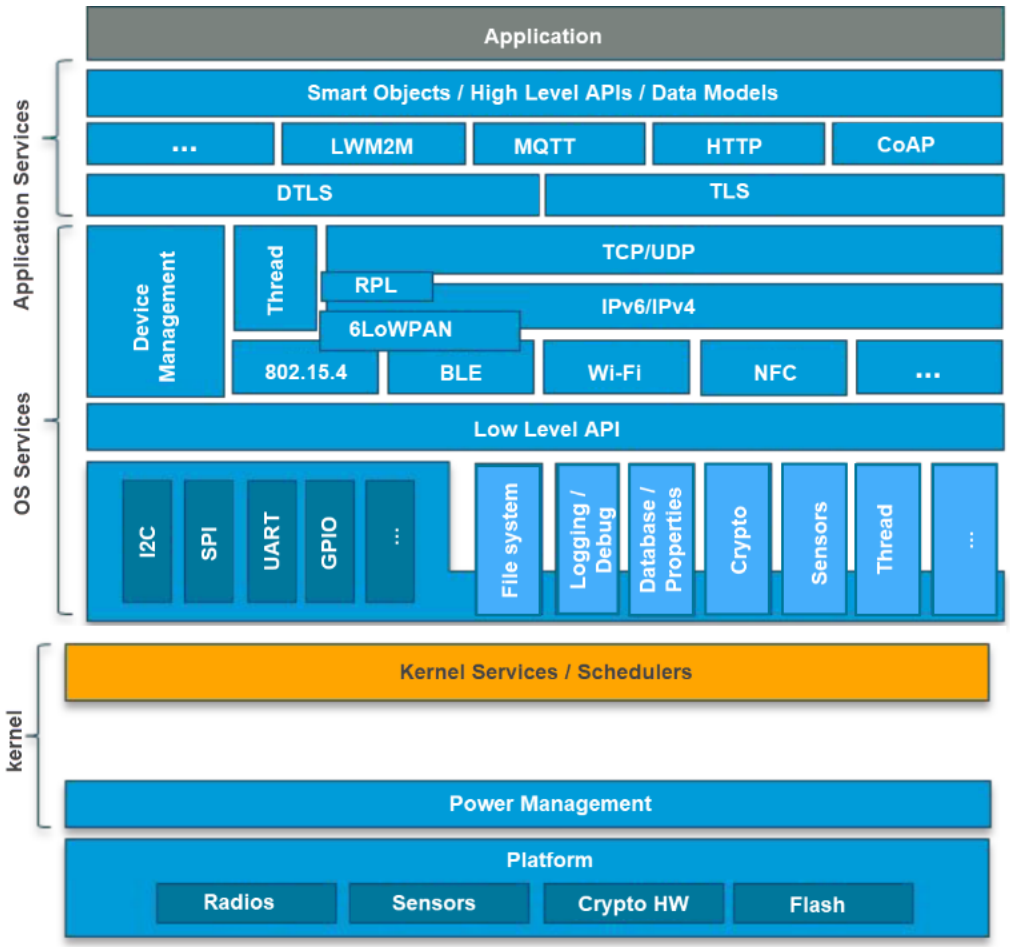

Рисунок 2.5 – Структура модулів програмного забезпечення пристроїв IoT під управлінням RTOS на прикладі Zyphyr

Для досягнення повної універсальності розробленого ПЗ, пристрої IoT мають працювати під управлінням ОС, яка ізолює програмний рівень від апаратного і дозволяє використовувати одне програмне забезпечення для різних систем, при цьому беручи на себе обов'язки планування, захисту та виконання програм (рис. 2.6).

Рисунок 2.6 – Структура модулів програмного забезпечення пристроїв IoT під управлінням OS на прикладі Linux дистрибутива Barbara OS

В результаті сформуємо три основних сценарії роботи системи:

- розробляється унікальне ПЗ для кожного, унікального за набором функцій, baremetal пристрою IoT в системі. В цьому випадку необхідно звернути увагу на керування ресурсами пристрою, оскільки його схема роботи (рис. 2.4) не підтримує багатозадачність і програма має брати до уваги, що окрім виконання граничних обчислень, цей пристрій має мати змогу виконувати свої основні функції;

- розробляється універсальне ПЗ окремо для кожного типу апаратного забезпечення пристроїв IoT під керуванням RTOS в системі. В цьому випадку, RTOS виконує основні функції керування ресурсами пристрою і розробник може створювати програму, не беручи до уваги складнощі виконання основної програми пристрою IoT. Проте, цей пристрій має мати достатню кількість вільних ресурсів системи для виконання основної програми, програми модуля граничних обчислень та самої RTOS;

- розробляється універсальне ПЗ для кожної ОС, що використовуються пристроями IoT в системі. В цьому випадку, ОС виконує всі задачі керування ресурсами пристрою IoT і розроблена програма може не брати до уваги апаратних особливостей пристроїв. Проте, апаратна основа цих пристроїв, повинна відповідати мінімальним вимогам ОС.

2.3 Особливості передачі даних для обчислення

Основною задачею, для якої використовуються граничні обчислення, є машинне навчання, яке потребує обчислень матриць. Тому, для виконання завдань системи граничних обчислень, пристрій IoT має отримати наступні дані:

- довжина даних, що передаються. Оскільки матричні операції

зазвичай потребують два набори даних, тоді варто передавати довжину даних для двох аргументів окремо. У випадку інших задач, перший аргумент може визначати інформацію, що знаходиться в другому аргументі;

- форма даних, що передаються. Для кожного аргументу варто передавати одну змінну, що відповідає за кількість рядків і одну змінну, що відповідає за кількість стовпців. У випадку не типової задачі, форма першого аргументу може визначатися як таблиця, а другий мати форму одного рядка, наприклад, для передачі багатовимірних масивів, перший аргумент буде визначати форму даних, що знаходиться у другому аргументі;

- тип даних, що передаються. Відповідає за визначення розмірності та типу елементів масиву. У випадку не типової задачі, може впливати на читання даних програмою, які потім будуть перевизначені за допомогою інформації отриманої в аргументах;

- версія заголовку. Відповідає за інтерпретацію заголовку пристроєм IoT, для підтримки майбутніх оновлень до системи виконання граничних обчислень, або у випадку несумісності деяких пристроїв системи між собою;

- версія програмного забезпечення. Відповідає за розпізнавання пристроями IoT інших сумісних пристроїв, що підтримують такі самі операції для граничного обчислення, тобто на яких встановлена така сама версія модуля граничних обчислень.

На основі переліку необхідних даних, побудуємо заголовок для передачі в розробленій системі виконання граничних обчислень (рис. 2.7). Перший байт вказує тип заголовку. Цей, у нашому випадку, надлишковий байт дозволяє в майбутньому розширити або повністю замінити структуру заголовку не створюючи конфліктів зі старими версіями програмного забезпечення. Наприклад, можна вести новий заголовок, що замінює значення поля довжини даних з кількості байтів до кількості

байтів кратній 64, тим самим збільшуючи максимальний розмір даних в 64 рази, або можна створити новий заголовок, який містить тільки один байт для відображення довжини даних, тим самим зменшивши максимальну кількість даних, що передаються, в 64 рази. Наступний байт необхідний для визначення версії програмного забезпечення. На практиці можливі випадки, коли в одній локальній мережі знаходяться пристрої, що мають різні модулі для виконання граничних обчислень, у цій ситуації цей байт дозволяє розрізнити пристрої, які можуть виконати запит і які ні, тим самим уникаючи ситуації помилкової обробки даних.

В третьому байті передається тип даних, молодші чотири байти яких відповідають за розмірність елементів (1 байт, 2 байти тощо), а старші чотири байти відповідають за тип змінної (цілочисельний, з плаваючою комою тощо).

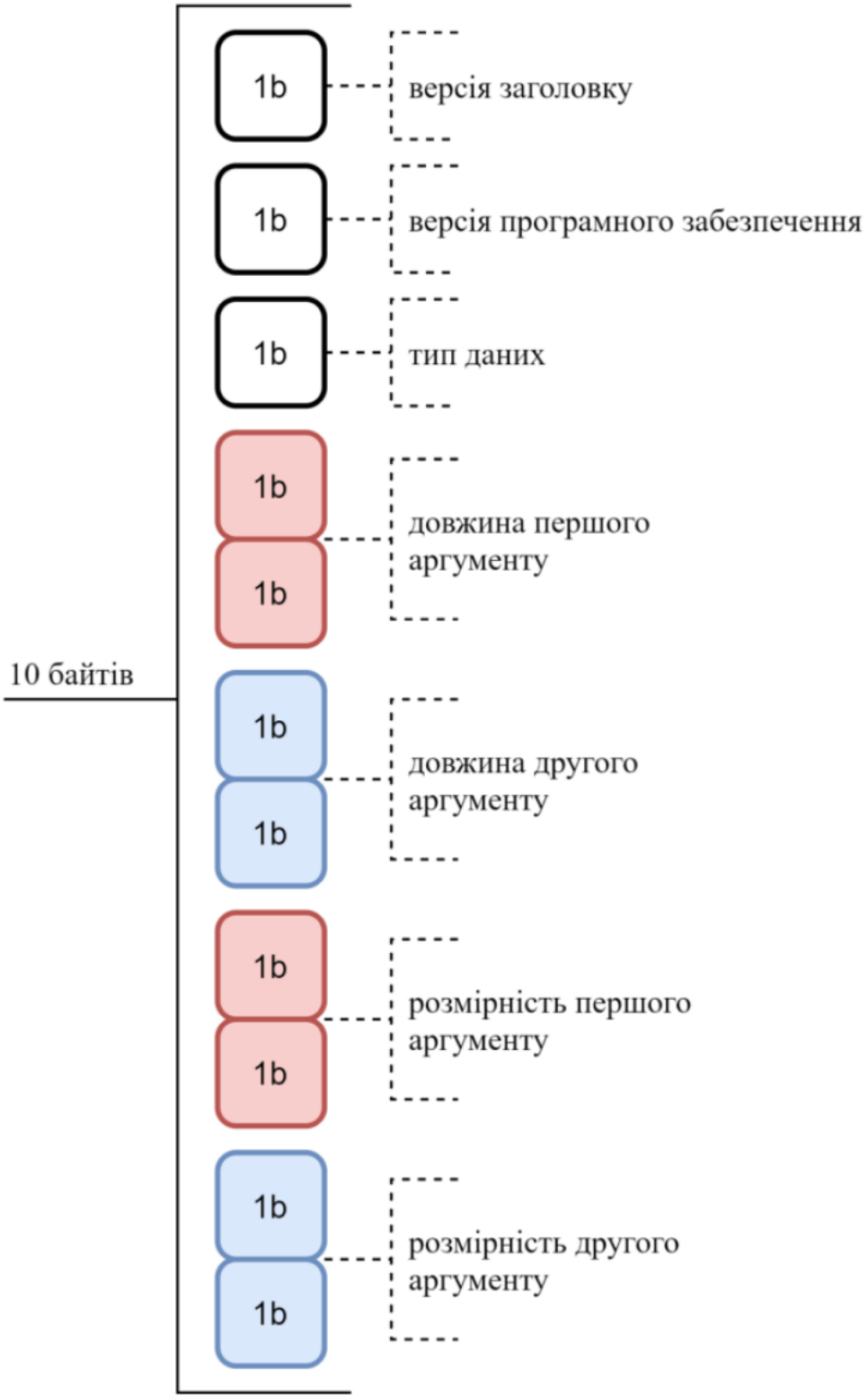

Рисунок 2.7 – Структура стандартного заголовку для передачі даних в системі виконання граничних обчислень

Наступними передаються дві пари байтів, які визначають довжину першого та другого аргументу відповідно. Ця інформація вказує на кількість елементів, які мають прочитати пристрої IoT. При цьому стандартний заголовок, дозволяє передати масиви даних до 65 тисяч елементів.

Завершують заголовок два байти, що відповідають за розмірність першого аргументу і два байти, що відповідають за розмірність другого аргументу. Розмірність аргументів складається з першого байту, що відповідає за кількість рядків масиву і другого байту, що відповідає за кількість стовпців масиву. Стандартний заголовок дозволяє передати до 255 рядків і 255 стовпців даних, але розмірність даних може бути перевизначеною всередині програми, оскільки розробник буде мати доступ до всіх переданих даних. Для цього потрібно буде повідомити програму, що дані мають бути вільно інтерпретовані, наприклад, вказавши 0 в якості кількості рядків і стовпців.

Формуючи відповідь, у стандартному випадку, існує необхідність передати тільки один масив даних. Тому достатньо сформувати заголовок (рис. 2.8), що містить інформацію тільки про один масив даних. При цьому, необхідно пересвідчитись, що розрахунки було закінчено успішно, тому заголовок передається у вигляді першого байту, що відповідає за версію заголовку, другого байту, що відповідає за версію програмного забезпечення та третього байту, що відповідає за код операції. Тоді у випадку, коли пристрій IoT не може виконати операцію, він повертає код помилки третім байтом. Наступними передається байт, що інтерпретується як тип даних, два байти, які позначають розмірність результуючого масиву, та два байти, що позначають розмірність

результуючого масиву. Так само як і у випадку заголовку для передачі даних, стандартний заголовок відповіді дозволяє передати до 65 тисяч елементів розмірністю до 255 рядків та 255 стовпців.

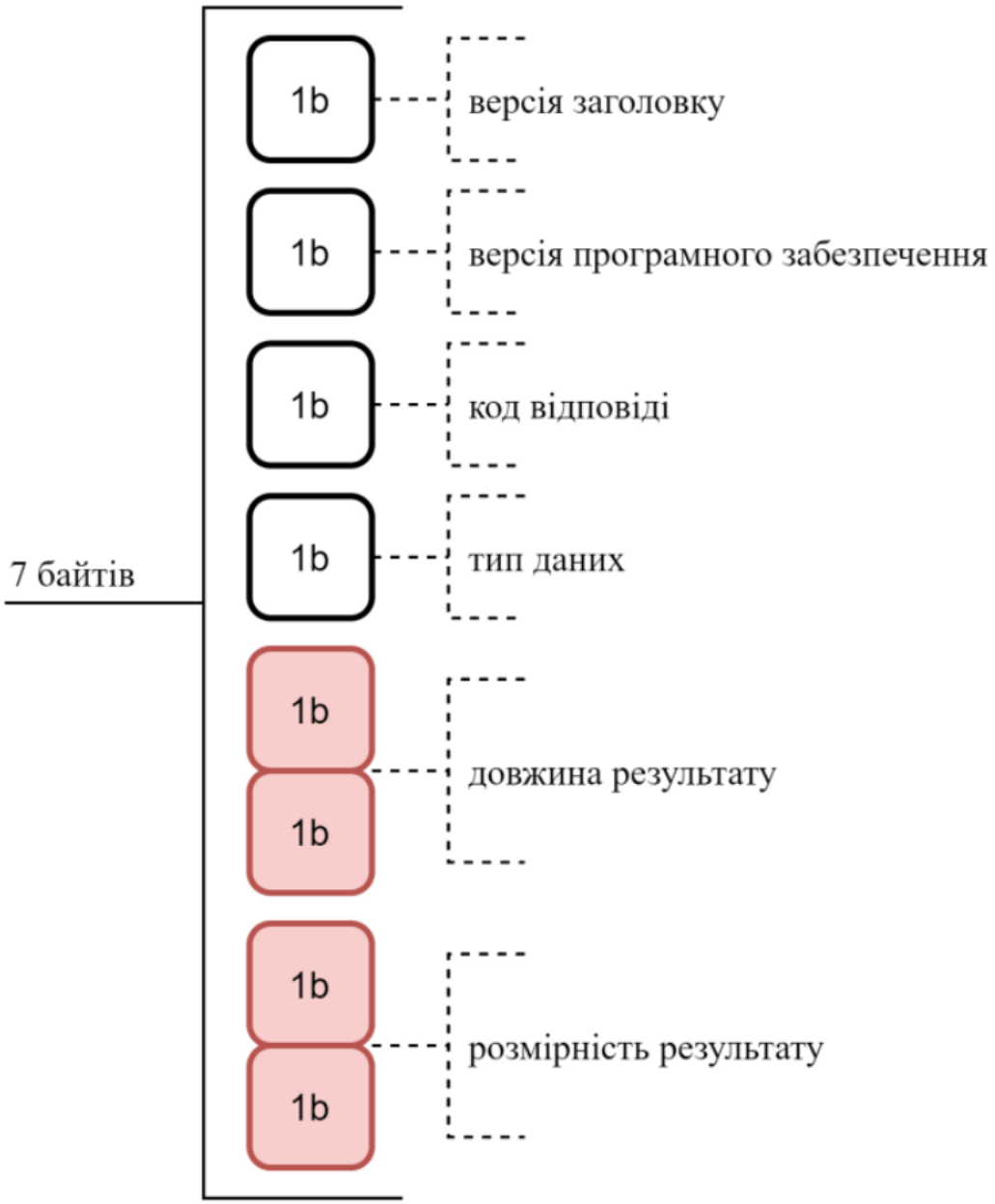

Рисунок 2.8 – Структура стандартного заголовку відповіді при передачі даних в системі виконання граничних обчислень

2.4 Підхід до організації системи граничних обчислень на наявних в мережі пристроях IoT

Створення системи виконання граничних обчислень на пристроях IoT, що знаходяться в одній локальній мережі, складається з наступних кроків:

- ідентифікація навантаженості окремих пристроїв IoT. В той час, як деякі пристрої не потребують значних обчислювальних ресурсів, як наприклад сенсори якості повітря, або замки, вони все рівно комплектуються відносно потужними платами, оскільки вони потребують встановлення з'єднання використовуючи стек TCP/IP і мають мати високу швидкість відповіді, для створення комфортного досвіду користувача. Розробляючи модулі виконання граничних обчислень для пристроїв цієї категорії, варто передбачити переривання обчислення для виконання основної задачі пристрою. При цьому, є пристрої, які задіюють більшу частину своїх ресурсів, як наприклад, IP камера. У цьому випадку варто передбачити тільки односторонню роботу пристрою, оскільки ця категорія пристроїв немає достатньої кількості вільних ресурсів, для виконання додаткових обчислень, і використовує хмарні обчислення для виконання своїх основних задач;

- розділення пристроїв на групи в залежності від їх навантаження та обчислювальних потужностей. За умови, якщо в системі наявні пристрої набагато більш потужні за решту, варто передбачити в програмному забезпеченні модулів виконання граничних обчислень, пріоритетне використання таких пристроїв. Оскільки, час передачі обмежуються характеристиками мережі і є схожим для всіх пристроїв, тобто передача даних на різні пристрої займе однаковий час, то виконання обчислень на

більш потужних пристроях дозволить значно заощадити загальний час виконання обчислень системою;

- розробка модуля програми. На цьому етапі важливо взяти до уваги загальну вільну потужність системи, оскільки немає сенсу створювати великий набір розподілених операцій для граничного обчислення, якщо для більшості з них не вистачить ресурсів;

- адаптація модуля програми для різних пристроїв IoT. Для пристроїв що керуються операційною системою, достатньо створити тільки одну програму у той час, як baremetal та RTOS пристрої вимагають унікальний підхід для кожного пристрою. На цьому етапі важливо використати дані з перших двох кроків для того, щоб пристрої IoT мали правильну завантаженість. У випадку універсальної програми для ОС варто використати модульність ПЗ і розділити серверну та клієнтську частину програми, оскільки деякі пристрої IoT не потребують граничних або хмарних обчислень і мають вільні ресурси для допомоги іншим пристроям, тобто можуть виконувати тільки роль серверу, у той же час інші пристрої IoT, можуть мати обмеженні ресурси і потребу у граничних обчислення, тим самим мають виконувати тільки роль клієнта;

- Тестування системи. На цьому етапі варто пересвідчитись, що розроблений модуль граничного обчислення, встановлений в пристрої IoT, не перешкоджають виконанню основних функцій системи. Оскільки у найгіршому випадку, пристрій IoT може мати велику затримку, яка буде помітною користувачем пристрою і буде перешкоджати нормальній роботі системи пристроїв IoT. Або пристрій IoT, на який відправляють задачу для граничного обчислення, буде затримувати виконання роботи системи, тим самим створюючи вразливість безпеці системи, якщо, наприклад, задача, яку було затримано була відправлена з камери, яка мала розпізнати рух зловмисника. Тому, на цьому етапі, важливо пересвідчитись, що пристрої продовжують ефективно виконувати свої основні функції при цьому не створюючи затримки для виконання

граничних обчислень.

Під час розробки, також можлива необхідність у аналізі пристроїв розробником, оскільки, якщо в системі очікується невелика кількість малопотужних пристроїв baremetal і велика кількість потужніх пристроїв з вільними обчислювальними ресурсами, то розробка унікального програмного забезпечення для кожного baremetal пристрою є дорожчою ніж очікувана вигода від використанні їх в системі і тому варто виключити їх з мережі виконання граничних обчислень. Проте, у випадку RTOS пристроїв та пристроїв під управлінням ОС, є можливість у використанні універсального програмного забезпечення і тому навіть менша кількість малопотужних пристроїв в системі має бути використаною, оскільки вони не потребують додаткового ПЗ.

2.5 Особливості реалізації системи граничних обчислень на наявних в мережі пристроях IoT

Виконання граничних обчислень на наявних в локальній мережі пристроях Інтернету Речей дозволяє досягти ефекту використання граничного серверу за допомогою задіяння вільних обчислювальних ресурсів пристроїв наявних в мережі. Це можна досягти підключаючи пристрої напряму один до одного, або використовуючи додатковий пристрій для розподілення задач в системі. Також можливий гібридний варіант, де додатковий пристрій використовується для моніторингу стану навантаження системи і пристрої використовують інформацію від цього пристрою для розподілення свої задач на інші пристрої в мережі. При цьому baremetal пристрої вимагають модифікації основної програми для підтримки граничних обчислень, у той час, як пристрої, що використовують RTOS та ОС підтримують багатозадачність і можуть виконувати програму для виконання граничних обчислень паралельно з основною програмою.

Створюючи заголовок для передачі даних, необхідно передбачити можливість його модифікації у майбутньому, наприклад, передаючи код заголовку першим байтом. Так як основними задачами для граничних обчислень є обрахунок матриць, разом з заголовком варто передати довжину та розмірність двох матриць даних та тип даних. Заголовок відповіді повинен містити код результату, на випадок, якщо пристрою потрібно буде відмовити у виконанні обчислень, та інформацію тільки про одну матрицю даних для передачі відповіді.

При розробці системи виконання граничних обчислень, варто звернути увагу на навантаженість та потужність пристроїв, виключивши з системи для виконання запитів малопотужні пристрої та пристрої з недостатньою кількістю ресурсів для виконання обчислень. У випадку baremetal пристроїв, важливо оцінити потенційний приріст обчислювальних потужностей системи для кожного окремого пристрою IoT, оскільки вони потребують повної переробки ПЗ і приріст потужностей може бути меншим від втрат на розробку нового ПЗ.

Під час тестування системи необхідно переконатися, що пристрої IoT не тільки продовжують стабільне виконання свої основних задач, але й не створюють затримки у виконанні граничних обчислень, оскільки така затримка, може вплинути на роботу пристрою, що створив запит на обчислення і в найгіршому випадку призвести до затримки в роботі важливих IoT систем.

Розглянуто спосіб організації граничних обчислень, який використовує вільні ресурси пристроїв Інтернету Речей, що є наявними в локальній мережі і призводить до підвищення захищеності даних за рахунок зменшення трафіку, який передається з локальної мережі, та зменшення вартості виконання таких обчислень за рахунок використання вже наявних апаратних потужностей. Запропоновано підхід до організації системи реалізації граничних обчислень на локальних пристроях Інтернету Речей, який не залежить від апаратної складової використаних

плат, і дозволяє додавати нові пристрої в систему виконання граничних обчислень без оновлення програмного забезпечення в модулях використаних плат.

На основі запропонованого способу розроблена структура комп'ютерної системи, яка дозволяє реалізувати гнучку систему виконання граничних обчислень без використання сторонніх обчислювальних пристроїв, і дозволяє зменшити таким чином час обробки даних.

3 УНІВЕРСАЛЬНА ПРОГРАМА ВИКОНАННЯ ГРАНИЧНИХ ОБЧИСЛЕНЬ НА ПРИСТРОЯХ ІНТЕРНЕТУ РЕЧЕЙ

3.1 Програмні засоби системи виконання граничних обчислень

Розроблена, за підходом наведеним у розділі 2, система виконання

граничних має виконуватися на різних версіях операційних систем (ОС), тому мовою розробки було обрано Python. Інтерпретатор Python забезпечує однакове виконання програми в незалежності від апаратної складової та ОС пристрою Інтернету Речей (IoT), проте виконання коду таких програм сповільнений, через використання динамічної типізації та відсутності компіляції програми.

Для вирішення недоліків інтерпретатора Python, було використано модуль Cython, що надає можливість статичної компіляції вихідного коду Python і підтримку можливості використання оптимізованих функцій мови С в коді Python. Це дозволяє створювати розширення мовою С для програми Python і при цьому зберігає кроссплатформеність цієї мови програмування. Cython обєднує можливості Python і С та дозволяє:

- писати код Python, що використовує модулі С або С++ в такий самий спосіб як і звичайні модулі мови;

- легко перетворити інтерпретуємий код Python на скомпільований код С, додавши декларації статичного типу використовуючи синтаксис Python;

- скомбінувати вихідний код Python та С під час виконання пошуку помилок (debugging);

- ефективно взаємодіяти з великими наборами даних, наприклад за допомогою багатовимірних масивів NumPy;

- швидко створювати програми у великій та широко використовуваній екосистемі CPython;

- інтегрувати розроблену програму з наявним кодом та даними із застарілих, низькорівневих або високопродуктивних бібліотек та програм мовами С та Python.

Для передачі даних через стек TCP/IP можна використати декілька модулів Python:

- модуль socket стандартної бібліотеки Python, що надає доступ до

інтерфейсу сокетів Берклі та доступний на всіх сучасних Unix ОС, а також на Windows ОС, MacOS, та додаткових платформах. Модуль звертається напряму до інтерфейсу API сокетів операційної системи і тому його поведінка не залежить від апаратної плати пристрою IoT. Інтерфейс модуля є прямою адаптацією викликів сокетів системи Unix до об'єктно орієнтованого стилю мови Python: функція socket() повертає об’єкт типу socket що імплементує системні виклики сокетів. Типи параметрів є більш високого рівня ніж в стандартному системному інтерфейсі написаного мовою C, оскільки алокація пам’яті для буфера даних є автоматичною для операцій отримання даних через сокет, а довжина буфера для передачі інформації через сокет задається неявно;

- модуль asyncio, що містить надбудову над модулем socket, для паралельного виконання операцій над ними використовуючи async/await синтаксис. Цей модуль є основою багатьох асинхронних фреймворків Python для високопродуктивних мереж, веб серверів та інших систем, що потребують асинхронного виконання задач. Він ідеально підходить для реалізації асинхронних задач читання сокетів та високорівневої структуризації мережевого коду. asyncio додатково надає інструменти для виконання коду Python паралельно з повним контролем над його виконанням і дозволяє ефективно імплементувати протоколи передачі даних.

В нашому випадку, клієнт та сервер мають підтримувати декілька з’єднань одночасно, тому для програми сервера та клієнта використаємо модуль asyncio. У той же час, для отримання інформації про стан пристрою достатньо одного потоку, тому для функції пошуку пристроїв IoT в мережі використаємо синхронний модуль socket для забезпечення більшої швидкодії. Оскільки розроблена система граничних обчислень в першу чергу орієнтується на виконання розрахункових задач, для математичних операцій було використано модуль NumPy. NumPy є основним модулем для виконання наукових обчислень на Python і надає

засоби створення багатовимірних масивів даних, засоби перетворення цифрової репрезентації даних в масиві, засоби швидкого виконання операцій над масивами, таких як операції над масивами, операції сортування, операції зміни форми масиву, операції дискретного перетворення Фур'є, та інші базові операції лінійної алгебри та математичної статистика. Серед переваг використання модуля NumPy потрібно виділити, що його побудовано навколо використання об’єктів типу ndarray. Це інкапсулює багатовимірні масиви однорідних типів даних, причому багато операцій виконуються в скомпільованому коді для максимальної продуктивності. Існує кілька важливих відмінностей між масивами NumPy та стандартними послідовностями Python:

- масиви NumPy мають фіксований розмір, що визначається при створенні, на відміну від списків Python, які можуть динамічно зростати. При спробі змінити розміру масиву ndarray буде створено новий екземпляр об’єкту і видалено оригінал;

- всі елементи масиву NumPy мають однаковий тип даних і розмір у пам'яті, але можна створити масив об’єктів Python, включаючи об’єкти модуля NumPy, досягаючи тим самим масиву, що складається з різних за розміром елементів;

- масиви NumPy, завдяки своїй структурі, сприяють виконанню просунутих математичних операцій над великою кількістю даних. Як правило, такі операції виконуються ефективніше, ніж це можливо за допомогою вбудованих засобів Python;

- більшість наукових і математичних модулів Python використовують масиви NumPy, так як вони не тільки підтримують операції над масивами типу ndarray, але й перетворюють стандартні масиви мови Python перед обробкою. Іншими словами, масиви NumPy дозволяють розробляти універсальні засоби для сучасного науково-математичного програмного забезпечення на основі Python.

Але найбільшою перевагою використання NumPy в мережевому програмному забезпечення є вбудовані засоби перетворення масивів даних в масив байтів за заданою формою. Це є корисним, оскільки передача даних через сокети стеку TCP/IP можлива тільки у вигляді байтів, при цьому різні baremetal системи можуть читати дані як зліва направо так і зправа наліво, тому для сумісності пристроїв IoT, що працюють під управлінням ОС та baremetal пристроїв, необхідно реалізувати можливість зміни порядку байтів при перетворенні з інших типів даних і модуль NumPy надає такі засоби.

3.2 Алгоритм передачі даних пристроєм клієнтом

Прикладне програмне забезпечення (ПЗ) пристрою IoT може використовувати функції розробленого модуля для створення задачі на виконання граничних обчислень (рис. 3.1). Процес складається з наступних кроків:

- прикладне ПЗ створює запит на виконання граничних обчислень, який складається з коду операції та даних для обробки і викликає відповідну функцію розробленого модуля;

- функція створення задачі для виконання на пристроях IoT в локальній мережі отримує інформацію від прикладного ПЗ;

- з списку адрес обираються доступні пристрої IoT, на які буде розподілено завдання;

- головна програма розподіляє завдання між доступними пристроями і запускає підпрограми для кожного з пристроїв в окремому потоці використовуючи модуль asyncio;

- підпрограма трансформує вхідні дані в масив байтів, використовуючи функцію to_bytes() масиву NumPy;

- підпрограма відправляє дані до обчислення на пристрої IoT використовуючи функцію loop.sock_sendall() модуля asyncio;

- підпрограма читає отримані дані в масив NumPy і повертає результат прикладному ПЗ, що створив запит на граничне обчислення.

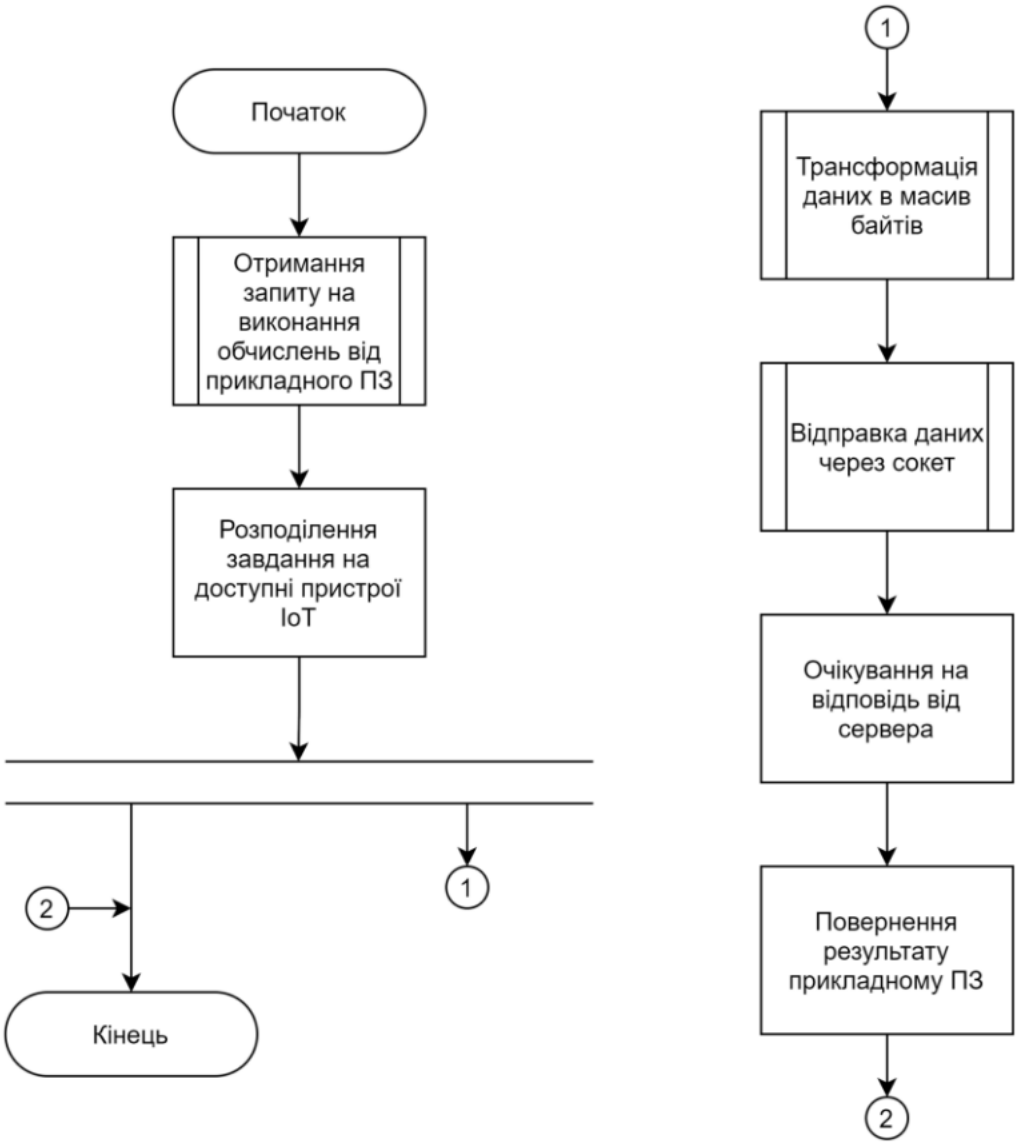

Рисунок 3.1 – Алгоритм відправки даних на обчислення пристроями

IoT, що виконується модулем клієнтом граничних обчислень

Наведений алгоритм дозволяє одночасне виконання всіх частин обчислення використовуючи багатозадачність системи. При цьому, прикладне ПЗ зможе отримати результат відразу після завершення обчислень пристроєм, не очікуючи на закінчення роботи решти пристроїв, що може бути корисним для задач, які можуть використовувати часткові розв’язки. Наприклад, за умови, що аналіз даних відбувається по декільком критеріям, отримавши негативний результат по одному з них, програма вже може зробити висновок, що не всі критерії отримають позитивний результат. У випадку систем безпеки, отримавши результат розпізнавання небезпеки по одному з критеріїв, пристрій IoT вже зможе надіслати сигнал тривоги, тим самим скоротивши час відповіді у небезпечній ситуації.

3.3 Алгоритм обробки даних пристроєм виконувачем граничних обчислень

Процес обробки даних (рис. 3.2), що поступають на пристрій, який виконує граничні обчислення, складається з наступних кроків:

- розроблений модуль запускає серверну програму, яка виконується у нескінченному циклі;
- серверна програма очікує на появу нового клієнта;
- при надходженні запиту на виконання граничних обчислень від клієнтського модуля пристрою IoT, головна програма запускає підпрограму обслуговування клієнта, використовуючи модуль asyncio;
- головна програма повертається до очікування нових клієнтів;
- підпрограма читає заголовок з сокету, перетворюючи масив байтів за допомогою модуля NumPy;
- використовуючи інформацію з заголовку, підпрограма читає дані

до обробки з сокету, використовуючи функцію loop.sock_recv(sock, n) модуля asyncio, яка читає n байтів з сокету sock;

- використовуючи інформацію з заголовку, підпрограма виконує запитані операції над даними, прочитаними на кроці 6;

- підпрограма відправляє результати обчислення використовуючи функцію to_bytes() модуля NumPy і функцію sock_sendall(sock, data) модуля asyncio, яка відправляє масив байтів data через сокет sock.

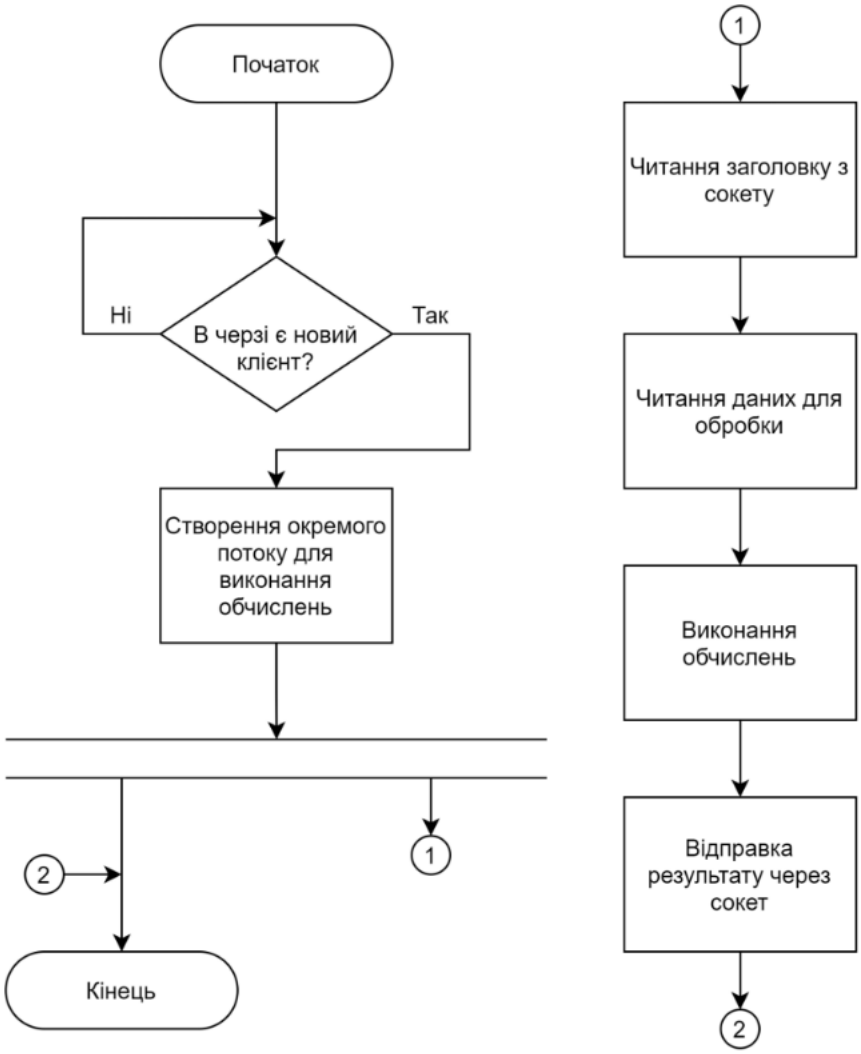

Рисунок 3.2 – Алгоритм виконання граничних обчислень на пристрої IoT, що виконує функції серверу, запити на який надходять від модулів клієнтів

Описаний алгоритм дозволяє паралельне виконання декількох завдань граничних обчислень використовуючи багатозадачність системи. За рахунок цього, пристрій IoT, що виконує функції граничного серверу, має можливість одночасно обслуговувати декілька інших пристроїв клієнтів. Це особливо корисно у випадках, коли пристрій IoT, який виконує обчислення, має апаратні ресурси для виконання двох, або більше, програм одночасно.

3.4 Опис розробленого модуля виконання граничних обчислень

Структурна схема модуля виконання граничних обчислень зображена на рисунку 3.3. Модуль складається з 2 пакетів і одного виконуваного файлу Python в корінній директорії (Edge Computing). Файл Edge_Computing.py містить класс Edge_Computing, що є вхідною точкою програми і відповідає за надання розробнику доступа до всіх складових програми. До модуля входять наступні пакети:

- main – містить абстрактні класи, що відповідають за представлення основних елементів модуля як програма серверу, програма клієнта та програма пошуку та відгуку для пристроїв IoT в локальній мережі;

- workers – містить абстрактні класи, що відповідають за представлення змінних частин програми, які визначає розробник, такі як робота з сокетами та виконання обчислень.

Клас Edge_Computing містить наступні функції:

- getServer(self, port), що повертає статичний екземпляр класу Edge_Server та встановлює номер порта port на якому виконується програма граничного серверу;

- getClient(self, addrs), що повертає статичний екземпляр класу Edge_Client та встановлює список адресів пристроїв IoT, на які можна направити запити на виконання граничних обчислень;

- getStatus(self), що повертає статичний екземпляр класу Edge_Status.

Також в модуль Edge_Computing.py входять наступні додаткові класи:

- Edge_Addrs, що містить список кортежів з ip адрес та портів для динамічного збереження адрес пристроїв IoT в локальній мережі;

- Edge_task, що містить інформацію про необхідні обчислення, такі як заголовок та вхідні дані.

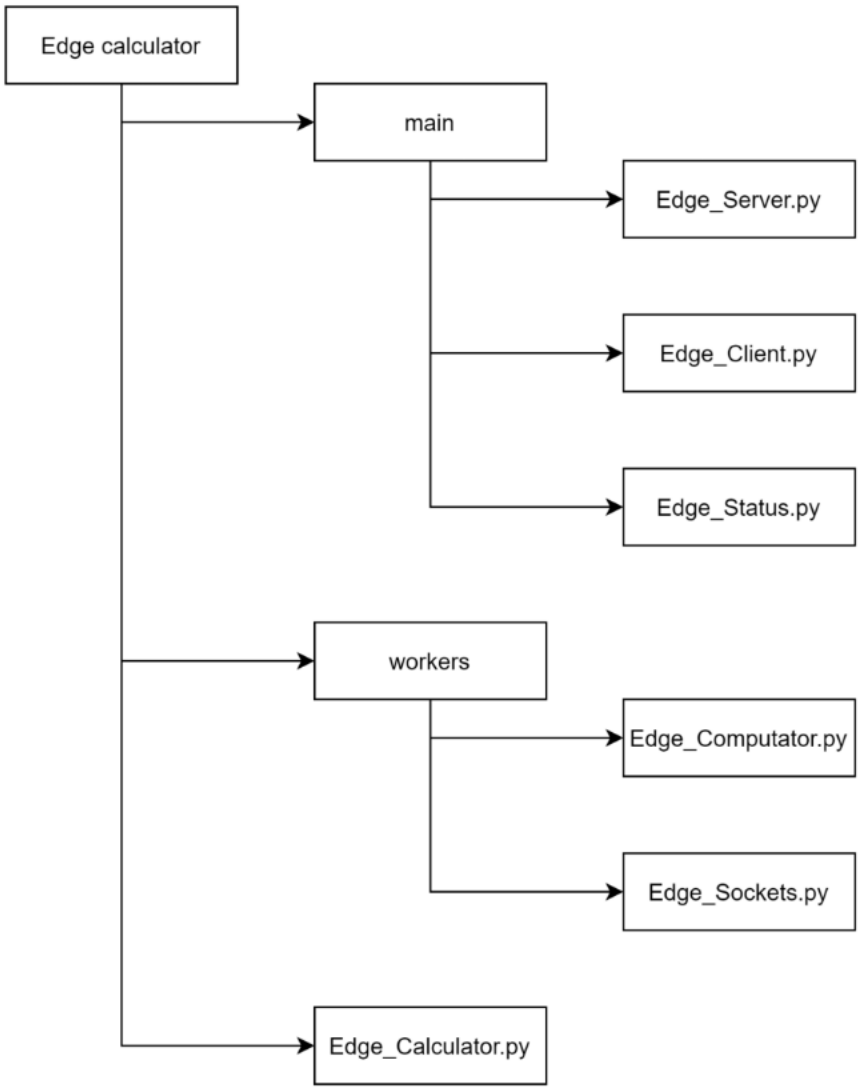

Рисунок 3.3 – Структура компонентів модуля Edge Calculator

До класу Edge_Client.py входять наступні функції:

- init (self, addrs) – функція ініціалізації класу, де addrs це екземпляр

класу Edge_Addrs, що використовується для пошуку доступних пристроїв IoT в локальній мережі;

- connect(self, n) – функція, що створює n з’єднань до пристроїв IoT в локальній мережі;

disconnect(self) – функція розриву з’єднання з підключеними пристроями IoT;

- set_header(self, operation, arg1_m, arg1_n, arg2_m, arg2_n, data_type) – функція оновлення параметрів заголовку, де operation це код операції, arg1_m і arg1_n це номер рядків та стовпців першої матриці, в той час як arg2_m і arg2_n – другої, а data_type це кодове представлення типу даних (як було описано в попередньому розділі);

- send_header(self) – функція відправки згенерованого заголовку на підключений пристрій IoT. Вимагає попереднього встановлення параметрів задачі через функцію set_header();

- send(self, arg1, arg2) – функція відправки даних для обчислення на підключений пристрій IoT. Ця функція на початку своєї роботи викликає функцію send_header(self) для передачі інформацію про розмір та форму інформації;

- receive_header(self) – функція очікування і читання даних заголовку отриманих через сокет з пристрою IoT, що виконує обчислення;

- receive(self) – функція очікування і читання результату обчислення отриманого через сокет з пристрою IoT. Ця функція перед виконанням основних інструкцій викликає функцію receive_header() для отримання інформацію про розмір та форму інформації.

До класу Edge_Server.py входять наступні функції і класи:

- init (self, port) – функція ініціалізації класу, де port це номер порта для TCP/IP сокета, що використовується для отримання даних для обчислення від пристроїв IoT в локальній мережі;

- start(self, n) – функція, що запускає підпрограму виконання граничних обчислень в окремому потоці, яка може створювати не більше

n з’єднань до пристроїв IoT в локальній мережі;

- stop(self) – функція припинення отримання нових задач на обчислення від підключених пристроїв IoT, при цьому очікуючи на завершення виконання вже розпочатих обчислень;

- receive(self) – функція очікування і читання завдання на обчислення отриманого через сокет з пристрою IoT. Ця функція спочатку отримує інформацію про передані дані з перших байтів, що є заголовком задачі, а потім читає решту інформації у вигляді, передбаченому заголовком;

- send(self, res) – функція відправки результату обчислень на підключений пристрій IoT. Ця функція генерує заголовок для передачі інформацію про розмір та форму даних.

Клас Edge_Status має головною задачею пошук пристроїв IoT в локальній мережі та визначає наступні функції:

- init (self, edge_Client, edge_Server) – функція ініціалізації класу, де edge_Client та edge_Server це працюючі екземпляри класів Edge_Client та Edge_Server;

- start(self, port) – функція, що запускає процес TCP/IP сервера в окремому потоці, задачею якого є стверджувальна відповідь на всі запити на перевірку, наявності програмного забезпечення необхідного для виконання граничних обчислень, від клієнтських модулів IoT;

- stop(self) – функція припинення роботи підпроцесу викликаного функцією start();

- search(self) – функція ідентифікації інших пристроїв IoT в локальній мережі, використовуючи методи класу Edge_Client.

Додатково модуль Edge_Computator містить словник edge_computators, що містить функції, які може виконувати даний модуль граничних обчислень та їх коди, а модуль Edge_Sockets містить надбудову над вбудованими функціями роботи з сокетами, для можливості їх швидкої заміни в майбутньому.

Розроблено модуль організації граничних обчислень, який

забезпечує високу швидкодію та універсальність програмного забезпечення. Показано, що при використанні платформи Python досягається найбільша сумісність програмного забезпечення з різними операційними системами, в порівнянні з мовами, що компілюються у системні команди і потребують розробки під кожну апаратну платформу та операційну систему окремо, а підтримка виклику функцій мови С дозволяє досягти найвищої швидкодії виконання обчислень.

Показано, що оскільки робота модуля базується на наслідуванні, розробник може швидко адаптувати розроблене ПЗ під будь-які завдання, а винесені в окремий модуль функції для граничних обчислень дозволяють швидко додати, замінити або видалити набір операцій, що підтримуються модулем виконання граничних обчислень.

4 ТЕСТУВАННЯ ВИКОНАННЯ ГРАНИЧНИХ ОБЧИСЛЕНЬ НА БЕЗСИСТЕМНИХ ПРИСТРОЯХ ІНТЕРНЕТУ РЕЧЕЙ

4.1 Проектування системи

Через відсутність операційної системи, розробка системи виконання граничних обчислень має вестись індивідуально під кожний baremetal пристрій. У цьому випадку необхідно, не тільки підібрати бібліотеки для роботи з апаратними і мережевими можливостями пристрою, але й середовище розробки, яке дозволить скомпілювати та відлагодити програму і отримати інформацію про стан пристрою під час роботи. Іншою задачею є розробка структури ПЗ, яка дозволить проводити адаптацію для різних плат пристроїв IoT без необхідності створення повністю нової програми.

4.1.1 Вибір платформи проєктування

При розробці програмного забезпечення для baremetal пристроїв Інтернету Речей, платформою проектування було обрано Arduino IDE. До головних переваг є вбудовані компілятори для більшості плат для

виробництва пристроїв IoT та розумних пристроїв і наявності інструментів відлагодження (рис. 4.1), що надають можливість користувачу отримати інформацію з плати пристрою IoT через серійний інтерфейс та COM порт комп`ютера.

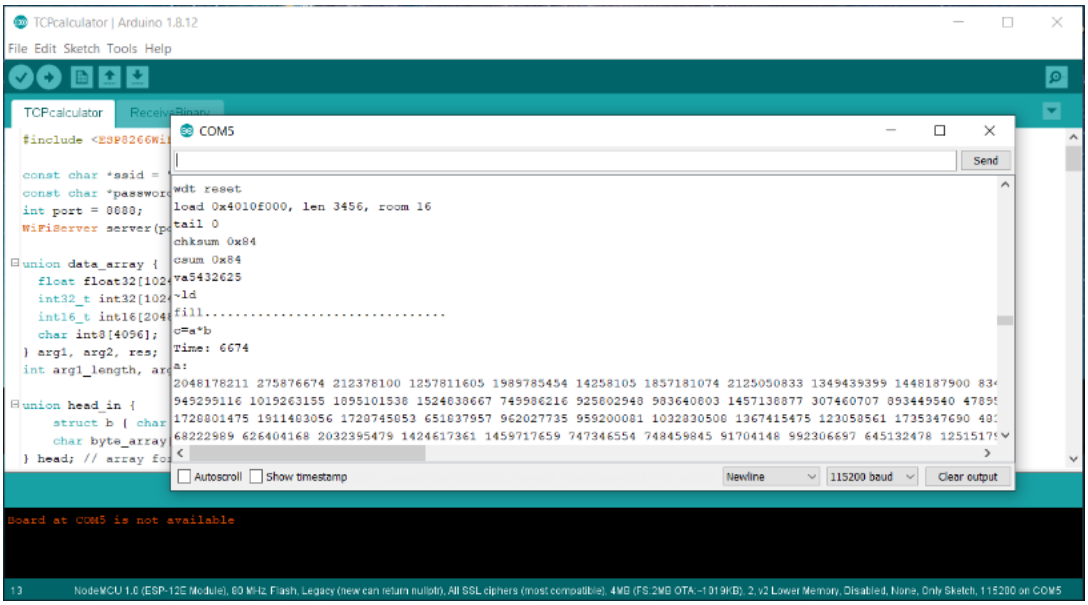

Рисунок 4.1 – Налагодження програми через вбудований засіб Serial Monitor в Arduino IDE

В якості апаратної платформи було обрано плату NodeMCU, яка працює на базі мікросхеми ESP8266 та має вбудований модуль Wi-Fi. Стандартна бібліотека Arduino для цієї плати, містить сокети TCP/IP, завдяки чому плата може бути інтегрована в систему виконань граничних обчислень, не дивлячись на низькі системні ресурси (рис. 4.2), яких не достатньо для використання повноцінної ОС.

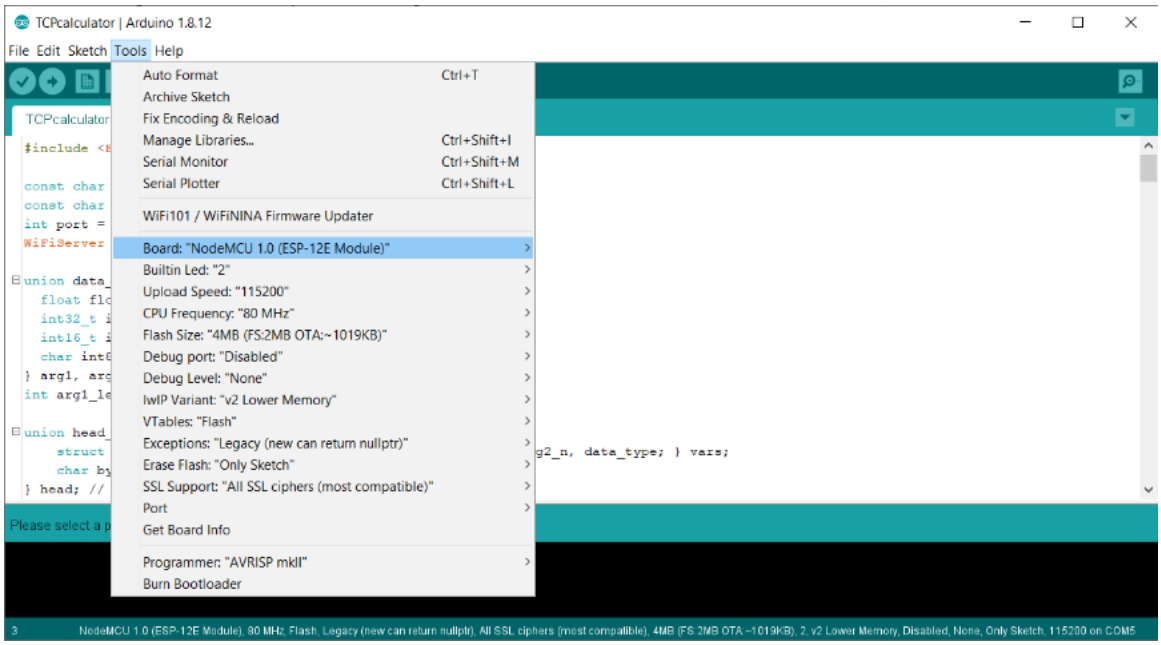

Рисунок 4.2 – Технічні характеристики плати ESP8266 в Arduino IDE

Також плата NodeMCU підтримує оновлення свого програмного забезпечення через Wi-Fi, завдяки чому її не потрібно підключати до комп'ютера і оновлення програми виконання граничних обчислень може бути виконане навіть для вже існуючих пристроїв IoT. Це створить можливість у майбутньому змінити набір функцій, які можуть бути виконані на розробленому пристрої, або змінити конфігурацію програмного забезпечення.

4.1.2 Компоненти програми

До розробленого ПЗ входять наступні модулі:

- TCPcalculator, що містить функції операцій граничних обчислень, які може виконувати пристрій;

- TCPserver, що містить модуль серверної програми, яка отримує завдання на виконання граничних обчислень;

- TCPclient, що містить модуль клієнтської програми, яка створює завдання на виконання граничних обчислень іншими пристроями IoT;

- MainProgram, що містить основну програму пристрою IoT.

В кореневій директорії програми знаходиться файл MainProgram.ino, який є точкою входу програми і містить функції setup() та loop(). цей файл також містить основну програму пристрою IoT, яка через обмеженість кількості паралельних процесів плати, виконується послідовно з програмою виконання граничних обчислень.

Клас TCPcalculator не містить функцій, які задіяні в виконанні основних задач модуля граничних обчислень і передбачає створення таких функцій окремо для кожного пристрою. У нашому випадку він містить функцію matrixmul(data_array arg1, data_array arg2, data_array& res, head_in head, head_out& res_head), що виконує матричне множення даних в arg1 на дані в arg2 використовуючи інформацію про форму і розмір даних з заголовку head і формує результуючі дані res заповнюючи інформацією про їх розмір і форму в вихідному заголовку res_head. Також цей модуль описує наступні типи даних:

- data_array – для зберігання даних масивів, які обробляються модулем;

- head_in – для зберігання параметрів з заголовку, що надійшов від клієнта;

- head_out – для заповнення вихідного заголовка параметрами про результуючу матрицю, що буде надіслано пристрою клієнту.

До класу TCPserver входять наступні функції:

int read_to(WiFiClient client, char arr[], int n) – читає n байтів в массив arr від пристрою клієнта client;

int readheader(WiFiClient client, head_in& head, int n) – читає заголовок вхідних даних head, довжиною n байтів що надійшов від пристрою клієнта

client;

int readArg(WiFiClient client, data_array& arg, int n) – читає n байтів в массив arg від пристрою client;

int readdata(WiFiClient client, data_array& arg, int m, int n, int type, int& arg_length) – читає масив байтів від client, що відповідає масиву розміром m на n елементів типу type і повертає масив arg довжиною arg_length байтів.

int calculate(head_in head, data_array arg1, data_array arg2, data_array& res, head_out& res_head, int& res_length) – виконує операцію, яка записана в head.operation, над масивами arg1 і arg2, довжина і розмірність яких знаходиться в head, і повертає заголовок res_head довжиною res_length байтів;

void edgeserverloop() – вхідна точка програми серверу, яка викликається у функції loop модуля MainProgram.ino.

Класу TCPclient містить наступні функції:

- int save(WiFiClient dev, char arr[], int n) – читає n байтів в массив arr від пристрою dev;

- int writeheader(WiFiClient dev, head_in& head) – надсилає пристрою dev заголовок для обчислення даних head;

- int readRes(WiFiClient dev, data_array& res, int n) – читає масив результату res, довжиною n що надійшов від пристрою dev;

- int readResheader(WiFiClient dev, head_out& head) – читає заголовок результату head, що надійшов від пристрою dev;

- int writedata(WiFiClient dev, data_array& arg, int& arg_length) – надсилає пристрою dev масив даних arg довжиною arg_length байтів;

- int edgecompute(char operation, data_array arg1, data_array arg2, char type, data_array& res, head_out& res_head) – створює запит на виконання методом граничних обчислень операцію operation над даними arg1 та arg2 і формує заголовок, що містить інформацію про масиви даних, повертаючи результат у вигляді матриці res та інформації про неї в

res_head. Це вхідна точка програми серверу, яка викликається функціями, які потребують граничних обчислень.

Для швидкої та ефективної трансформації даних масиву байтів у масив заданого типу, було використано програмний елемент мови C union (рис. 4.3), що дозволяє використовувати один і той самий фрагмент пам’яті для змінних різного типу та розмірності. Завдяки цьому, прочитавши байти з сокету ми можемо одразу записати їх в масив байтів, після чого звернувшись до масивів 2-байтних та 4-байтних даних ми отримаємо значення яке відповідає значенню записаних в масив байтів даних.

```
8  union data_array {
9    float float32[1024];
10   int32_t int32[1024];
11   int16_t int16[2048];
12   char int8[4096];
13 } arg1, arg2, res;
```

Рисунок 4.3 – Використання одного фрагмента пам’яті в якості масивів різних типів та довжини за допомогою програмного елемента union

Основна програма функціонування плати в якості пристрою IoT знаходиться в модулі MainProgram.ino. Так як використана плата не містить вбудованого планувальника завдань, вхідні точки основної програми та програми серверу системи виконання граничних обчислень знаходяться в функції loop(), яка почергово виконує кожну з них у нескінченному циклі.

4.2 Алгоритм передачі і обробки даних

Так як система побудована на базі плати, що підтримує тільки роботу у baremetal режимі, то виконання всіх завдань пристрою може бути виконане тільки в послідовному режимі, що було описано в розділі 2.2 (рис. 2.4). Через це, при розробці ПЗ пристрою IoT, необхідно включити виклики функцій основних задач IoT та системи виконання граничних обчислень в один цикл (рис. 4.4).

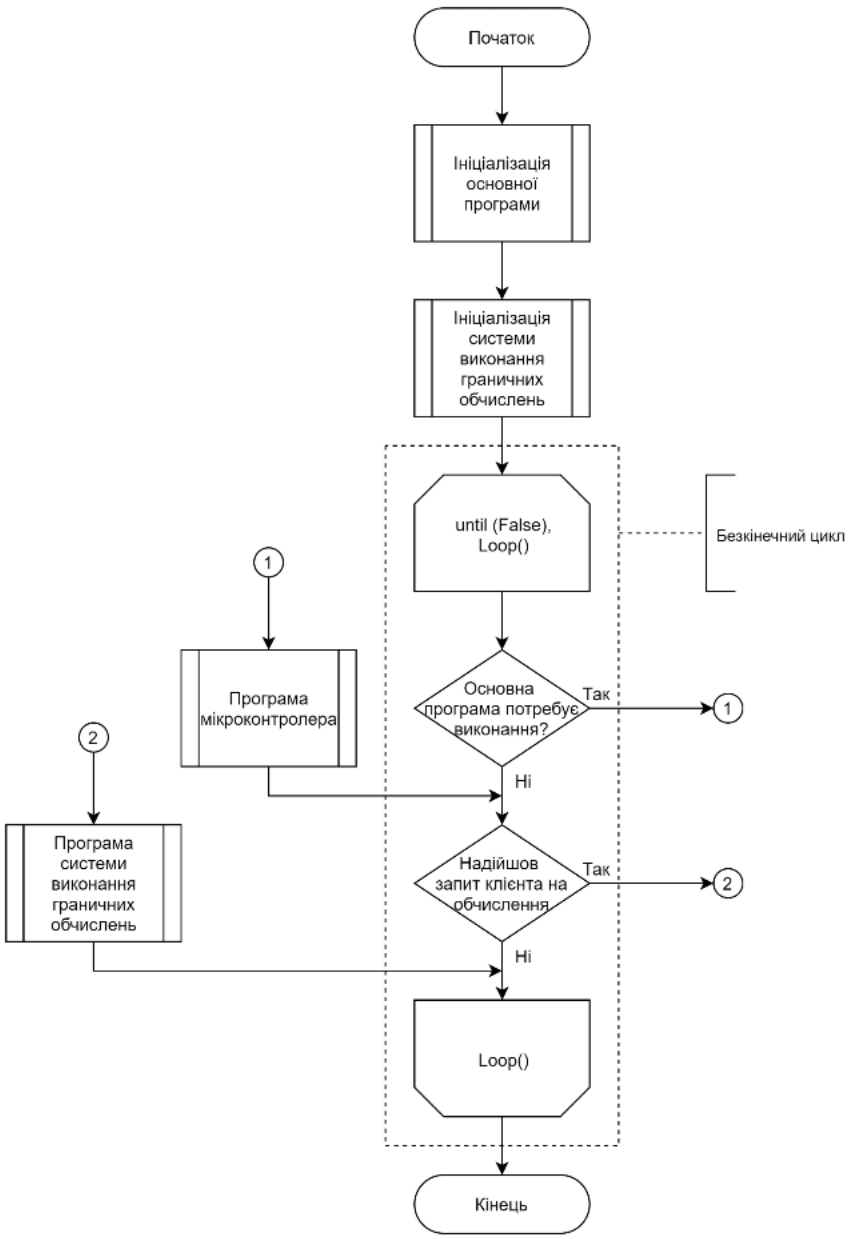

Рисунок 4.4 – Алгоритм роботи пристрою IoT з урахуванням системи виконання граничних обчислень і основної програми

У випадку програми виконання граничних обчислень, пристрій має викликати відповідні функції з основної програми, і час виконання таких функцій буде впливати на час виконання основних задач. Так як функції виконання граничних обчислень покликанні на оптимізацію часу обчислень, ми можемо знехтувати їх часом виконання, за умови, що виконання граничних обчислень на інших пристроях IoT є швидшим за виконання обчислень на самому пристрої. У той же час, серверна програма має запускатися окремо і буде створювати затримку виконання. Оскільки, основна програма має більший пріоритет, розробник може штучно виділити більше часу на її виконання, помістивши її виклик у окремий цикл (рис. 4.5).

Параметри циклів M і N беруться як відношення часу виконання задач основної програми до часу виконання системи граничних обчислень, при цьому розробник може коригувати їх в меншу сторону за умови більшого пріоритетності іншого завдання. Проте їх збільшення є небажаним, оскільки це призведе до збільшення очікування перемикання системи на іншу задачу. Рекомендовано використовувати параметри M і N в межах 1..5, щоб система мала відносно малий сталий час відгуку для кожної з задач.

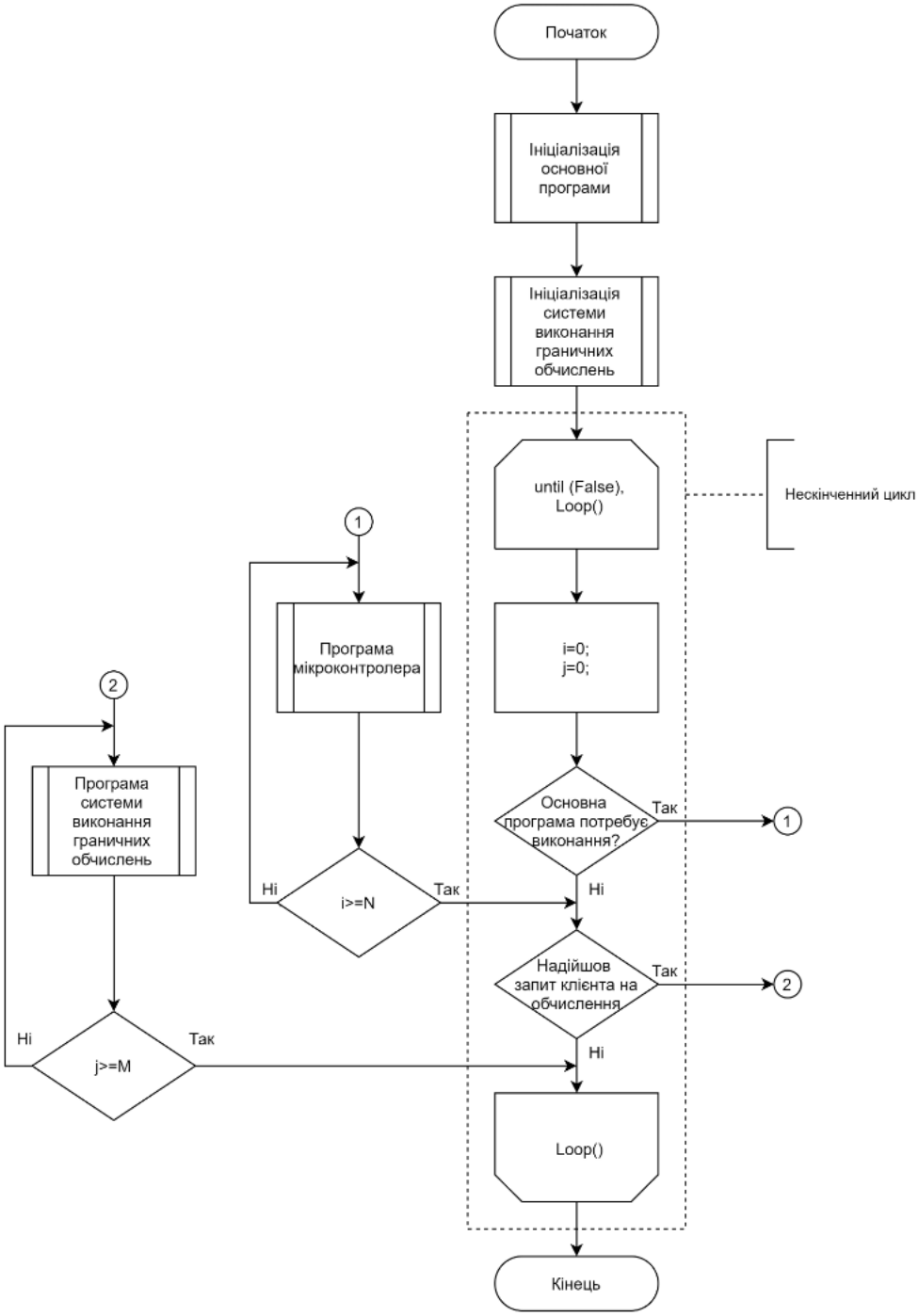

Рисунок 4.5 – Алгоритм роботи пристрою IoT з урахуванням різниці в часі виконання основної програми і системи виконання граничних обчислень

4.3 Тестування виконання граничних обчислень на платах ESP8266

За допомогою розробленої системи можна одержати пришвидшення системи за рахунок використання декількох пристроїв IoT однакової потужності без необхідності у використанні спеціалізованих пристроїв у систему. Одна із ситуацій в якій пристрої Інтернету Речей потребують хмарних обчислень це є реалізація задач машинного навчання, а однією з основних операцій, що використовується алгоритмами машинного навчання є множення матриць. Проведемо тестування виконання граничних обчислень на малопотужних пристроях IoT для оцінки співвідношення часу передачі даних та часу виконання операції множення матриць над цими даними.

Побудуємо запропоновану систему виконання граничних обчислень, що містить два пристрої IoT на базі модулів NodeMCU, один з яких використовується в ролі модуля сервера, а інший – клієнта (рис. 4.6). Оскільки безпровідна передача даних вносить значну затримку при передачі даних, плати для тестування мають додаткове з’єднання через фізичний порт GPIO16 для мінімізації затримки отримання модулем сервером інформації про початок та кінець тестування від модуля клієнта. Для отримання результатів тестування в серійному моніторі Arduino IDE, модуль сервер під'єднано до комп’ютера через USB порт.

Обмін інформації між цими двома платами виконується за допомогою мережевих протоколів стеку TCP/IP. Додатково від модуля клієнта до сервера, разом з двома 2х-вимірними масивами даних, передається програмний заголовок, який описаний в розділі 2, для

визначення форматування вхідних даних. Сервер повертає результат обчислень клієнту у вигляді заголовку для визначення довжини і розмірності результуючого масиву та відповідного масиву даних результату.

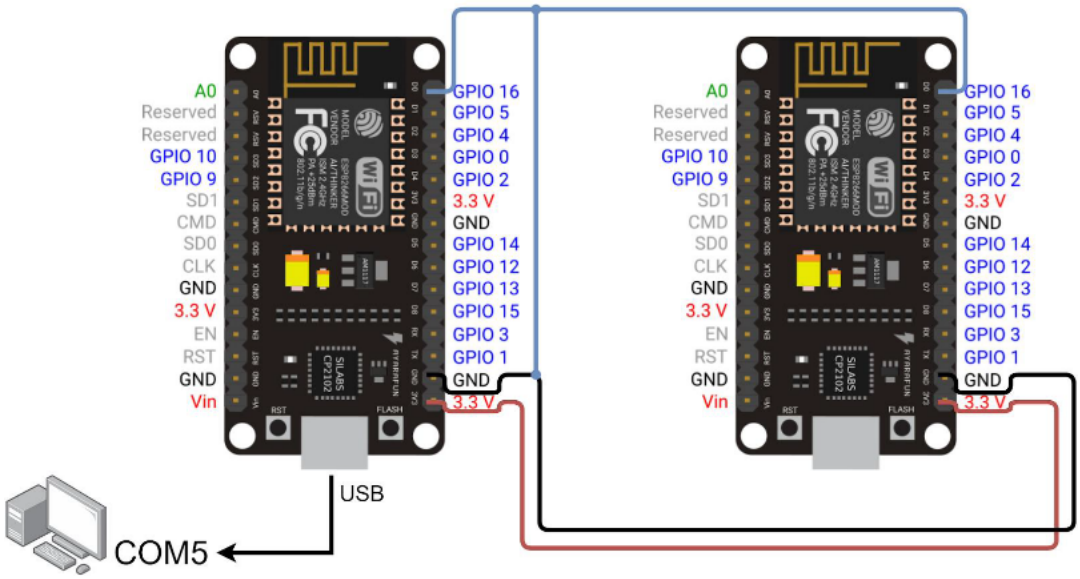

Рисунок 4.6 – Схема підключення системи з двох плат ESP8266 для тестування

Тестування починається з отримання модулем сервером даних від клієнтської системи, виконання обчислень над переданими даними та закінчується поверненням результатів клієнтській системі. Час передачі рахується від надходження сигналу на порт GPIO16 про початок передачі від плати клієнта до отримання останнього байту через TCP/IP сокет від модуля клієнта і від надсилання результатів обчислень клієнту до отримання фізичного сигналу підтвердження про кінець передачі від системи клієнта. Оскільки тестування відбуваються в нескінченному циклі, програма продовжується з початку, доки плати підключені до

джерела живлення.

На рисунку 4.7 наведено результати тестування розробленої системи для масивів різної розмірності, елементами яких є цілочисельні числа довжиною 4 байти.

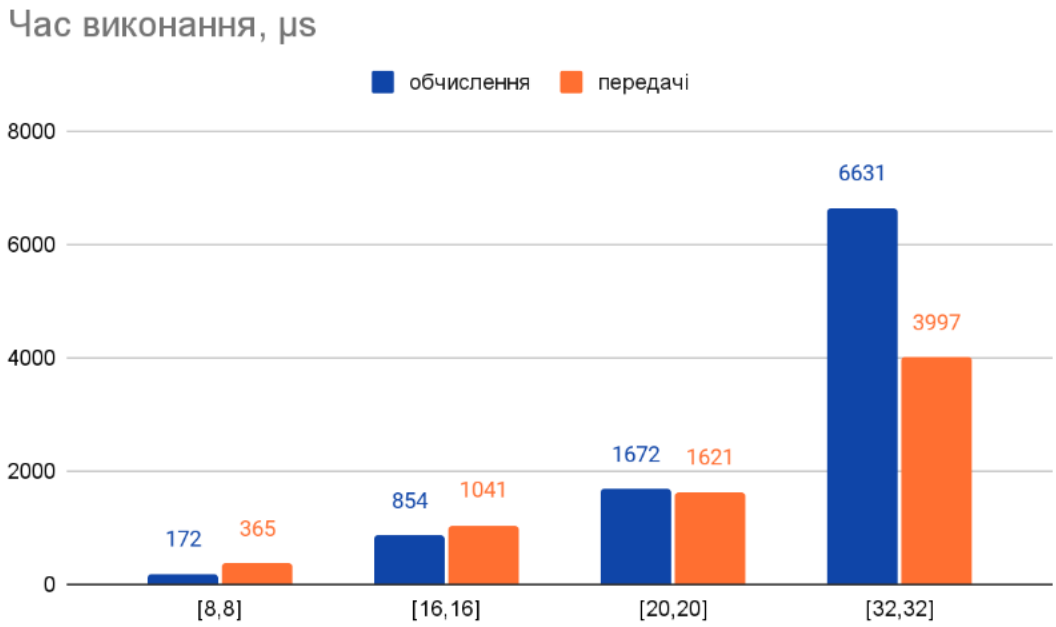

Рисунок 4.7 – Тестування граничного обчислення операцій матричного множення для цілочисельних даних на платі ESP8266

З результатів видно, що для розглянутої системи передавати масиви даних менших за матрицю [20,20] елементів має негативний вплив на ефективність використання ресурсів системи виконання граничних обчислень, оскільки час використаний для передачі буде більшим за час обчислення на самому пристроїв. Проте час обчислення збільшується значно швидше за час передачі даних, тому для даних, більших за матрицю [20,20] елементів варто використовувати граничні обчислення

використовуючи розглянуті пристрої IoT, адже тоді вдасться досягти значного прискорення в порівнянні з виконанням обчислень на одному пристрої.

З аналізу отриманих результатів, маємо, що час обробки масиву [128,128] елементів, становить щонайменше 240 мілісекунд, а час передачі буде більшим за 64 млс. Тоді розділивши цю задачу на два пристрої IoT, кожен з яких буде обчислювати половину задачі досягаємо загальний час виконання 162 млс, таким чином пришвидчуючи систему на 32%.

Але через обмежену кількість ресурсів, обробка великих обсягів даних на цих модулях може бути виконана тільки передаючи дані до обчислення частинами, тому проведено тестування пришвидшення загального часу виконання обчислень в залежності від обсягу переданих даних (рис. 4.8) за умови, що пристрій буде послідовно обробляти дані, що надходять до обчислення і при цьому відправляти частину даних до обробки на інший пристрій IoT.

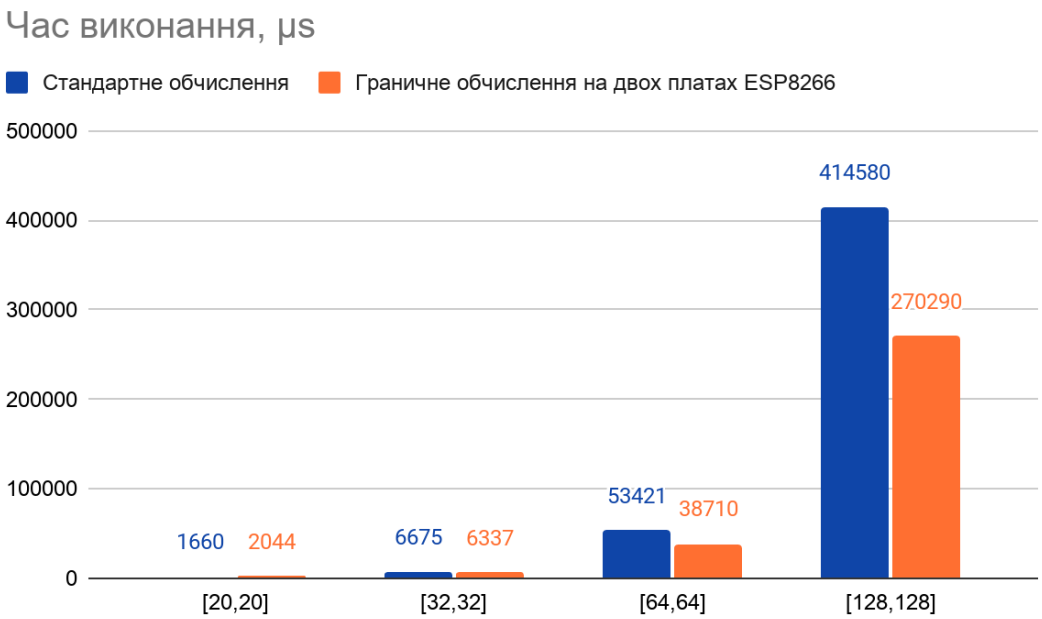

Рисунок 4.8 – Порівняння часу виконання системою граничного обчислення операції матричного множення з розділенням задачі на два пристрої IoT з виконанням обчислень без залучення системи граничних обчислень

З результату другого тесту, отримаємо, що масиви даних розміром менше за [32,32] елементів, при розділені задачі між двома пристроями витрачають більше часу на обчислення ніж за умови використання одного пристрою. Отже необхідною умовою, для ефективного розділення задачі на 2 пристрої, є час передачі, який менший за час обчислення, щонайменше на одну четверту.

Запропонованої системи з двох плат NodeMCU не достатньо для виконання складних обчислення, проте досягаючи зменшення часу роботи на 32% при задіяні двох пристроїв, ця система має ресурси для ефективного виконання обчислень середньої складності.

4.4 Тестування виконання граничних обчислень на платі ESP8266 і пристрою на базі ОС Linux

Протестуємо виконання граничних обчислень на платі ESP8266 та більш потужному пристрою IoT, що працює під ОС Linux. Задачею граничних обчислень, так само як і в підрозділі 4.3, оберемо виконання матричного множення.

Очевидно, що більш потужний пристрій IoT виконує обчислення швидше за малопотужню плату ESP8266, тоді тестування за умови використання плати ESP8266 в ролі клієнта граничних не є доцільним, оскільки граничні обчислення, в цьому випадку, буде вигідно виконувати за умови, що час передачі даних, що в цьому випадку обмежується

апаратними особливостями плати ESP8266, є нижчим за час виконання граничних обчислень безпосередньо на цьому пристрої (рис. 4.7).

Для дослідження доцільності виконання граничних обчислень більш потужного пристрою, що підтримує багатозадачність, на платі ESP8266, порівняємо загальний час виконання обчислень на самому пристрої, час використання системою ресурсів пристрою при використанні граничних обчислень, та загальний час виконання граничних обчислень системою (рис. 4.9). Для отримання часу використання системою ресурсів пристрою при граничному обчислення, було збережено час, після відправки останнього байту на обчислення та час надходження першого байту результату.

Рисунок 4.8 – Порівняння часу виконання обчислень на самому пристрої, часу використання системою ресурсів пристрою при використанні граничних обчислень та загальний час виконання граничних обчислень системою для масивів даних різної розмірності

Згідно результатів тестування, видно, що виконання граничних обчислень на менш потужних пристроях значно збільшують час виконання задачі системою. Але при розмірності даних до обчислення [64,64] і більше елементів, можна досягти зменшення використання системою клієнтом ресурсів, які можуть бути потрібними для виконання більш пріоритетних задач. Отже, граничні обчислення на менш потужних пристроях надають можливість пристрою клієнту звільнити частину обчислювальних ресурсів, проте це збільшує час виконання цих задач, тому має бути використано тільки для задач, які не потребують швидкого рішення і мають низьку частоту надходження.

Розроблено програмне забезпечення модуля комп'ютерної системи, яке дозволяє виконувати граничні обчислення без залучення додаткових пристроїв до системи пристроїв IoT. Проведене тестування показало, що при виконанні граничних обчислень на менш потужних пристроях загальний час закінчення програми буде збільшено за рахунок меншої кількості обчислювальних ресурсів і втрати часу на передачу, але за рахунок підтримки багатозадачності операційною системою, кількість ресурсів, використаних програмою, зменшиться, що може бути корисним для виконання менш пріоритетних задач пристрою IoT. При цьому при виконанні граничних обчислень на пристроях IoT однакової потужності і без підтримки багатозадачності необхідно встановити мінімальний обсяг даних, при якому час виконання обчислень перевищує час передачі даних.

Для розглянутого пристрою при виконанні матричного множення, час передачі двох масивів розмірністю [20,20] є меншим за час виконання операції матричного множення над цими даними і є мінімальним обсягом даних, при якому граничні обчислення для пристрою є більш ефективними ніж виконання обчислень на самому пристрої. При виконанні обчислень над матрицею [128,128] елементів вдалося досягти

пришвидшення виконання задачі на 32% за рахунок виконання граничного обчислення на двох пристроях IoT.

ВИСНОВКИ

Проаналізовано пристрої Інтернету Речей і при цьому доведено, що граничні обчислення все ширше використовуються пристроями IoT для вирішення проблем надмірного трафіку та затримки виконання обчислень. Показано, що в останні роки значне збільшення кількості пристроїв Інтернету Речей призводить до надмірного навантаження на транспортні шляхи мережі Інтернет. Розглянуто різні моделі побудови систем виконання граничних обчислень і різні підходи до розробки програмного забезпечення пристроїв IoT.

Розглянуто спосіб організації граничних обчислень, який використовує вільні ресурси пристроїв Інтернету Речей, що є наявними в локальній мережі і призводить до підвищення захищеності даних за рахунок зменшення трафіку, який передається з локальної мережі, та зменшення вартості виконання таких обчислень за рахунок використання вже наявних апаратних потужностей. Запропоновано підхід до організації системи реалізації граничних обчислень на локальних пристроях Інтернету Речей, який не залежить від апаратної складової використаних плат, і дозволяє додавати нові пристрої в систему виконання граничних обчислень без оновлення програмного забезпечення в модулях використаних плат.

Розроблено модуль організації граничних обчислень, який забезпечує високу швидкодію та універсальність програмного забезпечення. Показано, що при використанні платформи Python досягається найбільша сумісність програмного забезпечення з різними операційними системами, в порівнянні з мовами, що компілюються у системні команди і потребують розробки під кожну апаратну платформу та операційну систему окремо, а підтримка виклику функцій мови C дозволяє досягти найвищої швидкодії виконання обчислень.

Розроблено програмне забезпечення модуля комп'ютерної системи, яке дозволяє виконувати граничні обчислення без залучення додаткових пристроїв до системи пристроїв IoT. Проведене тестування показало, що при виконанні граничних обчислень на менш потужних пристроях загальний час закінчення програми буде збільшено за рахунок меншої кількості обчислювальних ресурсів і втрати часу на передачу, але за рахунок підтримки багатозадачності операційною системою, кількість ресурсів, використаних програмою, зменшиться, що може бути корисним для виконання менш пріоритетних задач пристрою IoT. При цьому при виконанні граничних обчислень на пристроях IoT однакової потужності і без підтримки багатозадачності необхідно встановити мінімальний обсяг даних, при якому час виконання обчислень перевищує час передачі даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Young N. Cloud Computing: A to Z of Cloud Computing / Nobert Young., 2019. – 165 с.
2. What Is Edge Computing : URL : https://www.ibm.com/cloud/what -is- edge-computing.
3. Amazon Web Services (AWS) - Cloud Computing Services : URL : https://aws.amazon.com.
4. What is edge computing and why it matters : URL: https://www.networkworld.com/article/3224893/what-is-edge-computing- and-how-it-s-changing-the-network.html.
5. Google Cloud IoT solutions : URL : https://cloud.google.com/solutions/iot.
6. What Edge Computing Means for Infrastructure and Operations Leaders : URL : https://www.gartner.com/smarterwithgartner/what-edge-computing- means-for-infrastructure-and-operations-leaders/.
7. AWS Outposts : URL : https://aws.amazon.com/outposts/.
8. AWS IoT Greengrass : URL : https://aws.amazon.com/greengrass/.
9. Jetson Nano Developer Kit : URL : https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
10. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement. Journal of Big Data 2019. № 6.
11. Sfar A. R., Zied C., Challal Y. A systematic and cognitive vision for

IoT security: a case study of military live simulation and security challenges. International Conference on Smart, Monitored and Controlled Cities. 2017. № 2. С. 101–105.

12. Gatsis K., Pappas G. J. Wireless Control for the IoT: Power, Spectrum, and Security Challenges. Proceedings of the Second International Conference on Internet-of-Things Design and Implementation. 2017. № 2.

13. Zhou J., Cap Z., Dong X., Vasilakos A.V. Security and privacy for cloud- based IoT: challenges. IEEE Commun Mag. 2017. № 55(1). С. 26–33.

14. Sfar A. R., Natalizio E., Challal Y., Chtourou Z. A roadmap for security challenges in the internet of things. Digit Commun Netw. 2018. № 4(1). С. 118–37.

15. Minoli D., Sohraby K., Kouns J.. IoT security (IoTSec) considerations, requirements, and architectures. In: Proc. 14th IEEE annual consumer communications & networking conference (CCNC), 2017.

16. Gaona-Garcia P., Montenegro-Marin C. E., Prieto J. D., Nieto Y. V. Analysis of security mechanisms based on clusters IoT environments. Int J Interact Multimed Artif Intell. 2017. № 4(3). С. 55–60.

17. Behrendt F. Cycling the smart and sustainable city: analyzing EC policy documents on internet of things, mobility and transport, and smart cities. Sustainability. 2019. № 11(3). С. 763.

18. Bare Metal vs. RTOS vs. OS: What's the Best for Your IoT Solution? : URL : https://www.nabto.com/bare-metal-vs-rtos-vs-os/.

19. What is Arduino? : URL : https://www.arduino.cc/en/Guide/Introduction.

20. NodeMCU Documentation : URL : https://nodemcu.readthedocs.io/en/release/.

21. Raspberry Pi Documentation : URL : https://www.raspberrypi.org/documentation

22. An Open Source RTOSfor IoT : URL : https://www.microcontrollertips.com/zephyr-open-source-rtos-iot-faq/.

23. Understanding The Architecture Of Barbara OS : URL : https://medium.com/iot-security-review/understanding-the-architecture-of-barbara-os-f80d64243656

24. Python Documentation : URL : https://www.python.org/doc/.

25. Cython Documentation : URL : https://cython.readthedocs.io/en/latest/.

ДЕМОНСТРАЦІЙНІ МАТЕРІЛИ