ДИПЛОМНА РОБОТА
БАКАЛАВРА

ДРБ. ІП - 59.00.00.000 ПЗ

Група ІП-19-1К

Храбатин Іван

2022

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: "Методологія сервіс-орієнтованого балансування мобільних обчислень"

(назва згідно з наказом ректора)

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 59.00.00.000 ПЗ

(позначення)

Студент

ІП-19-1К /І.Р.Храбатин/
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

к.т.н., доц. /В.В.Бандура/
(посада) (підпис) (дата) (розшифровка підпису)

Консультанти:

з питань нормоконтролю

доц., к.т.н /В.Я. Піх/
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
Завідувач кафедри

проф., д.т.н. /В.І.Шекета/
(посада) (підпис) (дата) (розшифровка підпису)

2022
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою **ІПЗ**

проф. **В.І. Шекета**

“ ____ ” _____2022 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Храбатину Івану Руслановичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи)“Методологія_сервіс-орієнтованого балансування мобільних обчислень”

керівник проекту (роботи) Бандура Вікторія Валеріївна

затверджені наказом закладу вищої освіти від **“** ___ ” _____ 2022 р. № _____

2. Строк подання студентом проекту (роботи) 17 червня 2022р.

3. Вихідні дані до проекту (роботи) Результати переддипломної практики

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

1.Огляд_існуючих_концепцій,рішень_та_сервісів_в_даній_області

2.Побудова_моделі_або_алгоритму_власного_рішення

3.Документування реалізації власного оригінального рішення вибраними засобами

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Загальна структура системи для розвантаження (рис.2.1)

2. Пропонована структура системи для розвантаження (рис.2.2)

3. Монітор контексту (рис.2.3)

4. Система прийняття рішень (рис.2.4)

5. Менеджер зв’язку та планувальник розвантаження (рис.2.5)

6. Загальна блок-схема системи розвантаження (рис.2.6)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	01.02.2022	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	01.03.2022	виконано
3	Побудова моделі або алгоритму власного рішення	01.04.2022	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	01.06.2022	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	10.06.2022	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

В дипломній роботі запропонована система як цілісна архітектура з використанням сервіс-орієнтованої архітектури, в якій обчислювальні та інтенсивні завдання окремих додатків вивантажуються з мобільного пристрою на хмарний сервер. У цьому методі завдання виконуються віддалено як індивідуальний сервіс на хмарному сервері. Крім того, фреймворк розроблено для прийняття рішень щодо розвантаження на основі контексту пристрою.

КЛЮЧОВІ СЛОВА: СЕРВІС-ОРІЄНТОВАНА АРХІТЕКТУРА, МОДЕЛЬ, ФРЕЙМВОРК, МОБІЛЬНІ ОБЧИСЛЕННЯ, ХМАРНИЙ СЕРВЕР, ПРОЦЕС БАЛАНСУВАННЯ.

ANNOTATION

The thesis proposed a system as a complete architecture using a service-oriented architecture, in which computational and intensive tasks of individual applications are offloaded from a mobile device to a cloud server. In this method, tasks are performed remotely as an individual service on a cloud server. Additionally, the framework is designed to make offloading decisions based on device context.

KEY WORDS: SERVICE ORIENTED ARCHITECTURE, MODEL, FRAMEWORK, MOBILE COMPUTING, CLOUD SERVER, BALANCING PROCESS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 7
ВСТУП 8

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ 9
1.1 Мобільні обчислення 9
1.2 Мобільна хмарна екосистема 10
1.3 Мобільні хмарні обчислення 12
1.4 Проблеми розвантаження обчислень для мобільних хмарних обчислень 15
1.5 Постановка проблеми дипломної роботи 16
1.6 Методи обчислювального розвантаження 18
1.7 Опис основного методу дослідження 21
1.8 Аналіз існуючих технічних робіт по предметній області 30

2 АЛГОРИТМІЧНІ ТА МАТЕМАТИЧНІ РІШЕННЯ ДЛЯ ЗАДАЧ БАЛАНСУВАННЯ МОБІЛЬНИХ ОБЧИСЛЕНЬ 33
2.1 Загальна структура системи 33
2.2 Пропонована обчислювальна система розвантаження 34
2.3 Представлення основного алгоритму системи 41

3 РЕАЛІЗАЦІЯ МОРДЕЛЕЙ ТА ФРЕЙМВОРКУ ДЛЯ ЗДІЙСНЕННЯ БАЛАНСУВАННЯ ПРОЦЕСІВ В МОБІЛЬНИХ ОБЧИСЛЕННЯХ 43
3.1 Інформація про програмне забезпечення 43
3.2 Набори даних, що використовуються при реалізації 43
3.3 Розробка системи для самокерованих транспортних засобів 45
3.4 Реалізація Context Monitor 51

3.5 Реалізація механізму прийняття рішень 53
3.6 Процес використання Communication Manager і Offloading Planner 61

ВИСНОВКИ 63
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА 65
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

МО – мобільні обчислення

БД – База даних

СУБД - Система управління базами даних

ПЗ - Програмне забезпечення

ВСТУП

Використання та розгортання платформ і програм для смартфонів феноменально зросли по всьому світу. Ми використовуємо мобільні пристрої для різних завдань, таких як спілкування різними способами, надсилання електронних листів, онлайн-банкінг, перегляд Інтернету, перегляд відео, використання соціальних мереж та навігація за допомогою онлайн-карт.

Для виконання цих завдань доступні різні програми. Через зростання кількості мобільних додатків і вимог користувачів якість обслуговування перешкоджають обмеження на мобільній стороні, такі як обмеження ресурсів, обмежена енергія, обмежене доступне підключення та спільне бездротове середовище.

Оскільки ресурсів мобільних пристроїв недостатньо для виконання інтенсивних обчислень і ресурсомістких завдань, таких як розпізнавання зображень та обробка природної мови, використовується концепція мобільних хмарних обчислень. Мобільні хмарні обчислення усувають ці обмеження ресурсів, включаючи хмарні обчислення в мобільне середовище.

Це робиться за допомогою Computational Offloading, що включає в себе можливість виконання ресурсомістких програм на віддалених хмарних серверах.

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Мобільні обчислення

Мобільні обчислення – це технологія, яка дозволяє передавати дані, голос і відео за допомогою такого пристрою, як ноутбук, без підключення до фіксованого фізичного з’єднання [1].



Рисунок 1.1 - Мобільні обчислення

Існують три основні концепції мобільних обчислень – портативне обладнання, мобільне програмне забезпечення та мобільний зв’язок. Портативне обладнання включає мобільні телефони та інші доступні на ринку гаджети, які можуть підтримувати мобільні обчислення на пристроях. Прикладами такого портативного обладнання є смартфони, планшетні ПК, ноутбуки тощо. Мобільне програмне забезпечення включає різні програми, такі як веб-браузери, карти та ігри. Мобільний зв'язок включає канали зв'язку, такі як Wi-Fi, Bluetooth і стільникові мережі. Існують численні програми мобільних обчислень, як показано на рисунку 1.2, якщо вони використовуються з бездротовими мережами. Однак існують різні обмеження для мобільних

обчислень, як-от обмеження ресурсів, мобільне підключення, обмежена енергія та безпека.

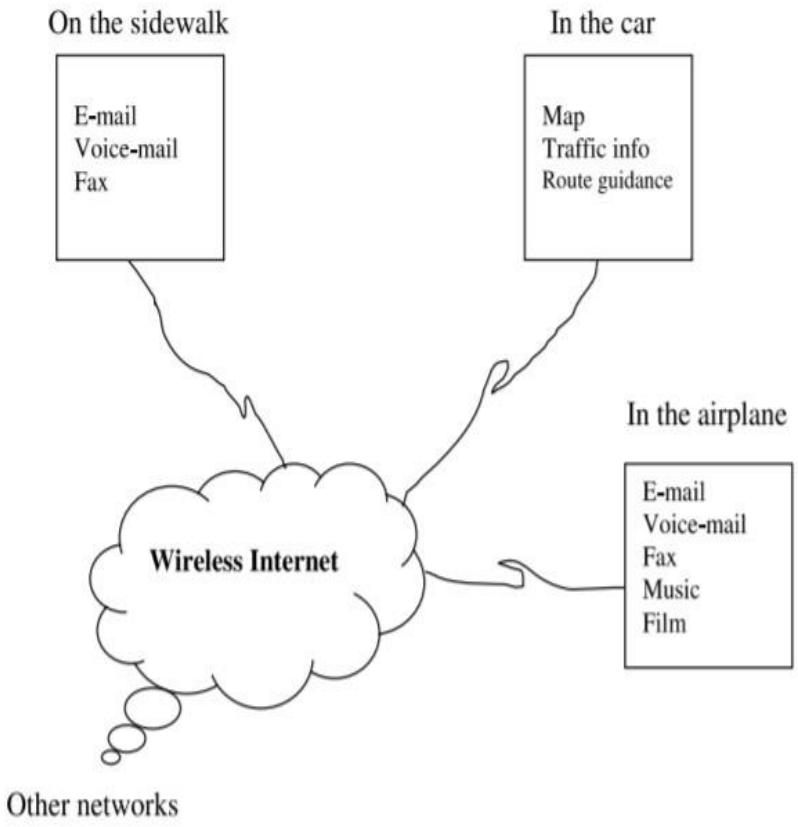


Рисунок 1.2 - Застосування мобільних обчислень

1.2 Мобільна хмарна екосистема

У цьому розділі ми пояснюємо мобільну хмарну екосистему та її різні компоненти. Ми також обговорюємо якість досвіду (QoE) користувача для вивчення впливу екосистеми мобільної хмари. У цій екосистемі мобільна система і хмарна система з'єднані мережею. Отже, екосистема складається з трьох окремих компонентів [2]:

1. Мобільна система
2. Мережа
3. Хмарна система

На QoE користувача впливають три вищезгадані компоненти екосистеми мобільної хмари, як показано на рисунку 1.3. QoE, у свою чергу, має чотири компоненти – використання пам’яті, споживання енергії, грошові витрати та швидкість передачі даних. Параметри середовища в екосистемі мобільної хмари поділяються на три компоненти залежно від джерела варіації – програми, мережа та платформа виконання.

Програми

Деякі мобільні програми вимагають обчислень і ресурсів, наприклад, розпізнавання об’єктів, реконструкції карти, пошуку зображень тощо, тоді як інші програми, як-от текстовий редактор, веб-браузер тощо, ні. Ці програми висувають різні вимоги до мобільної системи. Обробка цих змін для мобільної системи є серйозною проблемою.

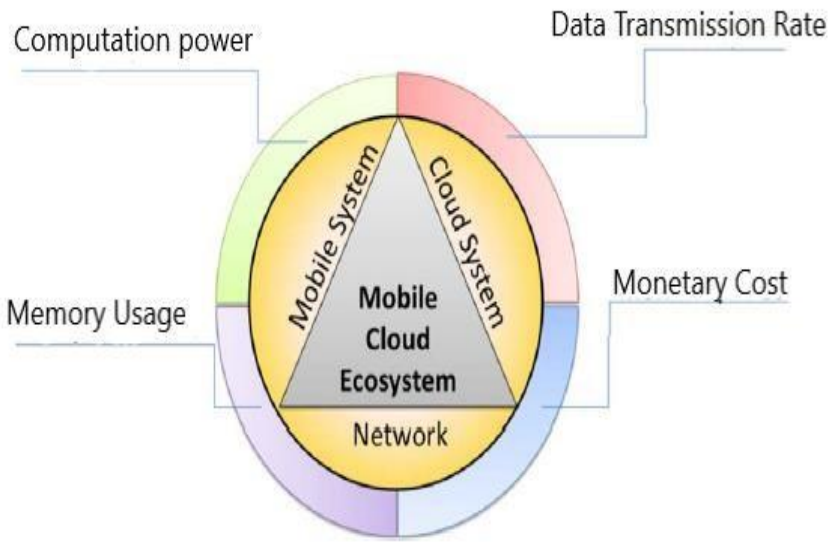


Рисунок 1.3 - Мобільна хмарна екосистема

Мережа

Сьогодні існує кілька типів бездротових мереж (стільникова мережа, Wi-Fi, Bluetooth). Однак бездротові мережі не є надійними та стабільними, оскільки розташування мобільних пристроїв час від часу змінюється. Крім того, під час руху мобільних пристроїв потужність сигналу різна.

Мобільні системи повинні адаптуватися до цих змін і продовжувати безперебійно функціонувати.

Платформа виконання

Виконавча платформа відноситься до мобільних пристроїв і віддалених хмарних серверів. Організація та обладнання мобільних пристроїв і хмарних систем мають ряд варіацій. Мобільна система повинна бути адаптивною до цих змін, оскільки неможливо розробити окрему систему для кожної конфігурації.

1.3 Мобільні хмарні обчислення

Важко використати весь потенціал мобільних комп'ютерів, незважаючи на їх зростання. Це пов'язано з його внутрішніми проблемами, такими як дефіцит ресурсів, часті відключення та мобільність. Ці проблеми можна вирішити за допомогою мобільних хмарних обчислень, виконуючи мобільні додатки на зовнішніх ресурсах, тобто в хмарі. Розширюючи можливості зберігання та обчислення мобільних пристроїв, використовуються хмарні ресурси, а отже, мобільні хмарні обчислення використовуються для розширення екосистеми мобільних хмар [3].

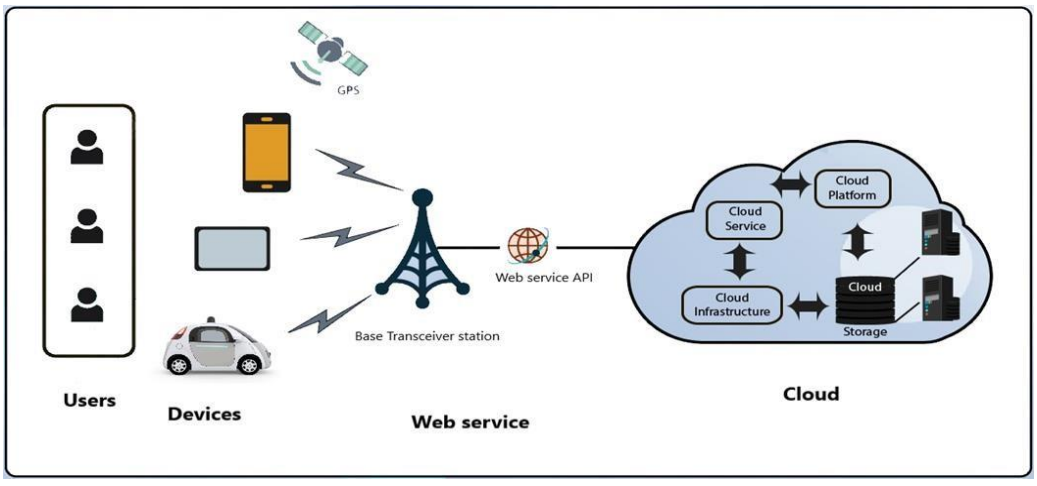


Рисунок 1.4 - Мобільні хмарні обчислення

Мобільні хмарні обчислення: сервіс-орієнтована архітектура

Сервісно-орієнтована архітектура (SOA) — це архітектура, яка використовується для розробки розподілених і сумісних додатків [4]. Розподілена програма має компоненти програми, які працюють у різних вузлах комп'ютера. Іншими словами, програма (комп'ютер або Інтернет), яка працює в одному місці в одній системі і використовує деякі функції або бізнес-логіку з іншої системи десь у світі. Інтероперабельна програма означає одну програму, розроблену однією мовою; наприклад, Python, може спілкуватися з іншим додатком, розробленим з використанням іншої мови, наприклад C#. Підхід, орієнтований на обслуговування, може забезпечити зниження вартості мережі. Накладні витрати на зв'язок також можуть бути значно зменшені, оскільки мережеві транзакції в основному містять лише інформацію, пов'язану із завданням, а не весь її код. Отже, SOA можна використовувати для надання послуг хмарних обчислень мобільним додаткам, де функціональні можливості завдань надаються як сервіси [5].

Мобільні хмарні обчислення: розвантаження обчислень

Як згадувалося раніше, користувач очікує, що мобільна система запускатиме різноманітні програми. Однак мобільна система обмежена різними параметрами в будь-який момент часу, такими як обмеження пам'яті, обмежена обчислювальна потужність існуючих мобільних процесорів тощо. Розвантаження обчислень має на меті дозволити використання пам'яті або інтенсивних обчислень у мобільних системах шляхом розподіленого виконання мобільні додатки. Це досягається шляхом запуску частини програми з мобільного пристрою на віддалені обчислювальні ресурси, такі як хмарний сервер. Щоб забезпечити розподілене виконання мобільного додатка, фреймворк розвантаження повинен визначити, як розділити будь-яку програму для планування на мобільному пристрої та хмарних серверах [2].

Незважаючи на те, що обчислювальна розвантаження виникла приблизно в 1970-х роках, її потенціал був широко вивчений лише після того, як бездротовий зв'язок та швидкість Інтернету були достатньо покращені та змогли його підтримувати [53]. Потужність розвантаження залежить насамперед від таких технологій, як стільникові мережі та мережі WiFi, оскільки ці мережі визначають доцільність обчислювального розвантаження. У сучасну епоху це було помічено в різних сценаріях, коли Wi-Fi здатний забезпечити з'єднання з високою пропускною здатністю. Більше того, мережі 5G приділяється багато уваги, оскільки вона може підтримувати збільшення пропускної здатності мережі та низьку затримку, а також відкриває шлях для MCC для вирішення проблеми розвантаження обчислень.

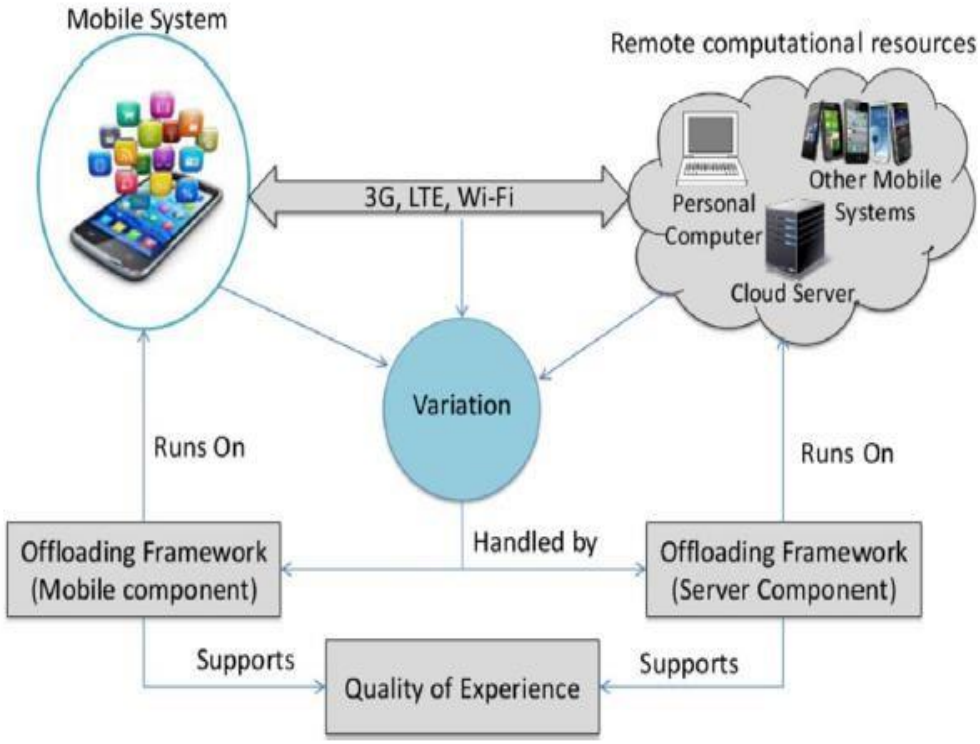


Рисунок 1.5 - Розвантаження обчислень

Отже, за допомогою цих бездротових мереж розвантаження додатків здійснюється до хмарних ресурсів. На рисунку 1.5 представлена загальна архітектура обчислювального розвантаження. Варіанти, згадані раніше в

Розділі 1.2, такі як програми, мережа та платформа виконання, слід обробляти під час розвантаження. Також слід підтримувати якість досвіду користувача.

1.4 Проблеми розвантаження обчислень для мобільних хмарних обчислень

Виходячи з вимог мобільних пристроїв, враховуються такі проблеми [12]:

1. Підключення до мережі та відмовостійкість

Оскільки мобільні пристрої постійно рухаються, свобода пересування та автономність спілкування під час використання мобільних хмарних сервісів є ключовими критеріями задоволеності користувачів. Однак під час руху існують деякі обмеження, які перешкоджають безперебійному підключенню та безперебійному доступу до хмарних служб. Швидкість обміну даними та пропускна здатність мережі відрізняються в міру подорожі користувачів мобільного зв'язку. Крім того, під час надсилання чи отримання даних користувачі втрачають зв'язок у деяких місцях; таким чином, слід передбачити відповідні відмовостійкі стратегії для підходів до розвантаження, щоб повторно надіслати втрачені компоненти та зменшити час реагування.

2. Автоматичний механізм

Все ще існує потреба в автоматизації існуючих обчислювальних систем розвантаження. Завдяки автоматизації процес розвантаження виконується без проблем, враховуючи навколишнє середовище. Виконати цей тип автоматизації непросте завдання, оскільки потрібен протокол для визначення та виявлення служб відповідно до поточного контексту та його обмежень.

3. Різноманітна платформа

Різноманітність і неоднорідність архітектур мобільних пристроїв і операційних систем є одними з проблем у нинішніх платформах розвантаження обчислень. Послідовний доступ до хмарних служб потрібен там, де мобільні пристрої можуть отримати доступ до послуг хмарних обчислень незалежно від

встановленої операційної системи або використовуваного обладнання. Стандартизована система розвантаження для різних мобільних платформ досі залишається складною проблемою в області МСС.

Вартість розвантаження

Використання хмарних ресурсів накладає фінансові витрати на кінцевих користувачів, які зобов'язані платити відповідно до Угоди про рівень обслуговування (SLA), ухваленої з постачальником хмари, який підтримує їх. Операції з розвантаження вмісту та передачі даних між хмарними провайдерами зазвичай спричиняють додаткові витрати для кінцевих користувачів. Тому при прийнятті рішень щодо розвантаження слід враховувати економічні фактори.

1.5 Постановка проблеми дипломної роботи

Оскільки контекст мобільного пристрою час від часу змінюється, стратегія обчислювального розвантаження повинна динамічно коригуватися, щоб досягти найкращої енергоефективності, отримати найкраще споживання продуктивності та зменшити грошові витрати відповідно до зміни цієї інформації про контекст. . Крім того, через притаманну складність контексту, на якому працюють мобільні пристрої, стратегія обчислювального розвантаження повинна адаптуватися автоматично, а не вручну. Як обговорюється в огляді літератури (розділ 2), для розвантаження використовуються різні методи. Однак деякі з них є ручними, деякі не є гнучкими, а деякі є просто опитуваннями, які не містять конкретних формул, які можна використовувати для розвантаження. Для вирішення цих проблем, викликаних змінами в контексті мобільних пристроїв, пропонується фреймворк розвантаження на основі SOA з використанням мобільних хмарних обчислень (МСС). Ця структура адаптується і приймає правильне рішення щодо стратегії розвантаження відповідно до змін в інформації про контекст.

Існує механізм вибору послуг, який вибирає потрібну послугу з кількох послуг.

Впровадження підключення мобільного пристрою до Інтернету дає можливість перевантажувати інтенсивні обчислення та ресурсомісткі завдання з мобільного пристрою на віддалені хмарні сервери для ефективного виконання. Однак, як згадувалося в Розділі 1.4, існує кілька проблем у розвантаженні обчислень для мобільних хмарних обчислень, таких як мережеве підключення та відмовостійкість, автоматичний механізм, різноманітна платформа та вартість розвантаження. Через ці обмеження жоден з існуючих підходів не здатний вирішити всі проблеми, а отже, не дозволяє користувачам повністю використовувати потенціал пристроїв. Отже, ця теза охоплює проблеми, згадані вище, шляхом створення системи обчислювального розвантаження для мобільних пристроїв на основі SOA.

Хмарні обчислення. Ця нова система для впровадження мобільних хмарних обчислень на мобільні пристрої створює переваги за рахунок віддаленого виконання різноманітних завдань додатків, які надаються як послуги. Ця система є вигідною для кінцевих користувачів з точки зору зниження витрат, швидшої та кращої продуктивності.

Основний внесок цієї роботи можна підсумувати таким чином:

1. Створено цільову функцію для визначення місця виконання служби:

$$OF = P(s1,s2,...,sn)*WP + M(s1,s2,...,sn)*WM + E(s1,s2,...,sn)*WE,$$

де s представляє послуги, P являє собою продуктивність (обчислення), M представляє пам'ять, E представляє економічні витрати, WP представляє вагове значення продуктивності, WM представляє вагове значення пам'яті і WE представляє вагове значення економічної вартості.

2. Створено фреймворк розвантаження. Залежно від адаптивної цільової функції та інформації про контекст, що змінюється, ця структура

розвантаження створює найбільш відповідну стратегію розвантаження в певний час.

3. Перетворені різні самокеровані модулі в систему на основі SOA для кращого обслуговування та легшого розширення.

4. Проведено ряд експериментів із системою на основі SOA, щоб переконатися, що адаптивна структура розвантаження, адаптивна цільова функція та механізм вибору адаптивного сервісу досягають найкращої ефективності споживання електроенергії, отримують задовільні продуктивності та знижують грошові витрати відповідно до змін контекст, на якому виконується мобільний пристрій, наприклад мобільні телефони, автомобілі тощо.

У цьому розділі пояснюються різні методи, які можна використовувати для реалізації обчислювального розвантаження. У цій главі далі обговорюються різні роботи, виконані на даний момент з використанням обчислювальної техніки розвантаження.

1.6 Методи обчислювального розвантаження

Міграція коду — це метод обчислювального розвантаження, при якому вихідний код вивантажується на віддалені машини під час виконання. Існує кілька робіт, які використовують цей метод. Одним із них є MAUI [6]. У MAUI окремі анотовані частини коду вивантажуються в проміжне поле. Для прийняття рішення щодо визначення розділів, які необхідно вивантажити, MAUI використовує концепцію цілочисельного лінійного програмування (ILP). Лінійне програмування — це математична модель, яка використовується для розв’язування задач прийняття рішень, що мають багато змінних рішень, які обмежені набором обмежень. ILP — це підгалузь лінійного програмування (LP), де змінні рішення обмежені, щоб утримувати цілі значення. Іншою роботою, яка використовує Code Migration, є ThinkAir [7], в якій автор виділяв віртуальні

машини для виконання розділів коду програми під час виконання. Однак це спричиняє великі витрати на керування, якщо віртуальні машини вводяться в експлуатацію під час виконання. У COSMOS: розвантаження обчислень як сервіс для мобільних пристроїв [8] розподіл завдань керується та вивантажується в хмарний екземпляр на віртуальних машинах. Однак цей метод налаштований лише для процесора Android x86.

Розвантаження шляхом реплікації

Розвантаження шляхом реплікації — це метод, за якого характеристики мобільного пристрою клонуються на віддалені машини. CloneCloud [9] запропонував цей метод для оптимізації обчислень мобільного пристрою. В основному він зосереджений на покращенні продуктивності та терміну служби акумулятора. Для розділення додатків вони використовували деталізацію на рівні потоків у цьому фреймворку. Статичний аналіз програми для виявлення обмежень міграції та профілювання програми для побудови моделей витрат об'єднані, щоб знайти компоненти, які можна вивантажити. У цій роботі рішення про розвантаження приймається під час виконання, а клонування виконується шляхом міграції потоків з мобільних пристроїв у хмару. Однак це неефективний процес для клонування вже існуючих компонентів. Іншою роботою, яка реплікує виконання коду на віддаленому сервері або хмарі та мобільному пристрої, є Tango [10]. Він повторює, а потім вибирає той, який має мінімальний час відповіді. Хоча повторення того ж є непрактичним процесом.

Інший метод розвантаження – це розміщення та планування. У цій техніці Mahmoodi et al. [11] розробив аналітичну структуру для планування розвантаження, але практичність під питанням, заснована на припущеннях про умови мережі. Враховуючи ці нездійсненні припущення, можна сказати, що краще вирішувати проблему з системної точки зору і будувати практично життєздатні механізми.

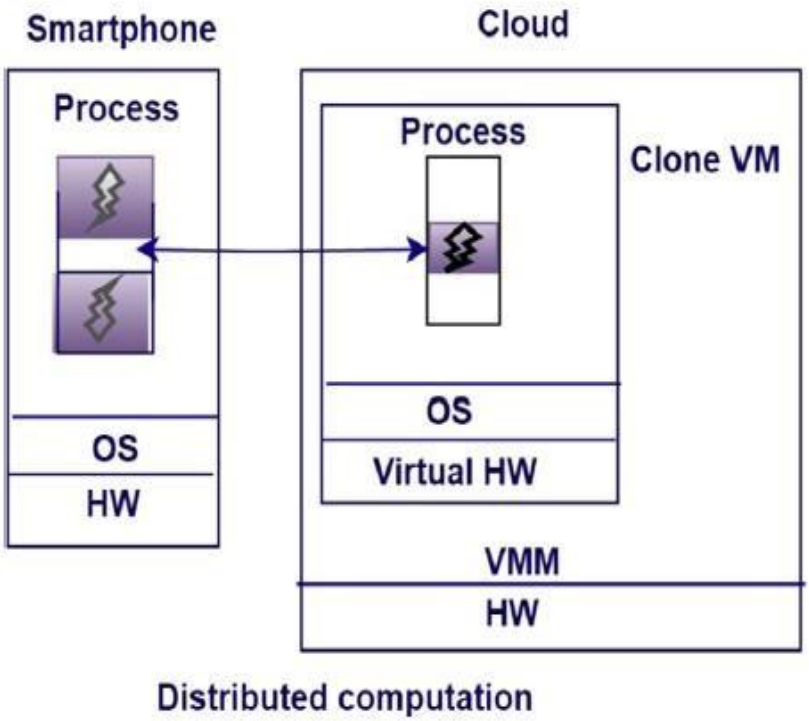


Рисунок 1.6 - Модель виконання CloneCloud

Дизайн зозулі

Конструкція зозулі була запропонована для вивантаження зі смартфонів, які працюють на платформі android, у хмару [12]. У цьому методі автор використав модель-заглушку Java для розвантаження, яка інтегрує інструмент розробки Eclipse з фреймворком Android з відкритим кодом. Ця модель підтримує як локальне, так і хмарне виконання коду та працює на вже існуючій моделі діяльності/служби в Android. Використовуючи це, він розрізняє інтенсивні та неінтенсивні компоненти програми. У цьому дизайні виконується статичний розподіл коду, при якому частина коду, яку необхідно вивантажити на віддалений сервер, попередньо встановлюється мобільними пристроями.

Однак можна сказати, що цей метод є невдалим у сучасному гетерогенному мережевому середовищі. Тому, щоб впоратися з цією неоднорідністю, було введено динамічне розбиття коду.

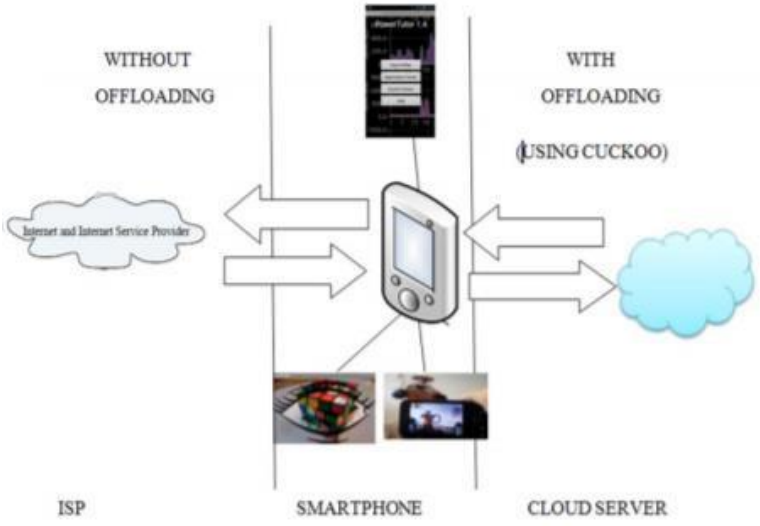


Рисунок 1.7 - Розвантаження за допомогою Cuckoo Framework

1.7 Опис основного методу дослідження

Багато попередніх робіт були зосереджені на розвантаженні обчислень для мобільних хмарних обчислень. Деякі статті використовують функцію/формулу для визначення місця, де потрібно виконати завдання. Наприклад, Ковачов та ін. [20] надають проміжне програмне забезпечення MACS (Mobile Augmentation Cloud Services), яке дозволяє адаптивно розширювати виконання додатків Android з мобільного пристрою в хмару. Проміжне програмне забезпечення відповідає за важку роботу з розділення програми, моніторингу ресурсів і розвантаження обчислень. Мобільні програми є еластичними, вони можуть працювати як стандартні мобільні програми, а також можуть досягати прозоро віддалених обчислювальних ресурсів. У програмі MACS є ядро програми, яке не можна вивантажити, і є кілька служб (Si), які інкапсулюють окремі функціональні можливості програми, які можна вивантажити (SRi). Для кожної служби профільовані такі метадані:

Вартість пам'яті – споживання пам'яті послуги,

Тип – можна вивантажити чи ні,

Розмір передачі – кількість даних для передачі,

Розмір надсилання – кількість даних, які потрібно надіслати, розмір отримання – кількість даних, які потрібно отримати,

Розмір коду – розмір скомпільованого коду.

Для розвантаження цей підхід розглядає k розвантажуваних модулів. Для цих модулів розмір передачі становить tr1, tr2, … trk. Розмір надсилання – send1, send2, … sendk, а розмір отримання – rec1, rec2, … reck. Функція вартості зображується таким чином:

$$\min x\ (ctransfer * wtr + cmemory * wmem + ccpu * wcpu)\ 0{,}1$$

Функція вартості використовується для прийняття рішення про виконання модуля локально або віддалено. Однак ця структура не передбачає мінімізації вартості виконання та невизначеності мережі.

Враховуючи обмеження MACS, Deng et al. [7] пропонує структуру розвантаження, в якій викликаються служби в робочих процесах, і приймається рішення про те, розвантажувати служби чи ні. У цій роботі пропонується система розвантаження для вирішення проблеми нестабільного підключення мобільних пристроїв до мережі. Цей підхід зосереджений на оптимізації споживання енергії та часу виконання мобільних послуг. Система розвантаження заснована на генетичному алгоритмі. У класі еволюційних алгоритмів генетичні алгоритми базуються на концепції «виживання найсильнішого» [2]. Рекомбінуючи найкращі альтернативи популяції та мутуючи їх час від часу, можна вирішити надзвичайно складні питання, для яких інакше написання програм було б безнадійно складним. Для прийняття рішень щодо розвантаження обчислень він використовує систему з підтримкою мобільності та відмовостійкість. Основна мета планувальника розвантаження — запропонувати оптимальну стратегію розвантаження, яка мінімізує споживання енергії та час виконання для виконуваних мобільних додатків (робочих процесів). Функція F(m) використовується в стратегії розвантаження

для прийняття рішень. F(m) для кожного мобільного пристрою m визначається як:

$$F(m) = wm \times Lm + (1 - wm) \times Em$$

Тут wm — це ваговий коефіцієнт, Lm — загальний час виконання, а Em — загальна споживана енергія під час виконання робочого процесу.

Автор висвітлив недоліки MACS, а також запропонував алгоритм для оптимізації відмовостійкості розвантаження. Однак алгоритм передбачає рух користувача, але є й інші можливості.

Оскільки є кілька припущень, зроблених у попередній роботі, Ashok et al. [5] запропонував архітектуру, де автомобільні додатки віддалено виконуються в хмарі, а завдання надаються як сервіси. Автори цієї роботи також розглянули нестабільність мережі, яка була недоліком фреймворка MACS. У цій статті, по-перше, визначаються проблеми хмарних обчислень у транспортних засобах. Проблеми – це неоднорідне середовище бездротової мережі; різна програмно-апаратна архітектура бортового блоку (OBU) автомобіля та хмари; плавне розвантаження завдань. У цьому підході сервіс-орієнтований підхід використовується для надання додаткам послуг хмарних обчислень. Ідентифікуються задачі, що інтенсивні на обчислення та дані, а потім використовується фреймворк розвантаження, в якому завдання, позначені як такі, що вивантажуються, виконуються як служби та надаються в хмарі. Платформа онлайн-розміщення використовується для прийняття рішень щодо перенесення завдань у хмару на основі набору змінних і набору витрат. Змінними є швидкість мережі, доступність сервера, модулі, що вивантажуються, та інші параметри оптимізації. Витрати включають час виконання модулів, місце для зберігання, використання ЦП та витрати енергії. Для розвантаження фреймворк виконується двома кроками: профілювання та кондиціонування. На етапі профілювання, така інформація, як розміри вхідних (вихідних) даних до (з) кожного модуля, зображена як din (dout), швидкість

висхідного (Wup) і низхідного каналу (Wdown), час виконання модуля на транспортному засобі (e) і хмара (e*) і отриманий розмір сховища коду програми модуля, зображений як s. На етапі кондиціонування ці параметри використовуються для отримання коефіцієнта часу виконання модуля E, який виглядає наступним чином:

$$E = \frac{e* + \frac{din}{Wup} + \frac{dout}{Wout}}{e}$$

На основі значення E приймається рішення про позначення модуля як розвантажуваного. Однак модулі, які потрібно вивантажити, статично позначені розробником. Отже, розвантаження не залежить від фактичної інформації про контекст.

Chen [13] запропонував структуру для контекстно-залежної розвантаження обчислень у мобільних хмарних обчисленнях, яка намагається надати рішення для попередньої статті. Як у попередньому документі Ashok et al. [5] не виконував розвантаження на основі контекстної інформації, ця стаття підтримує програми з можливістю розвантаження обчислень з урахуванням контексту. Він заснований на роботі мобільних додатків. У цьому підході, щоб уможливити вивантаження програми, пропонується шаблон проектування. Потім для вибору хмарного ресурсу відображається оціночна модель. Після цього представляється фреймворк, який реалізується на стороні клієнта та сервера для підтримки вищезгаданого шаблону проектування та моделі оцінки. У цьому контексті під час розробки програми методи класифікуються за різними категоріями. Перша категорія — це прив’язані програми, які взаємодіють з пристроями введення або виведення та зовнішніми службами, або які реалізують інтерфейс програми (користувацький інтерфейс). Друга категорія — це мобільні додатки, які можуть працювати на різних платформах виконання, як-от мобільний пристрій або хмарний сервер. Параметрами, які розглядаються для оцінки, є споживання енергії батареї та час виконання програм. Недоліком цього методу є те, що не використовується сервісно-

орієнтована архітектура. Таким чином, розподілені та взаємодіючі програми не підтримуватимуться.

Щоб скористатися перевагами SOA, Хані та ін. [8] представили безпечну енергоефективну та якісну архітектуру (EEQoSA) для мобільного хмарного хэндовера. Це сервісно-орієнтована архітектура, яка складається з чотирьох шарів:

Прикладний шар

Цей рівень складається з сервера домашнього абонента (HSS), який з'єднується з серверами хмарних обчислень як корпоративний сервер. Щоб підтримувати передачу даних, HSS підключається до рівня мультимедійної підсистеми протоколу Інтернет (IPMS). Він включає в себе сервер інформації про підписку (SRI), сервер диспетчера ролей і сервер оновлення розташування (LU). Перш ніж передавати дані хмари законному користувачеві мобільної хмари, виконується шифрування для безпечного передачі даних

2. Рівень IPMS Цей рівень надає корисні послуги, як-от перегляд веб-сторінок, електронна пошта, відео на вимогу, відеоконференції та Інтернет-послуги. Він включає процес реєстрації, який допомагає отримувати оновлену інформацію про місцезнаходження від користувача мобільної хмари. IPMS використовує функцію керування сеансом виклику, щоб прив'язати загальнодоступний ідентифікатор користувача до IP-адреси користувача мобільної хмари.

Комунікаційний рівень

Він направляє дані та синхронізує шари медіа та IPMS. Сервер LU запускає процес перереєстрації, що складається з двох рівнів - періодичної перереєстрації (PRR) і повторної реєстрації для зміни можливостей (RRCC). Ці рівні потребували процесу обміну повідомленнями для завершення перереєстрації. Тому наприкінці процесу хэндовера необхідно розрахувати енергію для реєстрації, щоб визначити залишок потужності мобільного пристрою.

2. Медіа з шаром підключення

Він складається з агента медіа-ресурсів (MRA) і контролера функцій медіа-ресурсів (MRFC). MRA контролює наявну інформацію про функції медіаресурсів і передає відповідну інформацію на сервер аутентифікації. MRFC об’єднує потоки медіа та керує спільними ресурсами. Це дозволяє лише автентичним користувачам завершити процес перереєстрації після початку передачі і зменшує виникнення тривалих затримок під час передачі.

Ця архітектура забезпечує безпеку передачі даних і гарантує якість обслуговування; однак для продуктивності враховується лише енергоефективність.

Оскільки всі ці попередні роботи, які використовують різні цільові функції та алгоритми для прийняття рішень, мають багато обмежень, ми розробили структуру, яка враховує всі параметри і може працювати надійно. Нижче наведено оглядові документи, які ми переглянули для нашого дослідження, щоб отримати інформацію про різні методи, які використовуються для розвантаження, проблеми, з якими стикаються, та шляхи їх вирішення.

У [2] представлено огляд найсучасніших адаптивних алгоритмів, які використовуються для обчислювального розвантаження. Сутності описані за допомогою мобільної хмарної екосистеми. Потім вони використовуються для визначення різних джерел змін у системі. Завдяки мобільності мобільного пристрою параметри змінюються під час роботи. Таким чином, екосистема мобільних хмар використовується для визначення параметрів і впливу цих змінних параметрів. Різні рішення класифікуються для адаптивного розвантаження на основі різних параметрів та їх адаптивних рішень. Для різних середовищ розвантаження існують показники якості досвіду, включаючи економію енергії, грошові витрати тощо. Нарешті, надається вплив цих параметрів на якість досвіду користувача.

Інша робота [5] надає огляд та майбутні напрямки технологій розширення, які можна використовувати для мобільних хмарних обчислень. Методи розширення — це обчислювальні моделі та рішення для аутсорсингу обчислень і зберігання мобільних пристроїв на більш потужні обчислювальні ресурси, які можуть покращити обчислювальну потужність та енергоефективність мобільного пристрою. Отже, ці методи використовуються для збільшення, покращення та оптимізації обчислювальних можливостей мобільних пристроїв. Він пропонує розгорнуту таксономію та огляд сучасних методів розширення мобільних хмар і платформ для обчислень і зберігання. Наведено таксономію методів аугментації обчислень, як показано на рисунку 1.8.

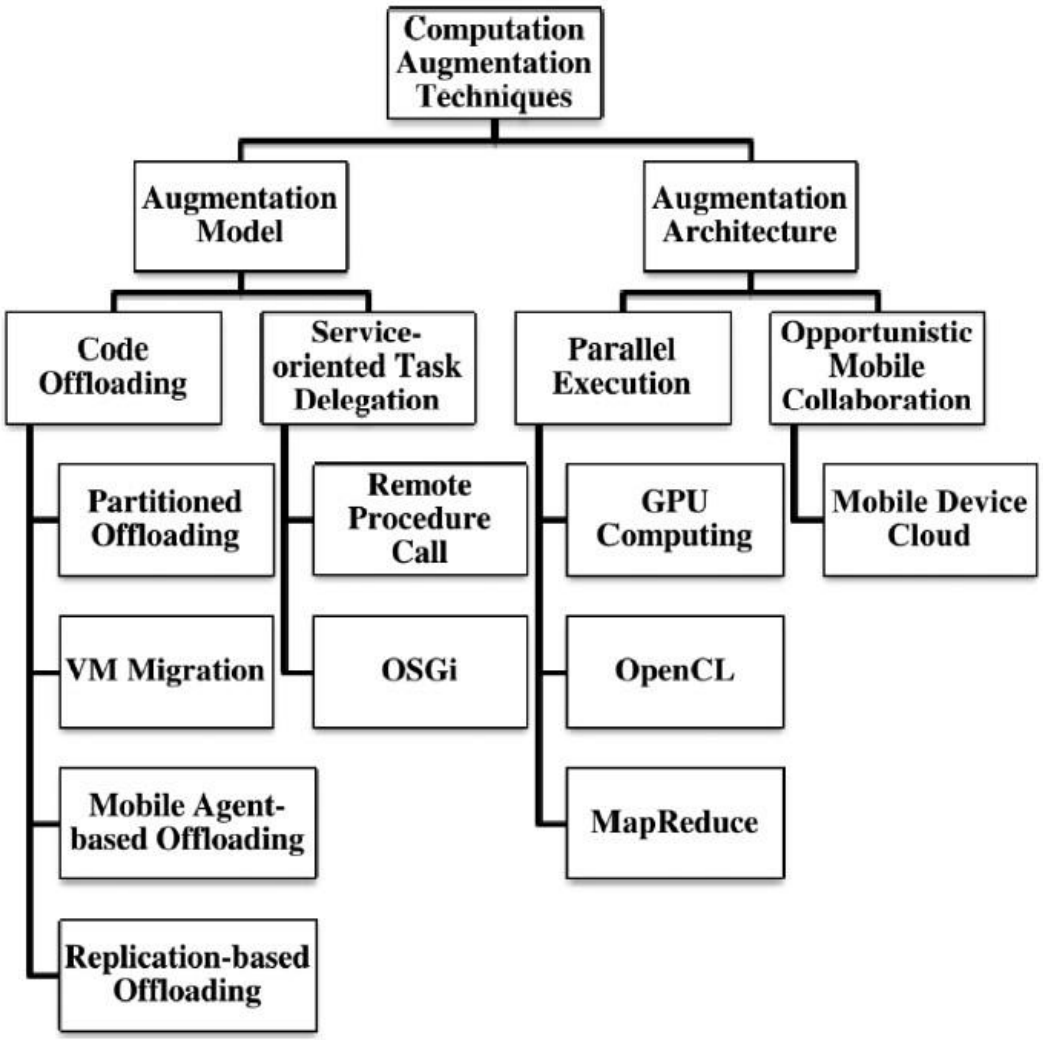


Рисунок 1.8 - Таксономія методів збільшення обчислень

У цій таксономії розглядалися методи та підходи, які використовуються в розширенні мобільної хмари для об’єднання гібридних хмарних ресурсів у загальний пул ресурсів мобільних пристроїв, який забезпечить надійний та енергоефективний аутсорсинг обчислень через мобільну хмару як послугу.

Далі йде таксономія розширення сховища, яка показана на рисунку 1.9, де обговорюється архітектура, орієнтована на дані для зберігання даних у хмарах та хмарі мобільного пристрою. Він також вивчає різні життєво важливі питання, як-от захист даних і взаємодію даних.

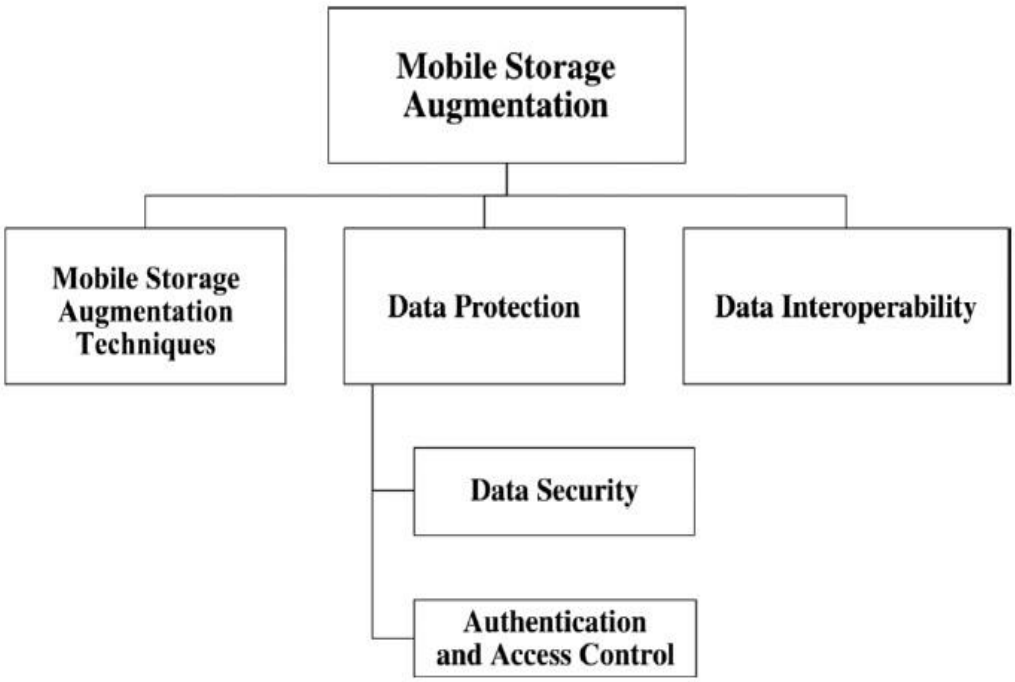


Рисунок 1.9 - Таксономія збільшення мобільного сховища

В опитуванні також проаналізовано різні суттєві технічні прогалини для подальшого вивчення.

Роота [4] фокусується на сталому розвантаженні в мобільних хмарних обчисленнях. У цьому огляді наведено порівняння найсучасніших робіт, пов’язаних з розвантаженням з урахуванням енергоспоживання в рамках розвантаження на основі МСС. Вони зібрали наявні дослідження та

класифікували їх відповідно до точки зору мобільного пристрою та продуктивності хмари.

Ці аспекти спостерігаються з точки зору обробки розвантаження з урахуванням енергії та компромісу між резервуванням енергії та ефективністю виконання. Він містить інформацію про алгоритмічне проектування та реалізацію різних методик. Критичний аналіз методів розвантаження показує, що для класифікації відкритих проблем можна використовувати наступні три функції:

1. Розбиття завдань

Функціональність розділення завдань заснована на сумісності та зручності використання. Оскільки апаратне забезпечення мобільного пристрою та хмари відрізняється, обробляти мобільні додатки в хмарі важко. Крім того, оскільки продуктивність розвантаження завдань офіційно не визначена, вона оцінюється розробниками, а розподіл завдань здійснюється за допомогою підтримуваних інтерфейсів прикладного програмування (API). Щоб використовувати традиційні додатки для виконання в рамках потоку мобільних хмарних обчислень, потрібен значний обсяг роботи.

2. Профілювання

Функція профілювання збирає інформацію профілю пристрою, як-от використання ЦП, рівень заряду акумулятора, інформацію про мережу тощо, яка постійно змінюється. У більшості випадків інформація профілю є неточною, оскільки береться середнє значення певного інтервалу.

3. Прийняття рішень

Для будь-якої системи розвантаження прийняття рішень є найважливішою та найскладнішою частиною. Операція прийняття рішень залежить від кроку профілювання, згаданого раніше. Для фактичного виконання процес прийняття рішень використовує інформацію профілювання та надає результат на основі цього.

Далі в цьому документі пояснюється, як ці відкриті проблеми вирішуються різними методами розвантаження. Далі йде порівняння між Grid Computing і Cloud Computing. Крім того, досліджуються аспекти мережі, оскільки це важливий фактор, що впливає на продуктивність. Таким чином, вся ця інформація збирається, і надаються різні рішення з існуючих досліджень щодо енергозбереження.

Парса та ін. [9] пропонує підхід до обгортання існуючих програм у рівні веб-сервісів, щоб до компонента можна було отримати доступ через веб-сервіси. При цьому підході спочатку аналізується код, а потім він обертається у веб-сервіс. Фаза аналізу визначає спосіб використання функцій аналізованої програми у веб-середовищі.

Це виконується в чотири етапи:

- оцінка,
- перетворення,
- реінжиніринг,
- створення веб-сервісів.

Після виконання цих кроків отримується новий код, який містить цінні функції програми. На наступному кроці важливі функції загорнуті у веб-сервіси за допомогою класу сервісної шини. Клас сервісної шини виступає як проміжна ланка між існуючою системою та веб-сервісом.

Його основний обов'язок — перенести отримані функції у зовнішнє середовище. Ці два кроки виконуються, щоб створити інструмент, який можна встановити в середовищі програмування, щоб запустити програму в експлуатацію.

1.8 Аналіз існуючих технічних робіт по предметній області

У таблиці 1.1 наведено інформацію про роботу, виконану дотепер дослідниками в області, тісно пов’язаної з цією дослідницькою роботою; також згадуються рік і внески.

Таблиця 1.1 – Існуючі наукові дослідження в предметній області

Науково-дослідна робота	Внески	Обмеження
Рамка для розвантаження контекстно-залежних обчислень у мобільних хмарних обчисленнях	- Пропонує шаблон проектування для обчислювального розвантаження	- Фреймворк не включає ресурсомісткі програми. Він розглядає лише інтенсивні обчислення програми.
	- Представляє модель оцінки для розрахунку скороченого часу виконання та затримки мережі, а потім вибирає хмарний ресурс для розвантаження	- Розглянутими параметрами є лише час виконання та енергоспоживання. Грошова вартість, споживання пам’яті та економічна вартість не враховуються.
Бхаттачарья А. та Де П. (2017).	- Рішення класифікуються для адаптивного розвантаження на основі параметрів, до яких може адаптуватися система	
Енергоефективна сервісно-орієнтована архітектура для передачі мобільної хмари від Хані К. Б. і Діхтера		- У цій роботі розглядається параметр безпеки, проте експерименти не проводяться, тому рішення для можливих зловмисних атак не надаються.
J. P. (2017).	- Впроваджує безпечну енергоефективну архітектуру якості обслуговування (EEQoSA) для процесу передачі обслуговування в мобільних хмарних обчисленнях	- Такі параметри, як пам'ять і вартість, не враховуються, що свідчить про те, що обмеження ресурсів мобільних пристроїв не враховуються.
Розвантаження обчислень для робочого процесу служби в мобільних хмарних обчисленнях	- У цьому документі пропонується мобільна і відмовостійка система розвантаження для виконання обчислень стратегії розвантаження - Запропонований алгоритм розвантаження базується на генетичному алгоритмі - Цей папір розглядається в основному завдяки його технології відмовостійкості, у разі втрати зв'язку	- У цій статті передбачається рух користувача. Але оскільки є й інші можливості, він не працюватиме точно в усіх сценаріях. - Також передбачається, що потужність сигналу є постійною на деякій частині шляху.

Ден С., Хуан	- У цій статті пропонується архітектура, заснована на послугах, і розроблена структура для розвантаження обчислювальних програм для транспортних засобів.	- У цій статті розробник статично позначив модулі, які потрібно вивантажити. Отже, розвантаження не залежить від контекстної інформації.
Л., Тагері Дж. і	- Це прототип наскрізної системи розвантаження, яка використовує запропоновану структуру, керовану послугами.	- Це оглядовий документ, який лише надає інформацію про існуючі алгоритми розвантаження, що обумовлюють енергію.

2 АЛГОРИТМІЧНІ ТА МАТЕМАТИЧНІ РІШЕННЯ ДЛЯ ЗАДАЧ БАЛАНСУВАННЯ МОБІЛЬНИХ ОБЧИСЛЕНЬ

2.1 Загальна структура системи

У цій главі спочатку обговорюється загальна структура системи розвантаження, а потім детальне пояснення структури, що використовується в нашій обчислювальній системі розвантаження. Загальна структура системи розвантаження складається з різних модулів, таких як Client Proxy, Profiler і Solver. Ці модулі відповідають за виконання методу на стороні сервера. Крім того, наша обчислювальна система розвантаження заснована на сервіс-орієнтованій архітектурі, отже, методи перетворюються на послуги, а потім виконується розвантаження. Наша система розвантаження складається з таких модулів, як Context Monitor, Decision Making Engine, Communication Manager і Offloading Planner.

У наступному розділі обговорюється структура загальної системи розвантаження, а потім загальна блок-схема системи розвантаження.

У цьому методі спочатку враховуються дві вимоги:

1. Зібраний код знаходиться як на стороні мобільного пристрою, так і на стороні сервера

2. Клієнтський проксі-сервер, профайлер і вирішувач встановлюються на мобільній і серверній стороні.

Профайлер збирає інформацію про характеристики мережі з самого початку і відстежує цю інформацію після цього. Щоразу, коли викликається метод, профайлер вимірює його потенціал енергозбереження та профілює пристрій і мережу, щоб отримати інформацію про стан. Потім вирішувач працює з результатами, наданими профайлером, і визначає, де метод буде виконуватися віддалено. Проксі відповідає за контроль серверного пристрою та передачу даних.

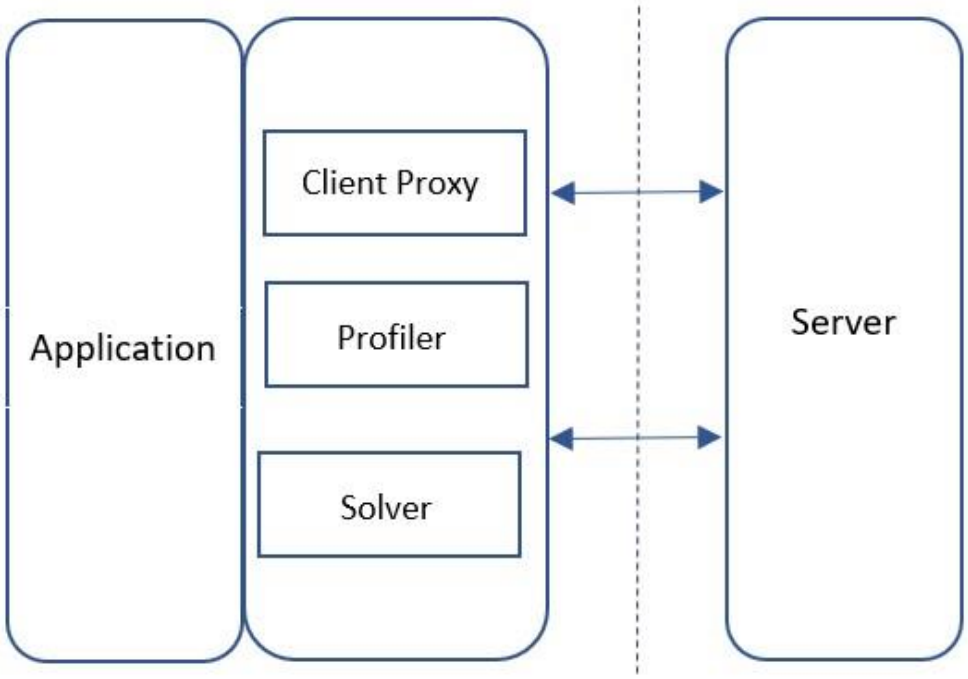


Рисунок 2.1 - Загальна структура системи для розвантаження

На рисунку 2.1 показано загальну структуру системи для розвантаження, в якій виконується міграція коду.

2.2 Пропонована обчислювальна система розвантаження

Оскільки запропонована система розвантаження обчислень заснована на сервіс-орієнтованій архітектурі, ми спочатку обговоримо роботу та важливість SOA.

У запропонованій системі компоненти перетворюються на сервіси, а взаємодія сервісів здійснюється за допомогою сервісно-орієнтованої архітектури. SOA — це стиль розробки програмного забезпечення, в якому послуги надаються компонентами програми іншим компонентам через мережевий протокол зв'язку [23]. Використовуючи SOA, компоненти можна описати більш чітко, що дає змогу структурувати служби. Отже, в системі деякі компоненти є постачальниками послуг, які надають необхідну послугу, а деякі

є споживачами послуг, які запитують і отримують послугу. Архітектура SOA має проміжне програмне забезпечення, яке є посередником між постачальником і споживачем.

Проміжне програмне забезпечення контролює зв'язок між постачальником і споживачем. SOA є особливо корисним у пропонованій системі через його слабко пов’язану природу, тобто інтерфейс сервісу не залежить від реалізації. Різні розробники можуть створювати різні додатки, створюючи один або кілька сервісів, не знаючи про основні реалізації веб служб.

Наприклад, служба може бути реалізована на python або .Net, а програма, яка використовує службу, може бути на іншій платформі або на іншій мові. Це також допомагає системам адаптуватися, залишаючи споживачів послуг окремо від змін, які відбуваються під час впровадження послуги [4]. Нарешті, використовуючи наявну програмну інфраструктуру для створення нових сервісів, SOA забезпечує кращу гнучкість у швидкому створенні програм і процесів.

Запропонована нами система включає мобільний пристрій, який складається з додатків, і хмарний сервер, який надає функціональні можливості додатків як сервісів. Зв'язок між постачальником послуг і споживачем послуг здійснюється за допомогою проміжного програмного забезпечення. Проміжне програмне забезпечення є основною частиною нашої системи розвантаження. Рисунок 2.2 показує запропоновану структуру системи розвантаження.

У цій системі проміжне програмне забезпечення складається з чотирьох різних модулів:

1. Монітор контексту
2. Машина прийняття рішень
2. Менеджер з комунікацій
4. Планувальник розвантаження

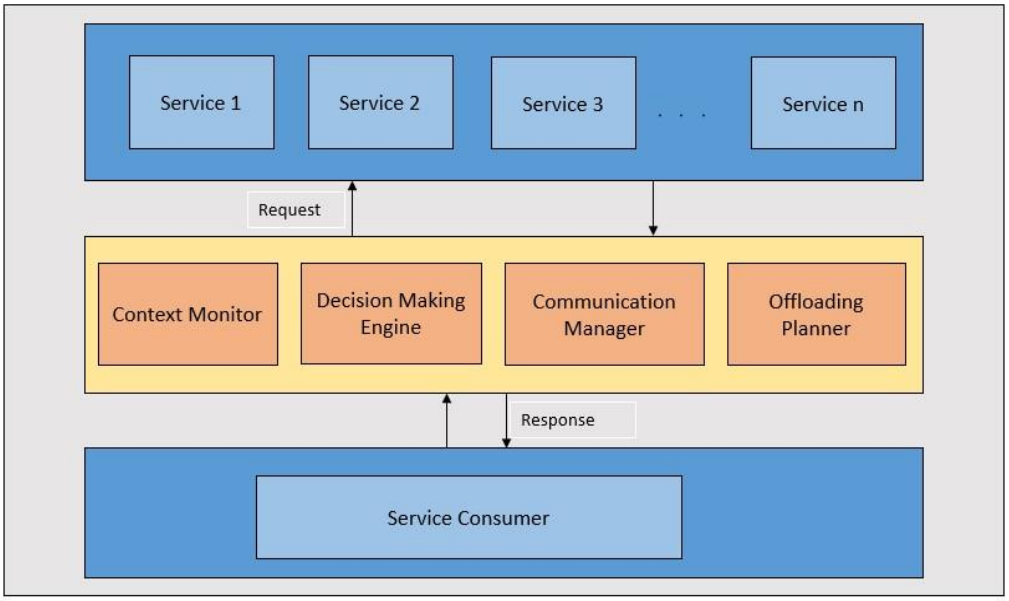


Рисунок 2.2 - Пропонована структура системи для розвантаження

У наступних розділах надається детальна інформація про модулі проміжного програмного забезпечення для системи розвантаження:

1. Монітор контексту

Монітор контексту відповідає за збір інформації про контекст і надання цих значень механізму прийняття рішень під час виконання служби. Ця контекстна інформація включає:

а. Профіль мобільного пристрою: обчислювальна потужність, середнє використання ЦП, обсяг доступної пам'яті;

б. Модель мобільності під час роботи: розташування автомобіля;

с. Стан мережі: наявність стільникової мережі (4G, 3G) та її потужність сигналу, наявність бездротової мережі (WiFi) та потужність сигналу

Для будь-якого мобільного пристрою, на якому виконується служба, Context Monitor збирає цю інформацію (згадану вище), а також час від часу оновлює інформацію. Потім ця інформація надається до механізму прийняття рішень. На рисунку 2.3 ми відобразили блок-схему для Context Monitor.

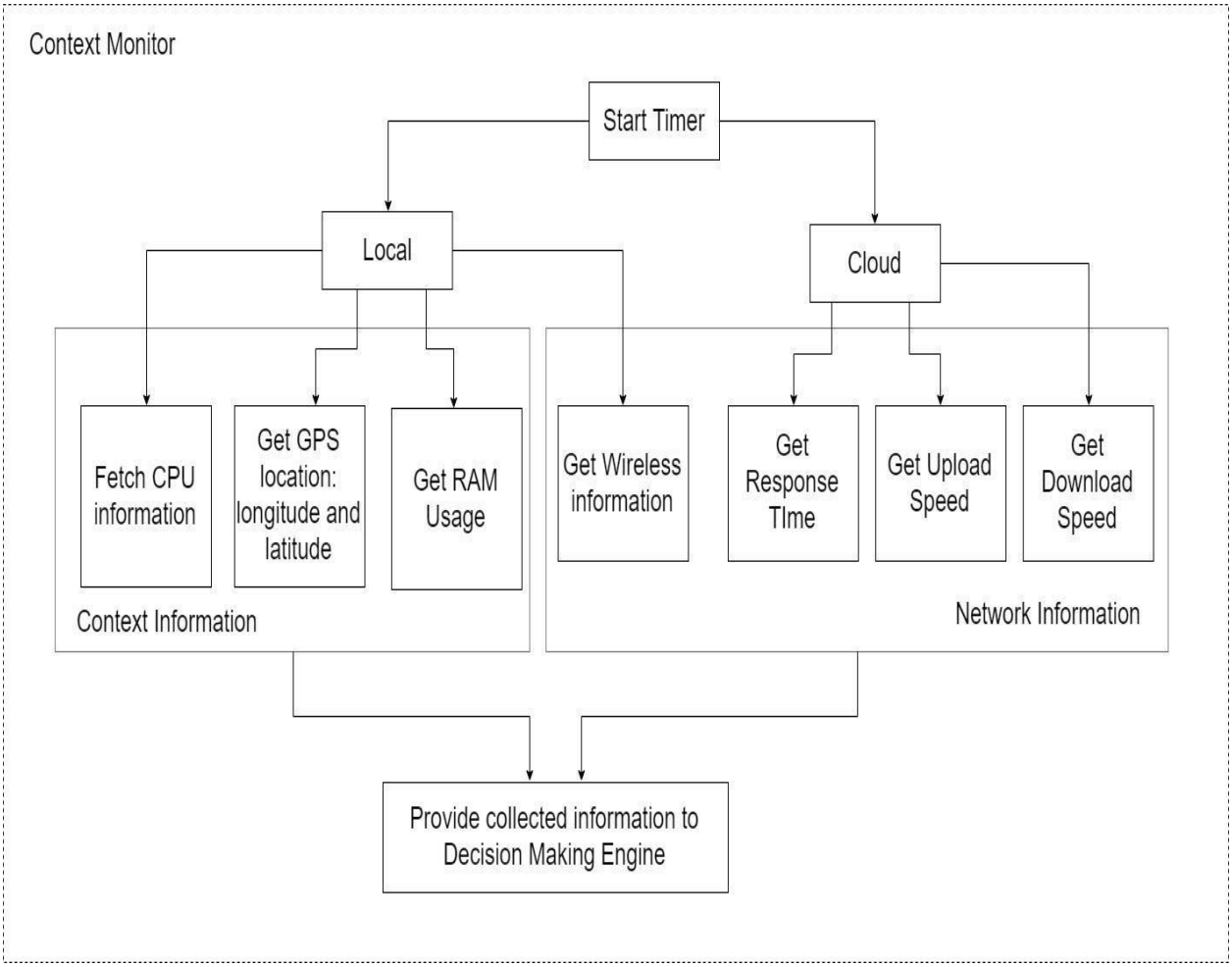


Рисунок 2.3 - Монітор контексту

Машина прийняття рішень

Механізм прийняття рішень спочатку аналізує контекстну інформацію, таку як доступні мережеві з'єднання, обчислювальна потужність, стан автомобіля, доступний розмір пам'яті, швидкість завантаження та швидкість завантаження. Потім він використовує результат аналізу інформації про контекст для прийняття рішень. Для прийняття рішення про місце виконання послуги. Використовується цільова функція. Цільова функція визначається наступним чином:

$$OF = P(s1,s2,\ldots,sn)*WP + M(s1,s2,\ldots,sn)*WM + E(s1,s2,\ldots,sn)*WE,$$

де s – послуги, P – продуктивність (обчислення), M являє собою пам'ять, E являє собою економічну вартість, WP представляє вагове значення продуктивності, WM представляє вагове значення пам'яті і WE представляє вагове значення економічної вартості.

Цільова функція розраховується як зважена сума продуктивності, пам’яті та економічної вартості. Значення ваги встановлюються відповідно до контексту мобільного пристрою. Наприклад, якщо пам’ять пристрою менша за порогове значення, ми збільшуємо значення ваги в цільовій функції. Отже, значення ваги встановлюються відповідно до переваг будь-якого параметра в цей момент. Значення ваги коригуються таким чином для обробки різних одиниць параметрів. Сума значень ваги завжди буде 1.

Отже, для кожного сервісу розраховується p.local і p.cloud, де p = p.local = W/C, де W — робоче навантаження служби, а C — потужність ЦП,

p.cloud = TR +RT +TO, де RT – час відгуку хмарного сервісу;

TR – час завантаження вхідних даних = розмір вхідних даних/швидкість передачі даних;

TO – час завантаження вихідних даних = розмір вихідних даних/швидкість передачі даних;

Далі йде розрахунок пам’яті та економічної вартості.

Економічна вартість послуги si визначається як: E(si) = E.Cloud(si) * IsOffloading,

де IsOffloading — це значення, яке дорівнює 0 або 1. 0 означає, що si виконується локальною службою, а 1 — що виконується хмарною службою. E.Cloud(si) = DI + DO, де DI — розмір вхідних даних виклику служби;

DO — це розмір вихідних даних виклику служби.

Потім Механізм прийняття рішень приймає рішення про зміну (якщо потрібно) значення ваги обчислювальної потужності, пам’яті та грошових витрат у функції цілі відповідно до різної контекстної інформації. Наприклад, якщо обсяг пам’яті в локальній системі нижче порогового значення, значення

ваги пам’яті (WM), яке вказує на пріоритет цього параметра, буде збільшено. Нарешті, якщо результат аналізу поточної інформації про контекст свідчить про те, що виконання на локальному рівні краще, Механізм прийняття рішень приймає рішення, що служба повинна працювати локально, інакше вона приймає рішення розвантажити службу в хмарі. Отже, рішення приймається на основі значення цільової функції. На рисунку 2.4 показана блок-схема механізму прийняття рішень.

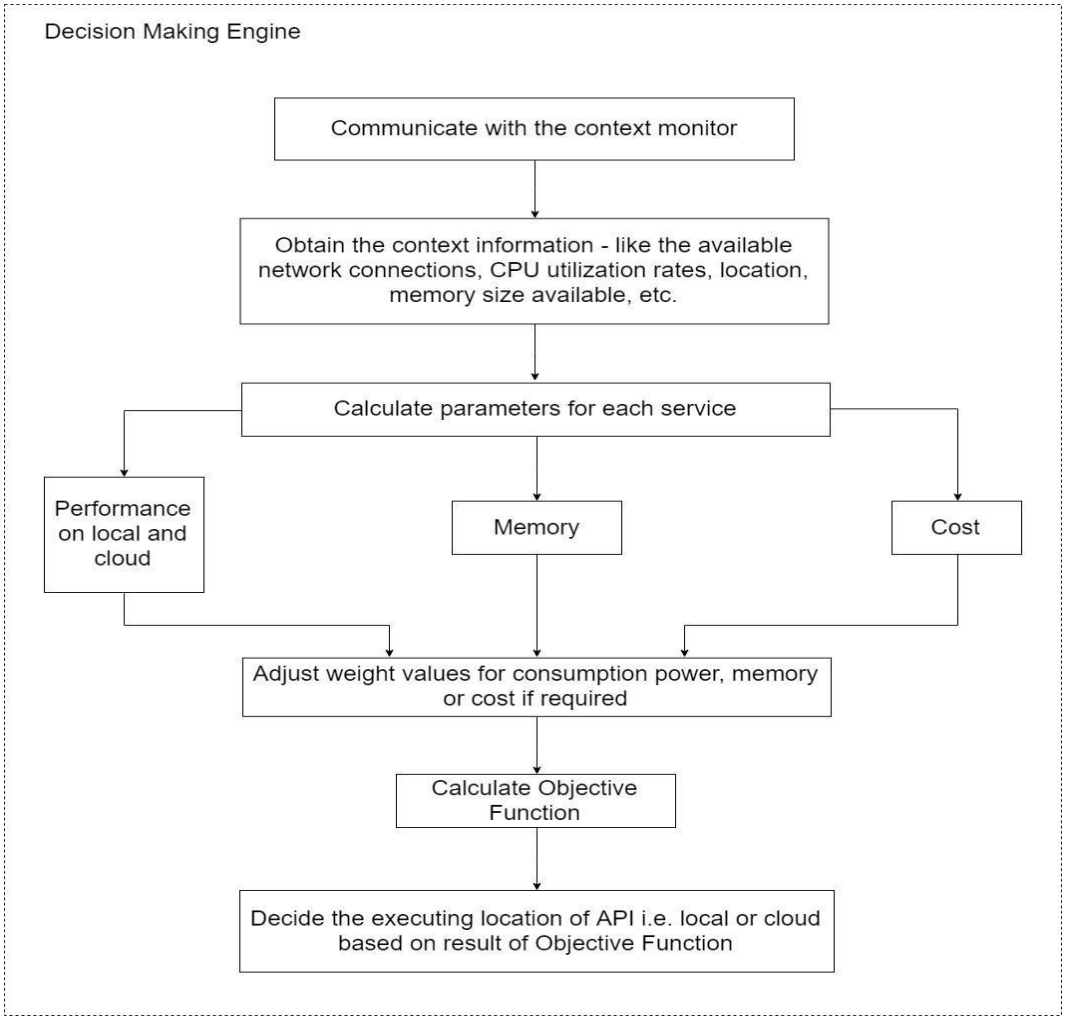


Рисунок 2.4 - Система прийняття рішень

Менеджер з комунікацій

Менеджер з комунікацій відповідає за передачу інформації між локальним і хмарним. Він інкапсулює нудні деталі серіалізації даних і

десеріалізації даних з одного формату в інший. Він також синхронізує план розвантаження між мобільним пристроєм і хмарою.

Планувальник розвантаження

Планувальник розвантаження відповідає за розвантаження сервісу в хмару. Планувальник розвантаження також вибирає одну послугу в ситуації, коли є більше одного кандидата. Слід зазначити, що, оскільки рішення про те, де працює служба, залежить від контекстної інформації в певний час, план розвантаження для певної моделі робочого процесу також час від часу буде змінюватися через зміну контекстної інформації. Отже, планувальник розвантаження виконує план на основі рішення та інформації про контекст. На рисунку 2.5 показано хід роботи Communication Manager і Offloading Planner

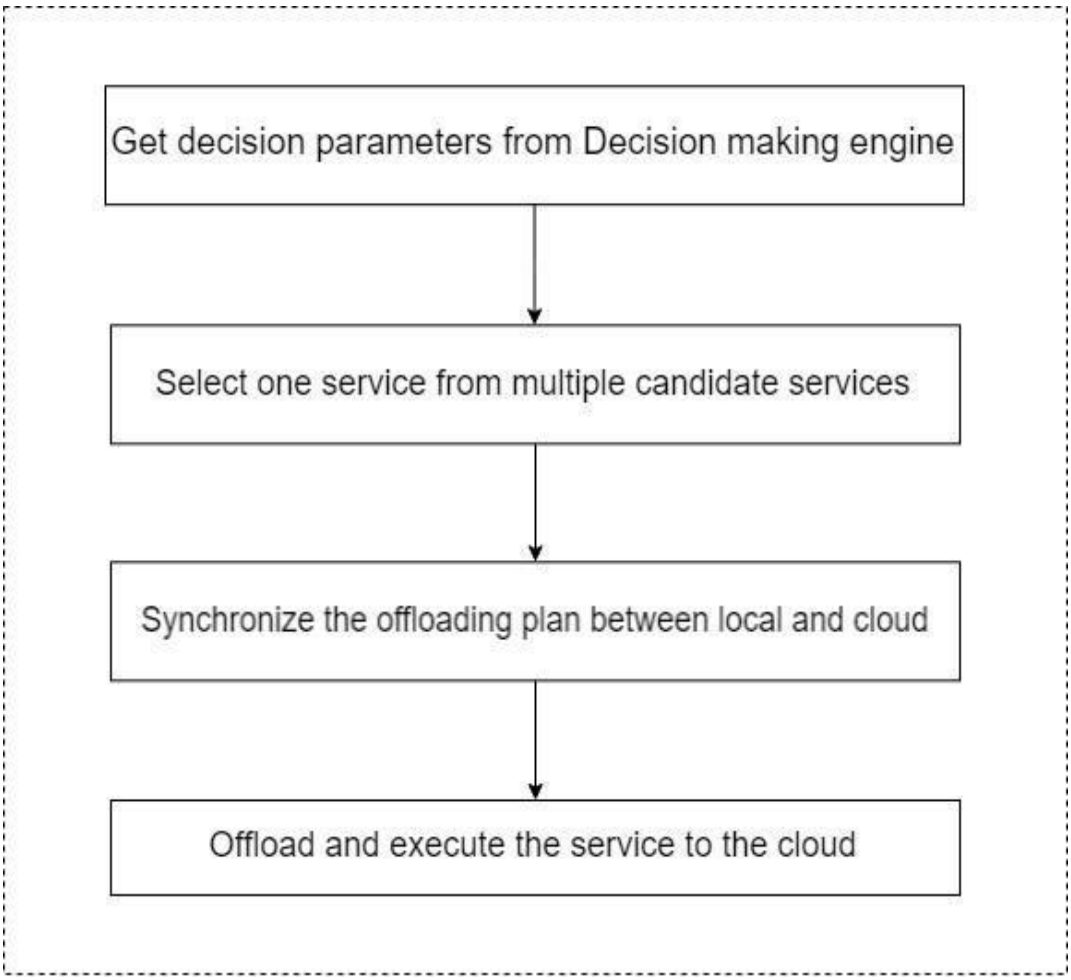


Рисунок 2.5 - Менеджер зв’язку та планувальник розвантаження

2.3 Представлення основного алгоритму системи

На рисунку 2.6 показана загальна блок-схема нашої системи розвантаження.

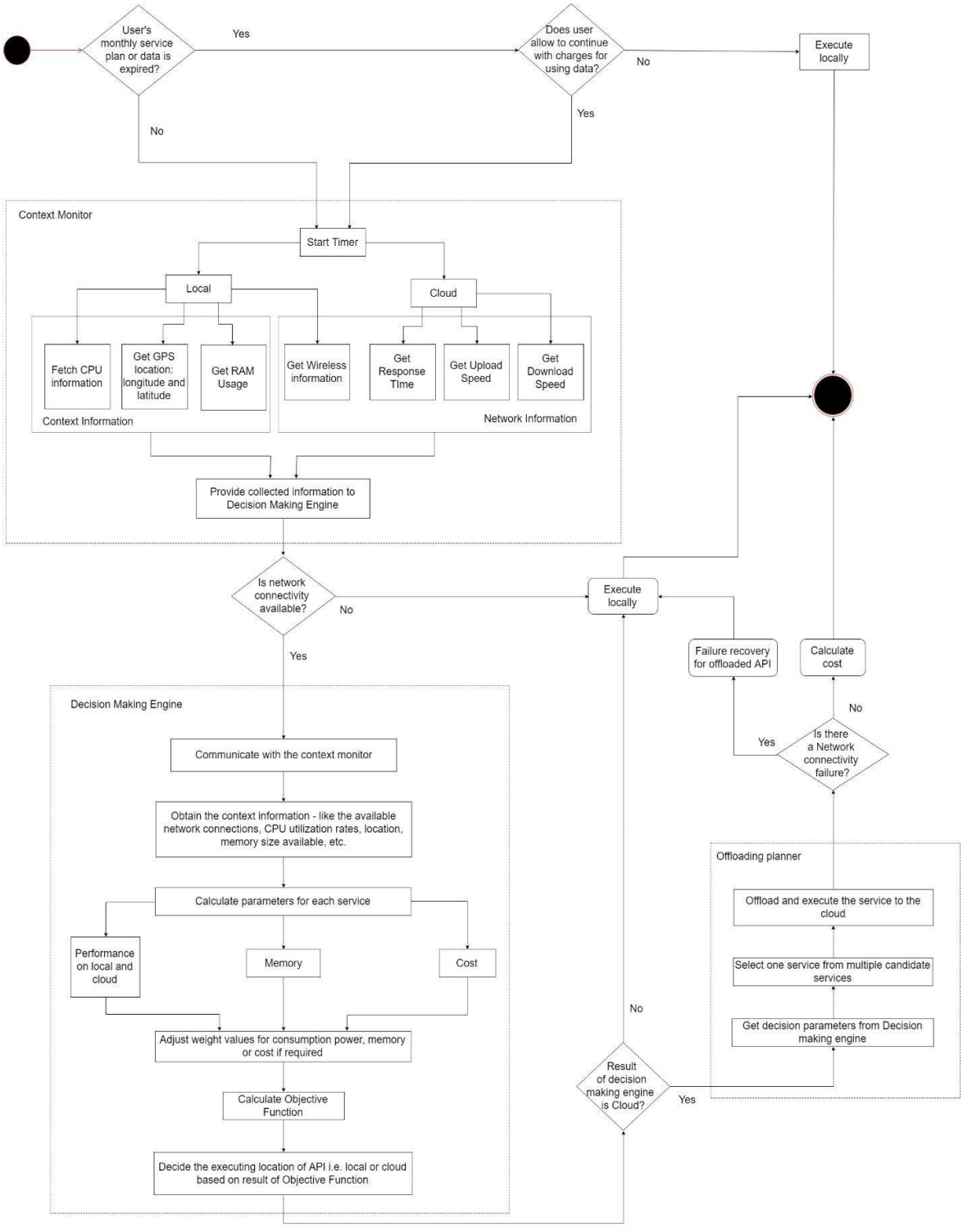


Рисунок 2.6 - Загальна блок-схема системи розвантаження

Як показано на рисунку, спочатку перевіряється термін дії тарифного плану користувача. Якщо термін його дії закінчився, у користувача буде запитано, чи хоче користувач дозволити додаткові збори зі свого облікового запису чи ні. Якщо термін дії плану даних не закінчився, процес переходить до наступного кроку, інакше служба виконується локально.

Потім Context Monitor збирає всю необхідну інформацію, як-от мережеві з'єднання, обчислювальна потужність, довгота та широта розташування пристрою, розмір доступної пам'яті, швидкість завантаження, швидкість завантаження тощо. З Context Monitor інформація про те, чи доступне мережеве з'єднання чи ні, виходить. Якщо вона недоступна, то служба буде виконуватися локально, але якщо вона доступна, інформація передається до механізму прийняття рішень. Механізм прийняття рішень аналізує інформацію та отримує значення різних параметрів, таких як продуктивність, пам'ять та економічна вартість, як обговорювалося в попередньому розділі. Потім у функції цілі він налаштовує значення ваги будь-якого параметра, якщо потрібно. Далі йде розрахунок цільової функції. На основі результату функції цілі приймається рішення про місце розташування (локальне чи хмарне). Якщо рішення є локальним, то послуга виконується локально, інакше інформація передається до Communication Manager і Offloading Planner. Communication Manager синхронізує план розвантаження між мобільним пристроєм і хмарою. Планувальник розвантаження керує планом розвантаження та виконує службу в хмарі. Потім розраховується вартість у випадку, якщо термін дії тарифного плану закінчився і з користувача потрібно буде стягувати додаткову плату за використання даних.

3 РЕАЛІЗАЦІЯ МОРДЕЛЕЙ ТА ФРЕЙМВОРКУ ДЛЯ ЗДІЙСНЕННЯ БАЛАНСУВАННЯ ПРОЦЕСІВ В МОБІЛЬНИХ ОБЧИСЛЕННЯХ

3.1 Інформація про програмне забезпечення

Запропонований підхід реалізовано в ОС Windows з використанням мови програмування C# та .NET Framework. Для експериментів використовується підхід на автономних автомобільних модулях. Перелік використовуваного програмного забезпечення та інструментів наведено в таблиці 3.1.

Таблиця 3.1 - Список інструментів, що використовуються для впровадження та експериментів

ПУНКТ	ДЕТАЛІ
Операційна система	Windows
Мови	C#
IDE	Visual Studio 2017
Framework додатків	ASP.NET

Реалізація запропонованої методології була виконана на ноутбуці Dell з Intel(R) Wireless-AC 9560. Хмарний сервер - Amazon Web Services.

3.2 Набори даних, що використовуються при реалізації

Для різних автономних автомобільних модулів для експериментів використовуються різні набори даних. Використовується набір даних ShapeNet [25], який є репозиторієм з інформацією про ключові точки для різних візуалізованих зображень 3D-моделей автомобілів. Модель DensePose [6], яка має власний набір даних істини про землю, зібраний вручну, під назвою COCO-DensePose, прийнята та інтегрована в нашу систему для тривимірної оцінки

пози пішоходів на дорозі. Для перевірки об’єктів та видалення об’єктів 3D віртуальний світ створено з використанням даних OpenStreetMap (VGI/краудсорсинг) і текстури фасаду із зображень перегляду вулиць Google (2D-види вулиць і супутникові зображення). Це тривимірне середовище містить стаціонарні (наприклад, будівлі) і змінні (наприклад, дерева) об’єкти. На рисунках 3.1 і 3.2 нижче показано приклад створеного 3D віртуального світу.



Рисунок 3.1 - Сконструйований 3D віртуальний світ

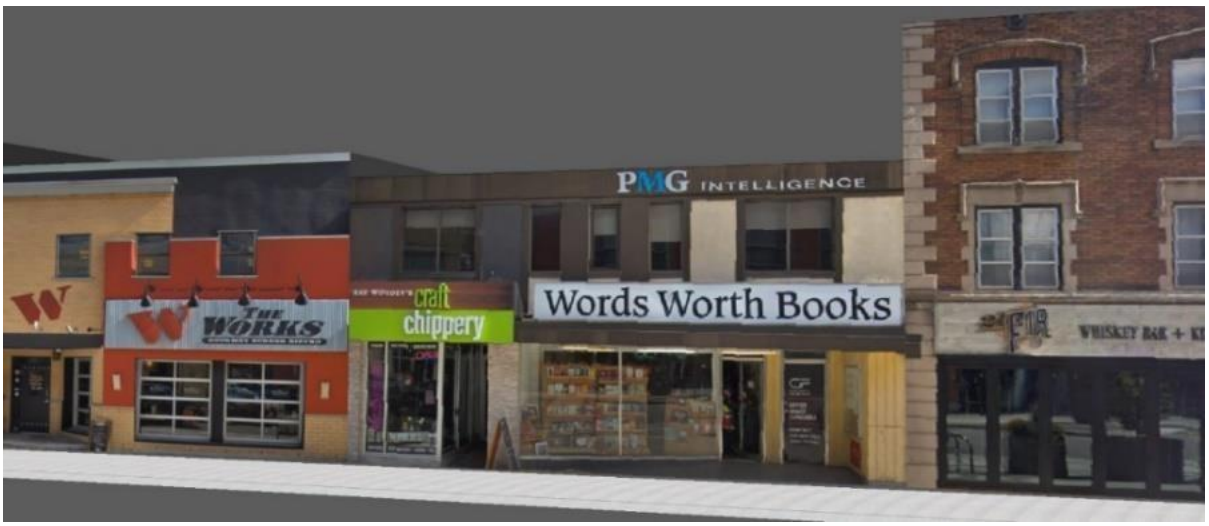


Рисунок 3.2 - Сконструйовані 3D-об'єкти

3.3 Розробка системи для самокерованих транспортних засобів

Як згадувалося, експерименти для нашої системи розвантаження проводяться на автономних модулях автомобіля. В основному для цієї системи існує шість різних модулів, які взаємопов'язані один з одним. Ці шість модулів:

1. Побудова на віртуальному 3D середовищі
2. Відтворені зображення відео в реальному часі
3. 3D-функція та виділення ключових точок
4. Видалення статичних і змінних об'єктів
5. Динамічне розпізнавання об'єктів
6. Динамічне виявлення об'єктів

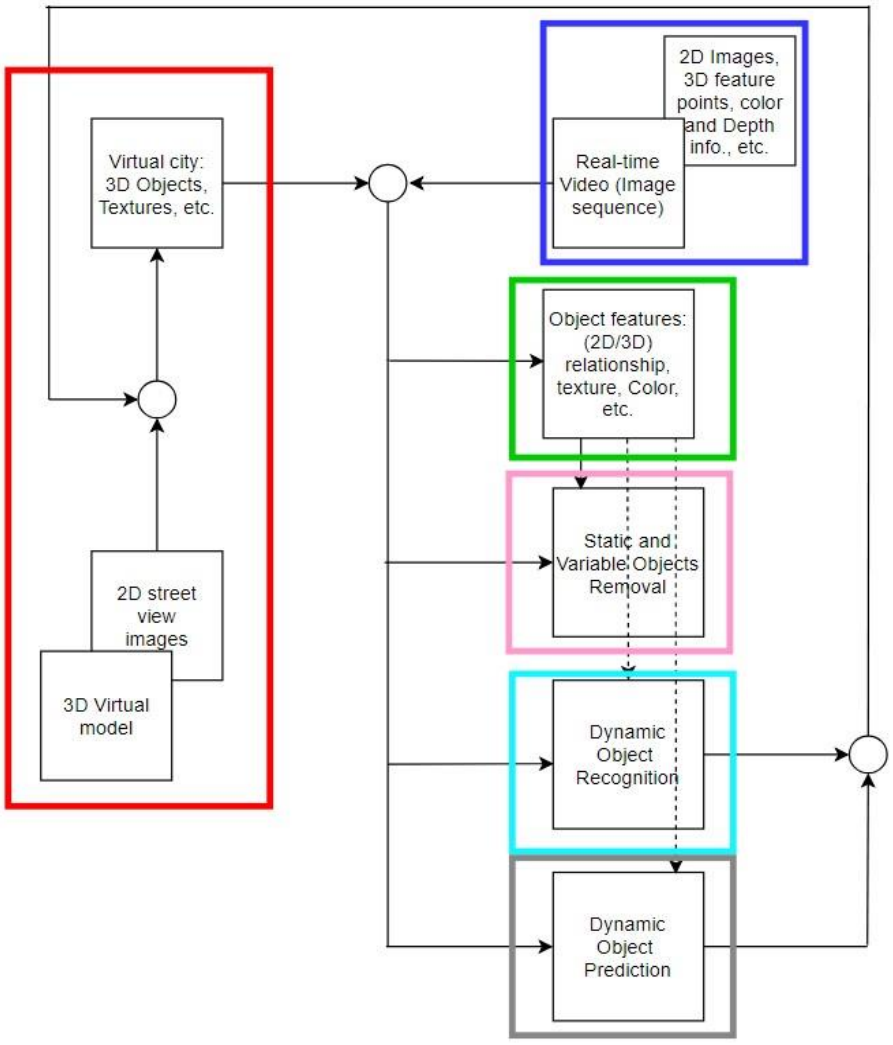


Рисунок 3.3 - Загальна система

Нижче наведено опис загальної системи з посиланням на рисунку 3.3: загальна система спочатку займається побудовою віртуального 3D-середовища з використанням даних OpenStreetMap (VGI/краудсорсинг) і текстури фасаду із зображень перегляду вулиць Google. Віртуальна тривимірна модель міста складається зі статичних об’єктів, наприклад, будівель, і деяких змінних об’єктів, наприклад, дерев. Крім цього, є окремий репозиторій, який містить 3D-моделі динамічних об’єктів, наприклад автомобілів. Модуль, позначений синім кольором на рисунку 3.3, показує відео в реальному часі (послідовності зображень), що передається як вхід до системи. Віртуальне середовище відтворюється, а функції ключових точок і тривимірні функції зберігаються в репозиторії; ця робота виконується в модулі, пофарбованому в рожевий колір. Модуль, позначений жовтим, є модулем усунення статичних і змінних об’єктів. У цьому модулі узгоджуються характеристики ключових точок вхідного зображення (синій модуль) і ключових точок віртуального середовища (рожевий модуль). Збіг ключових особливостей віртуального середовища та зображення в реальному часі підтверджує розташування автомобіля в реальному світі; це вирішує проблему геолокалізації самокерованого автомобіля. За допомогою відповідності також підтверджується розташування статичних об'єктів, і вони виключаються з процесу ідентифікації об'єктів; що забезпечує більше часу для ідентифікації та прогнозування динамічних об’єктів, таких як люди чи тварини на дорозі, оскільки саме вони впливають на навігацію самокерованої системи. Модуль, позначений блакитним кольором, займається розпізнаванням об’єктів та оцінкою пози динамічних об’єктів, присутніх у вхідному зображенні в реальному часі, наприклад автомобілів. Крім того, цей модуль відстежує розпізнані об’єкти з кількох кадрів відео та обчислює швидкість динамічного об’єкта. Розпізнаний об’єкт з інформацією про позицію разом зі швидкістю та розташуванням об’єкта використовується для оновлення динамічних об’єктів у 3D віртуальному середовищі. Модуль,

позначений сірим кольором, оновлює інформацію про динамічні об’єкти у віртуальне середовище.

Розвантаження — це процес перенесення обчислювальних або ресурсомістких завдань з мобільного пристрою на віддалений сервер або хмарні ресурси, як обговорювалося раніше. Оскільки цей підхід використовує архітектуру, орієнтовану на обслуговування, модулі, які обговорювались у попередньому розділі, обгортаються як служби, і ці служби вивантажуються в хмару. Перед розвантаженням служби на модулях потрібен процес розділення коду. По-перше, код розділяється, просто позначаючи деякі частини коду як невивантажувані. Це ті частини, які взаємодіють з будь-яким пристроєм введення або виведення, або це може бути частина, яка реалізує інтерфейс програми (користувальницький інтерфейс). Потім для окремих модулів виконується аналіз вихідного коду. Під час аналізу вихідного коду, якщо є деякі підмодулі, які є спільними для різних модулів, то вони ідентифікуються, і для них створюється лише один сервіс. Цю службу потім можна буде використовувати між цими модулями. Остаточне розвантаження здійснюється на основі механізму прийняття рішень, як описано в розділі 3.

Впровадження системи SOA

Після виконання початкового розбиття коду для отримання підмодулів використовуються такі модулі, як 3D-функція та виділення ключових точок, Видалення статичних і змінних об’єктів, а також Динамічне розпізнавання об’єктів та Оцінка пози. Підмодулі, такі як Пошук списку ключових точок, відповідних ключових точок, виявлення об’єктів, оцінка пози тощо, створюються, а потім обгортаються як служби.

Нижче наведено служби, створені для різних модулів і потоку:

1. Створюється сервіс для отримання 3D-об'єкта, який має раніше створене віртуальне 3D-середовище.

2. Потім створюється служба (FasterRCNN) для виявлення об'єктів для статичних і змінних об'єктів, таких як будівлі, лавки та дерева.

3. Потім для різних будівель і моделей автомобілів створюється сервіс вилучення точок ознак (CrtFeatExtraction).

4. Після виявлення об'єктів йде служба перевірки об'єктів (ObjVerification), знову ж таки для статичних і змінних об'єктів.

5. Потім створюється сервіс (HeatmapGeneration) для створення теплових карт для будівель і дерев.

6. Потім створюється служба видалення об'єктів (ObjElimination), яка включає інформацію про теплову карту.

7. Для розпізнавання об'єктів та оцінки пози динамічних об'єктів, таких як автомобілі та пішоходи, створені служби пошуку ключових точок, відповідних ключових точок, виявлення об'єктів та оцінки пози.

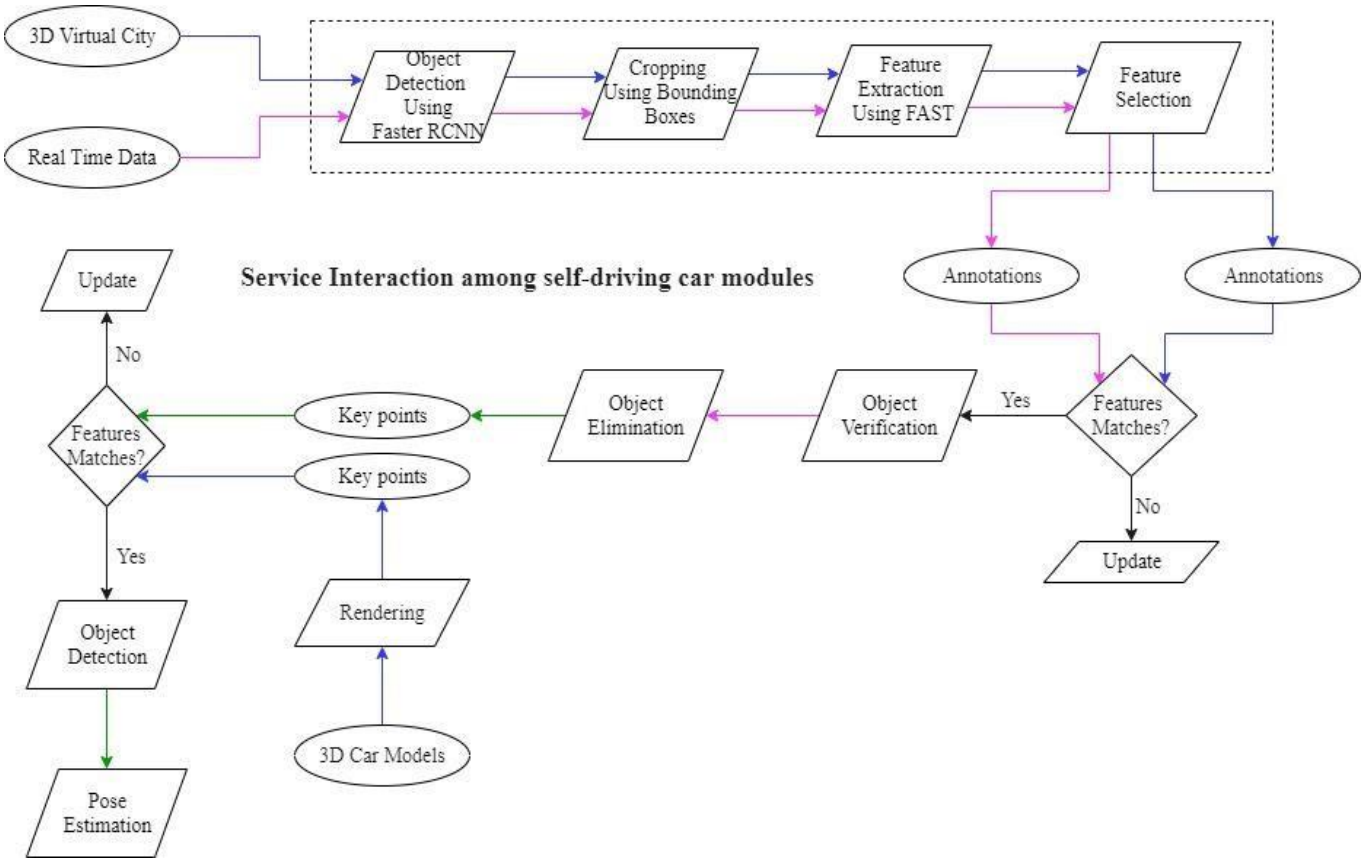


Рисунок 3.4: Сервісна взаємодія між модулями самокерованого автомобіля

Взаємодія цих модулів як сервісів показано на рисунку 3.4. Паралелограм представляє наші ідентифіковані послуги, послуги в пунктирному полі — це загальні послуги, які спільно використовуються між модулями, еліпс позначає вхідні або вихідні дані, а різні кольорові лінії демонструють потік окремих модулів.

Для обгортання коду як служби створюються Restful API. Це означає, що API призначений для отримання, створення, оновлення та видалення об’єктів за допомогою дієслів Http GET, POST, PUT, PATCH і DELETE.

Http використовується для зв’язку, а ядро ASP.NET має конвеєр проміжного програмного забезпечення, який викликається для кожного запиту. Для гнучкості всі шляхи додаються до файлу веб-конфігурації проекту, щоб будь-який файл можна було додати за цим шляхом і запустити як службу. У випадку, якщо служба надає текстовий результат, напр. список ключових пунктів, які відображаються у форматі XML відповідно до стандарту SOA. У випадку, якщо це зображення надається в результаті, воно зберігається/завантажується, а шлях отриманого зображення зберігається у файлі конфігурації веб, щоб служба, що запитує, могла отримати доступ до зображення з цього шляху, як показано на рисунку 3.5. Відповідно до галузевих стандартів (з огляду на Uber і Tesla) використовується концепція Web API. Важливі сховища даних (тобто зображення моделей автомобілів, людей тощо) з модулів інших студентів зберігаються на локальному рівні. Менш важливі зображення, такі як дикі тварини, зберігаються в хмарі, щоб вони не набули пам'яті на локальному.

```
  <add key="CarResults" value="C:\\Users\\Rajasi\\WebA
  <add key="BuildingResults" value="C:\\Users\\Rajasi\
  <add key="Result_Image" value="D:\\result.jpg" />
</appSettings>
```

Рисунок 3.5 - Знімок файлу Web Config проекту

Як згадувалося в Розділі 3.3.1 і раніше в Розділі 3.3.2, деякі підмодулі в коді мають спільні функції. Для цього буде зайвим створити один і той же сервіс для різних модулів. Таким чином, створюється єдиний сервіс, який можна використовувати в різних модулях. Наприклад, швидший метод rcnn використовується для виявлення об’єктів зображень у реальному часі, а також для зображень із віртуального світу.

Таким чином, створюється служба з ім’ям fasterRcnn, яку можна спільно використовувати для обох підмодулів. Єдина відмінність – це вхідні зображення, але оскільки код є динамічним, ім’я файлу зображення вказується у веб-конфігурації, а потім служба викликається відповідно до вимог. Інший подібний приклад — визначення кутів коду будівель, у якому дотримується той самий метод.

На рисунку 3.6 показано вхідне зображення для виявлення кутів, а на рисунку 3.7 — вихідне зображення, яке має кутові точки, виявлені швидким алгоритмом. За цим алгоритмом вибирається 32 точки, близькі до кутів.



Рисунок 3.6 - Вхідне зображення для швидкого алгоритму

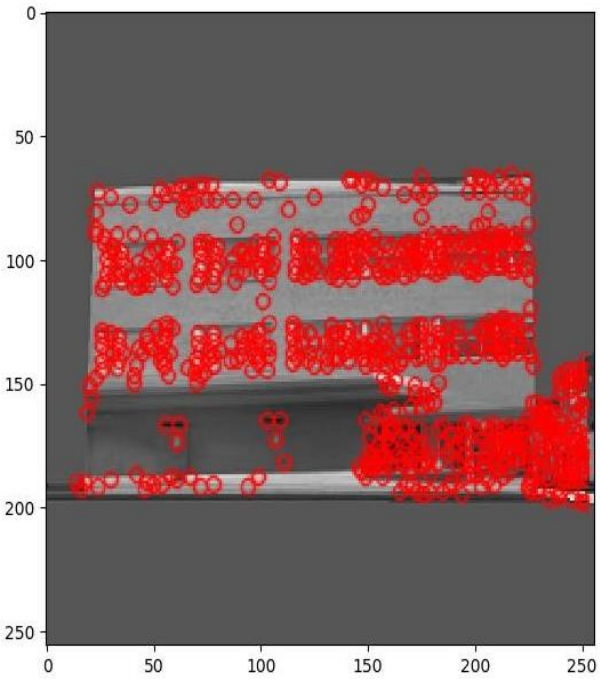


Рисунок 3.7 - Вихід швидкого алгоритму

3.4 Реалізація Context Monitor

Як згадувалося, контекстний монітор відповідає за збір контекстної інформації, яка включає профіль мобільного пристрою: обчислювальну потужність, середнє використання ЦП, розмір використовуваної пам'яті; Модель мобільності під час виконання: місцезнаходження автомобіля та стан мережі: наявність стільникової мережі (4G, 3G) та її потужність сигналу, наявність бездротової мережі (Wi-Fi) та потужність сигналу.

Для отримання цієї інформації було створено додаток-форму на C# і за допомогою ASP.NET Framework. Крім того, зібрана інформація зберігається у файлі, щоб час від часу її можна було відстежувати.

Нижче наведено класи або методи, які використовуються для отримання відповідної інформації:

Розташування: довгота та широта за допомогою класу GeoCoordinate система. Пристрій. РозташуванняБездротовий зв’язок: бездротова інформація за допомогою Клієнт Wlan бібліотеки Managed Wifi

Потужність обчислень. Отримайте обчислювальну потужність/використання процесора за допомогою інформації про процесор

Пам'ять: інформація про оперативну пам'ять за допомогою PerformanceCounter

Економічна вартість: розмір вхідних даних за допомогою request.ContentLength і розмір вихідних даних за допомогою response.GetResponseStream()

Вартість: швидкість передачі даних від оператора зв'язку на основі використання.

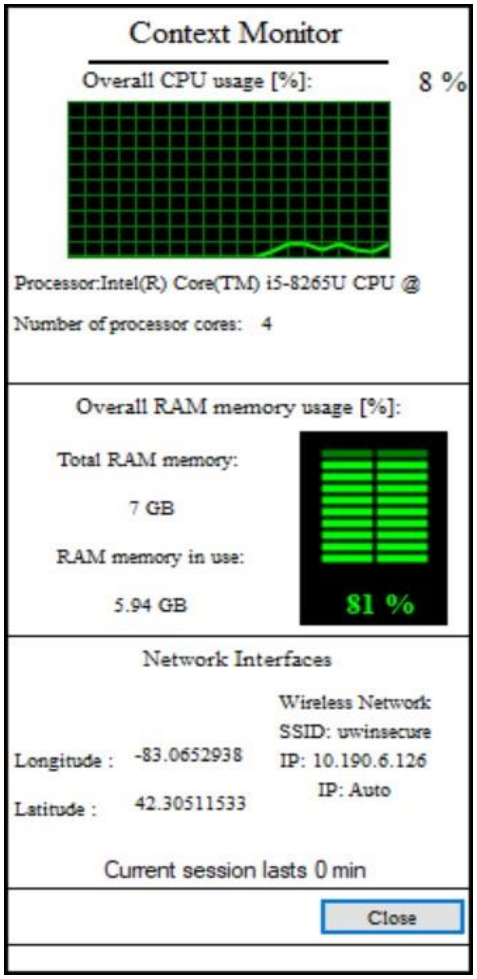


Рисунок 3.8 - Інтерфейс виводу моніторингу контексту

На рисунку 3.8 показана частина інтерфейсу користувача виводу Context Monitor, яка містить інформацію, згадану вище. На рисунку 3.9 показана інформація, збережена у файлі для цілей моніторингу.

```
RAM Percentage: 81 %
Total RAM: 7 GB
RAM Usage: 5.92 GB
Wireless IP: 10.190.6.126
Wireless SSID: uwinsecure
Wired IP: Not Detected
Wired Connected: Not Detected
Longitude: -83.0652938293894
Latitude: 42.3051153385679
End Data


Start Data 2019-10-29 10:38:31 AM
CPU Usage: 8 %
Battery Value: 42 %
RAM Percentage: 81 %
Total RAM: 7 GB
RAM Usage: 5.94 GB
Wireless IP: 10.190.6.126
Wireless SSID: uwinsecure
Wired IP: Not Detected
Wired Connected: Not Detected
Longitude: -83.0652938293894
Latitude: 42.3051153385679
End Data
```

Рисунок 3.9 - Виведення даних монітора контексту у файлі

3.5 Реалізація механізму прийняття рішень

У Механізмі прийняття рішень, як обговорювалося раніше, спочатку ми з’ясуємо, чи дозволяє користувач додаткову плату, якщо ліміт даних вичерпано, або Wi-Fi відключено. На рисунку 3.10 показано спливаюче вікно, яке відображається користувачам.

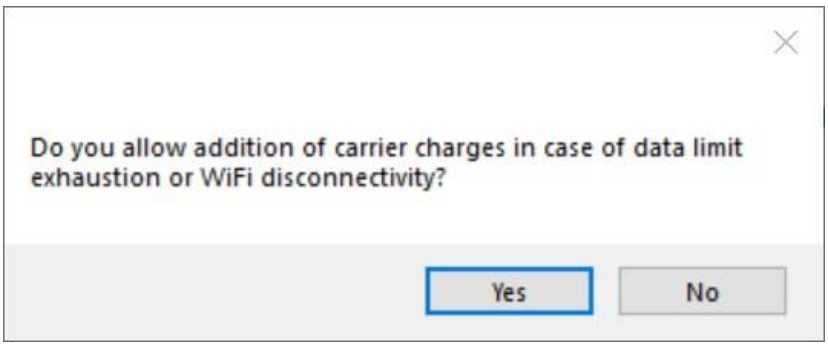


Рисунок 3.10 - Спливаюче вікно для отримання згоди користувача

Якщо користувач вибере «Ні», то послуга буде виконана локально, коли Wi-Fi буде відключено або ліміт даних вичерпано.

Ми розглянули наступні сценарії на основі роботи самокерованих автомобілів:

Сценарій 1: Сценарій за замовчуванням – це вважається нормальною ситуацією, враховуючи, що автомобіль щойно запустився. Таким чином, на даний момент використовується менше пам'яті, Wi-Fi також підключений у цей час, і виконується послуга щодо отримання вхідних зображень та отримання ключової точки на основі цих зображень.

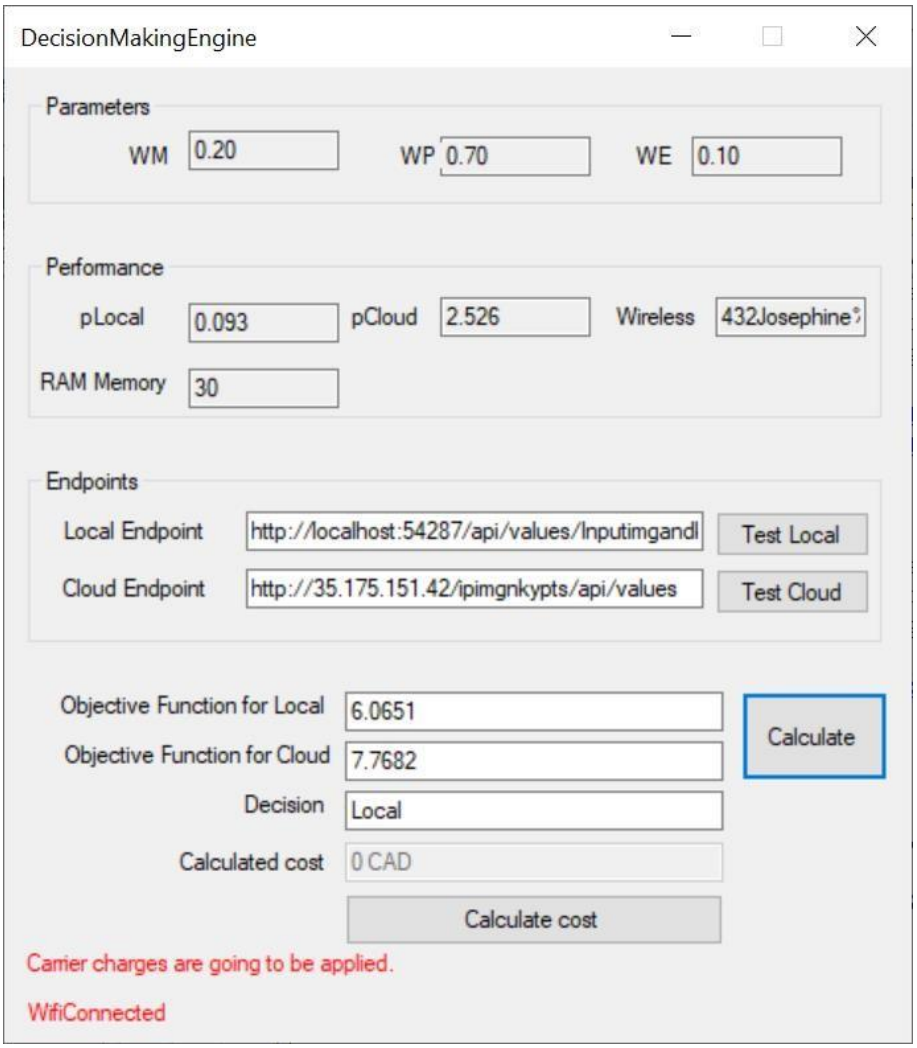


Рисунок 3.11 - Сценарій механізму прийняття рішень 1

Як відомо, є три параметри: Пам'ять, продуктивність і економічна вартість. На рисунку 3.11 показаний результат реалізації механізму прийняття рішень, який має три зважені значення WM, WP і WE, що представляють значення вагомості пам'яті, продуктивності та економічної вартості відповідно. Значення ваги вказують на пріоритет кожного параметра, за замовчуванням ми вважали продуктивність найвищим пріоритетом, тобто 70%, потім пріоритет пам’яті, тобто 20%, а потім економічні витрати, тобто 10%. Однак, якщо є вимога змінити пріоритет на основі інформації про контекст пристрою, то Механізм прийняття рішень зробить це динамічно. Таким чином, значення ваги змінюються для обробки різних одиниць параметрів. Сума значень ваги завжди буде 100%.

На рисунку 3.11 також є така інформація, як підключення до Wi-Fi (нижній лівий кут), вибір користувача на попередньому кроці (нижній лівий кут), бездротовий SSID та пам’ять, яка використовується на пристрої. Локальна кінцева точка — це локальна (на пристрої) URL-адреса служби. Хмарна кінцева точка — це URL-адреса служби AWS (Amazon Web Service). Інформація про AWS буде надана в наступному розділі. Значення plocal і pcloud отримуються після натискання кнопок Test Local і Test Cloud відповідно.

Тут p.local = W/C, де W — робоче навантаження служби, а C — потужність ЦП, p.cloud = TR +RT +TO, де RT — час відгуку хмарного сервісу;

TR – час завантаження вхідних даних = розмір вхідних даних/швидкість передачі даних;

TO – час завантаження вихідних даних = розмір вихідних даних/швидкість передачі даних; Економічна вартість для хмари — E.Cloud = DI + DO, де DI — розмір вхідних даних виклику служби;

DO — це розмір вихідних даних виклику служби.

Оскільки доступна пам’ять у даному сценарії становить 30%, значення ваги не змінюються.

Вся ця інформація розміщується в цільовій функції:

$$OF = P(s1,s2,...,sn)*WP + M(s1,s2,...,sn)*WM + E(s1,s2,...,sn)*WE,$$

На основі цього приймається рішення, перевантажувати сервіс в хмару чи ні. Тут значення локальної кінцевої точки вказує на URL-адресу служби, щодо якої ми приймаємо рішення. У цій ситуації URL-адреса призначена для окремої служби, але якщо потрібно, ми можемо додати крапку з комою та написати іншу URL-адресу, щоб прийняти рішення про розвантаження. У цьому випадку він спочатку виконуватиме службу один за іншим. Після цього він розраховує рішення для кожної окремої послуги. Отже, після отримання інформації для локальної та хмарної, рішення надається.

У даному сценарії Wi-Fi підключення доступне, тому вартість не розраховується. З результатів цільової функції ми бачимо, що результат вказує на те, що служба повинна виконуватися локально.

Сценарій 2: підключення до Wi-Fi, пріоритет пам’яті. Щоб приймати рішення в режимі реального часу на основі складних наборів даних, система штучного інтелекту самокерованого автомобіля потребує постійного, безперервного потоку даних та інструкцій. Це призводить до зменшення доступної пам’яті. Отже, у другому сценарії ми вважаємо, що пам’ять менша за порогове значення. Таким чином, пріоритет надається ваговому значенню пам’яті, і рішення приймається на основі цього значення ваги.

Сценарій 2 показаний на рисунку 3.12, де підключено Wi-Fi, тому вартість не буде розрахована. Значення доступної пам’яті становить лише 17%, і оскільки воно нижче порогового значення (25%), значення ваги змінюються. У цьому випадку пріоритет надається WM, отже, його значення дорівнює 0,6, а значення WP — 0,3. Після підрахунку всіх значень результат показує, що краще розвантажити сервіс і виконати його в хмарі.

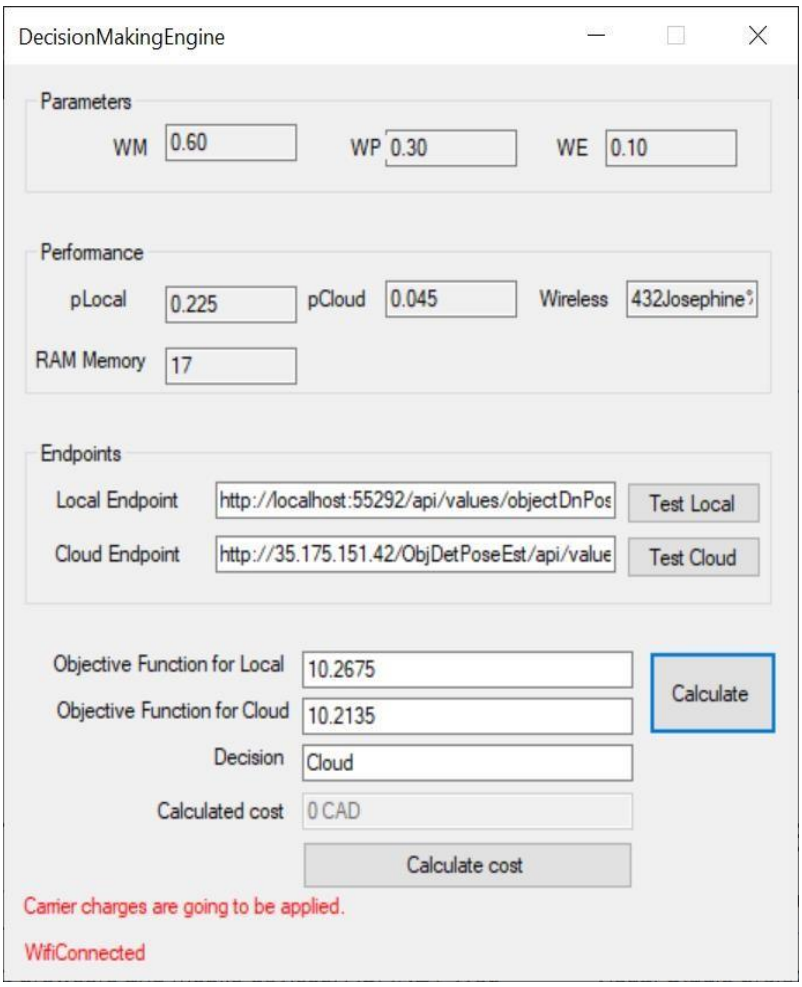


Рисунок 3.12: Сценарій механізму прийняття рішень 2

Сценарій 3: Wi-Fi не підключено, використовуйте мобільні дані. Оскільки самокеровані автомобілі подорожують різними регіонами, підключення до мережі не завжди доступне. У ситуації, коли автомобіль їде в тунель, підключення для системи буде недоступне. Тому ми повинні враховувати такі ситуації та бути готовим до належного виконання послуг та безпеки користувачів. У таких ситуаціях для роботи служб використовуються мобільні дані.

У сценарії 3 (Рисунок 3.13) доступна пам’ять перевищує порогове значення, тому значення ваги не змінюються. Однак з’єднання Wi-Fi недоступне, тому розраховується вартість розвантаження. На основі вибору користувача ми маємо змінну IsCarrierChargesAllowed, яка має значення true або false.

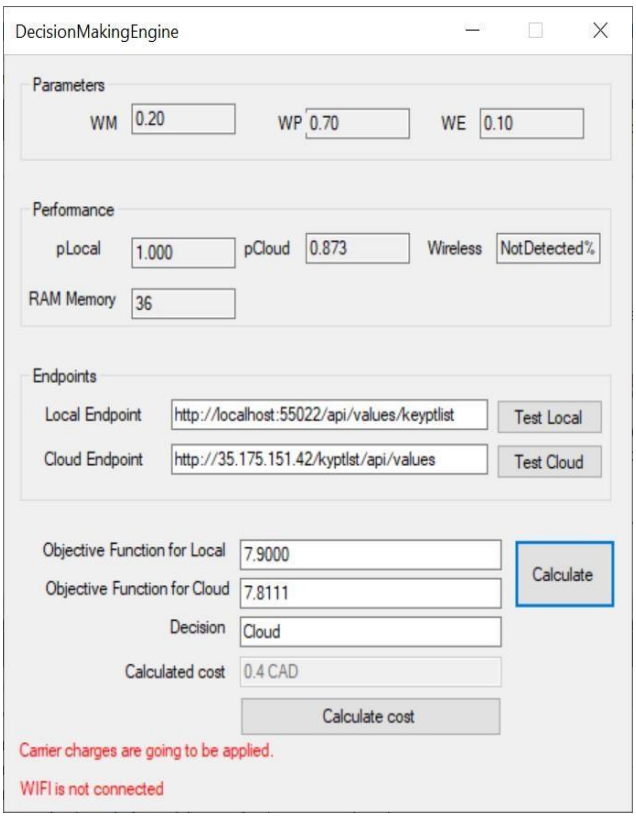


Рисунок 3.13 - Сценарій 3 механізму прийняття рішень

Тут у нижньому лівому куті позначка «Буде стягуватися плата за оператора» вказує, що значення IsCarrierChargesAllowed = true. Отже, плата розраховується за кодом, показаним на рисунку 3.14. Згідно з даними, використаними в МБ, розрахована вартість становить 0,4 CAD, а рішення про виконання послуги — хмарне. Таким чином, плата буде додана для користувача, а послуга буде виконуватися в хмарі.

```
//START Cost calculation
originalCharge = totalUsedmb;
CalculateDataCost();
BeforeCharge= totalUsedmb;
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
CalculateDataCost();
AfterCharge = totalUsedmb;
originalCharge = originalCharge + (AfterCharge - BeforeCharge);
totalUsedmb = originalCharge;
txtCalculatedCost.Text = Convert.ToString(totalUsedmb * dbWC) + " CAD ";
//END Cost calculation
```

Рисунок 3.14 - Код для калькуляції вартості

Сценарій 4: Wi-Fi не підключено, виконуйте локально – оскільки послуги та важливі дані зберігаються на бортовому блоці автомобіля, послуги можна виконувати локально без використання Інтернету, навіть якщо підключення Wi-Fi недоступне. У цих випадках продуктивність перевіряється механізмом прийняття рішень. Якщо продуктивність краща на локальному, тоді служба виконується локально, і мобільні дані не використовуватимуться.

Сценарій 4, показаний на рисунку 3.15, багато в чому схожий на сценарій 3. Єдина відмінність полягає в тому, що продуктивність на локальному рівні є кращою, що вказує на те, що служба буде виконуватися локально. Таким чином, плата не буде додана для користувача.

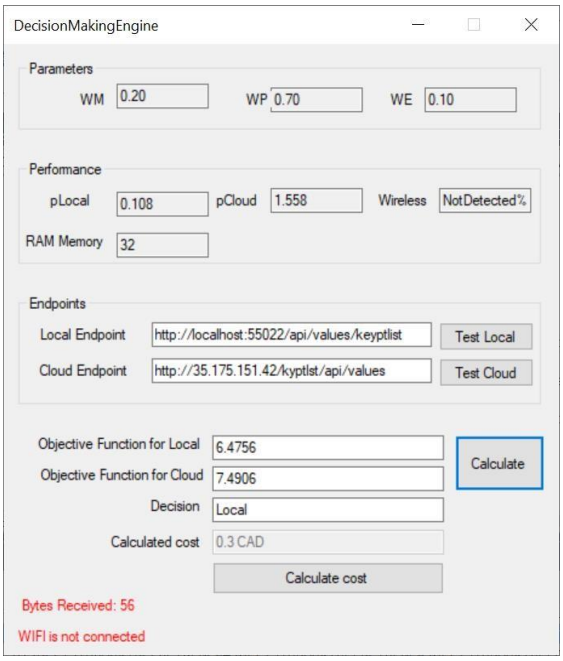


Рисунок 3.15 - Сценарій 4 механізму прийняття рішень

Сценарій 5: користувач відхилив додавання плати за передачу даних – як згадувалося раніше в цьому розділі, спочатку користувача запитають, чи хочуть вони дозволити додавання або плату за передачу даних у разі відключення мережі. Якщо Wi-Fi або дані доступні, то послуга може бути виконана в хмарі (на основі контекстної інформації), але якщо вона недоступна

і користувач відхилив додавання плати, то послуга буде виконуватися лише локально.

У сценарії 5 користувач відмовився включити додаткові витрати, тому значення IsCarrierChargesAllowed — false. Таким чином, мітка вказує, що «Збори оператора не стягуються», а кнопка обчислення вартості вимкнена. Крім того, з'єднання Wi-Fi недоступне. На основі обох умов послуга буде надаватися локально.

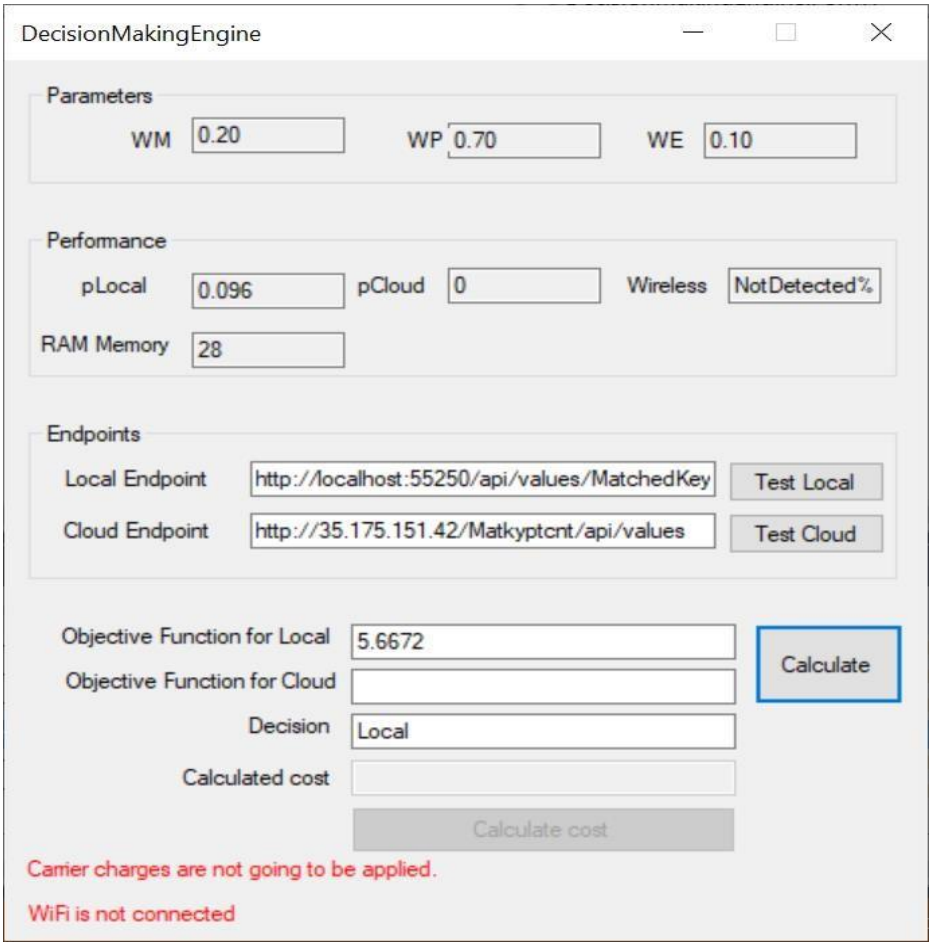


Рисунок 3.16 - Сценарій механізму прийняття рішень 5

Це були деякі зі сценаріїв, які можуть статися під час виконання. Наступний розділ описує процес, який матиме місце, якщо результатом роботи механізму прийняття рішень буде виконання служби в хмарі.

3.6 Процес використання Communication Manager і Offloading Planner

Для розвантаження сервісу використовується хмарний ресурс Elastic Compute Cloud (EC2) AWS. Він забезпечує змінний розмір обчислювальної потужності, дозволяючи користувачам створювати сервер у хмарі. Для цієї дипломної роботи було створено обліковий запис AWS Educate, оскільки AWS Educate є безкоштовним протягом року для студентів. Для створення екземпляра EC2 був використаний Windows AMI, як показано на рисунку 3.17.

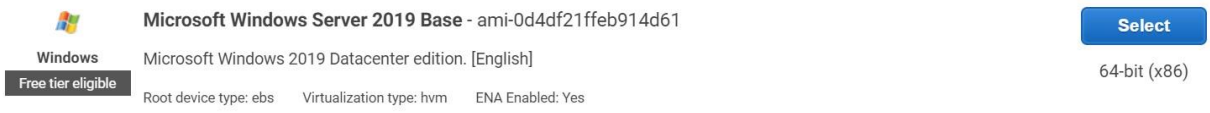


Рисунок 3.17 - Створення AWS EC2 Windows Server

IIS (Інтернет-інформаційні служби) встановлено та налаштовано так, щоб служби можна було виконувати на екземплярі EC2. З міркувань безпеки в AWS за замовчуванням заборонені вхідні запити (крім RDP) до цього сервера. RDP дозволено для підключення до сервера, щоб можна було вносити зміни, коли це потрібно. Отже, після створення екземпляра необхідно налаштувати групи безпеки. Як показано на рисунку 3.18, порти HTTP і HTTPS додаються, щоб API можна було викликати за допомогою їхніх URL-адрес.

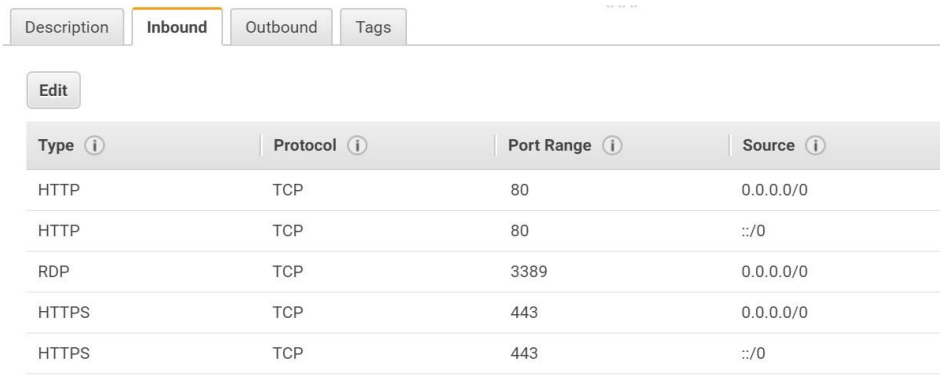


Рисунок 3.18 - Конфігурація груп безпеки

Communication Manager відповідає за взаємодію мобільних і хмарних мереж через з'єднання TCP-сокета через Wi-Fi або стільникову мережу. Після отримання інформації від механізму прийняття рішень, якщо послугу потрібно вивантажити в хмару, між пристроєм і хмарою відбувається зв'язок. Потім Планувальник розвантаження вибирає послугу, яку необхідно вивантажити.

Багато сучасних досліджень розвантаження обчислень для мобільних хмарних обчислень зосереджуються на техніках поділу та розвантаження коду, однак, коли самокерований автомобіль рухається, контекстна інформація, як-от місцезнаходження, доступні хмарні ресурси та умови мережі, змінюється. Тому для цих транспортних засобів надзвичайно важливо мати можливість динамічно розвантажувати програми і, отже, працювати ефективним і безпечним способом. Adiththan та ін. представляє підхід до адаптивного розвантаження для автоматизованого водіння. Автор розглянув умови мережі та вимоги до програми для використання віддалених ресурсів. Параметрами, які розглядаються в цьому методі, є характеристики мережі, вимоги до передачі даних і вимоги до обчислень. Однак важливий фактор, тобто грошова вартість, у цій роботі не враховується. Оскільки розвантаження служби здійснюється в хмару, важливо враховувати вартість, щоб зрозуміти її вплив на якість досвіду (QoE) для користувача. Є багато інших цікавих літератур, в яких використовуються такі методи навчання, як навчання з підкріпленням або адаптивне навчання, але у випадку з автономними автомобілями на дорогах постійно виникають нові ситуації. Тому не завжди безпечно використовувати ці методи навчання. Загалом, з експериментальних результатів і порівняння можна зробити висновок, що наша структура є надійним способом забезпечення обчислювальної розвантаження для додатків для самокерованих автомобілів.

ВИСНОВКИ

На даний час зростає використання мобільних пристроїв, таких як смартфони, планшети та ноутбуки, у різних сферах, таких як ігри, інформаційні системи, електронне навчання та охорона здоров'я. Проте існують різні обмеження для таких пристроїв, як-от потенціал процесора, час автономної роботи, ємність пам'яті та підключення до мережі. Незважаючи на те, що мобільні пристрої сьогодні мають велику пам'ять і швидкі процесори, цього недостатньо для виконання обчислювальних і ресурсомістких завдань, таких як розпізнавання зображень, обробка природної мови та виявлення об'єктів.

Таким чином, мобільні хмарні обчислення використовуються для подолання розриву між обмеженнями мобільних пристроїв і вимогами додатків. Це інфраструктура, яка може розширити сховище та обчислювальні можливості мобільних пристроїв, використовуючи необхідні хмарні ресурси. Цього можна досягти за допомогою Computational Offloading, що дозволяє виконувати обчислювальні та ресурсомісткі завдання на віддаленому сервері або хмарних ресурсах.

Хоча існують різноманітні платформи, що вирішують розвантаження обчислень, деякі підходи використовують статичне розвантаження, деякі не мають стандартної архітектури, а деякі є неефективними через клонування та високі накладні витрати на керування.

Отже, метою запропонованої системи, в даній дипломній роботі бакалавра є цілісна архітектура з використанням сервіс-орієнтованої архітектури, в якій обчислювальні та інтенсивні завдання окремих додатків вивантажуються з мобільного пристрою на хмарний сервер.

У цьому методі завдання виконуються віддалено як індивідуальний сервіс на хмарному сервері. Крім того, фреймворк розроблено для прийняття рішень щодо розвантаження на основі контексту пристрою.

Реалізація завдань програми як сервісів виконується на екземплярі, який може використовуватися будь-якою програмою на клієнті. У цій системі

враховуються такі фактори, як пам'ять, продуктивність, економічні витрати та вартість. Однак інші структури враховують деякі з цих факторів, але не всі в одній системі.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Abrial, J.R., Schuman, S.A., Meyer, B.: Specification language. In: On the Construction of Programs. (1980) 343–410

2. Jones, C.: Systematic Software Development using VDM. Prentice Hall (1990)

3. High, R., Kinder, S., Graham, S.: IBM's SOA Foundation – An Architectural Introduction and Overview. Technical Report 1.0, IBM (2005)

4. Beisiegel, M., Booz, D., Edwards, M., Hurley, O., Ielceanu, S., Karmarkar, A.,Malhotra, A., Marino, J., Nally, M., Newcomer, E., Patil, S., Pavlik, G., Rowley,M., Tam, K., Vorthmann, S., Waterman, L.: Service Component Architecture– Assembly Model Specification. SCA Version 0.96 draft 1, August 2006, BEA,Cape Clear, IBM, Interface21, IONA, Oracle, Primeton, Progress Software, RedHat, Rogue Wave, SAP, Software AG., Sybase, TIBCO (2006)

5. McIlroy, M.D.: Mass Produced Software Components. In Naur, P., Randell, B.,eds.: Proceedings of NATO Software Engineering Conference. Volume 1., Garmisch,Germany (1968) 138–150

6. Szyperski, C.: Component Software–Beyond Object-Oriented Programming.Addison-Wesley and ACM Press (1998)

7. Messerschmitt, D., Szyperski, C.: Software Ecosystem–Understanding an Indispensable Technology and Industry. MIT Press (2003)

8. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. 2 edn.Addison Wesley Professional (2003)

9. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Candidate Recommendation 27 March 2006, W3C (2006)

10. White, S.A.: Business Process Modelling Notation Version 1.0. Technical report,Business Process Management Initiative (BPMI) (2004)

11. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business

Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, International Business Machines Corporation, Microsoft Corporation (2003)

12. B¨orger, E., St¨ark, R.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer-Verlag (2003)

13. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University

Press (1996)

14. Christensen, E., Curbera, F., Meredith, G., Weeravarana, S.: Web Service Discription Language (WSDL) 1.1. W3C Note 15 March 2001, Ariba, International Business Machines Corporation, Microsoft (2001)

15. Object Management Group: UML2.0 Object Constraint Language (OCL) Specification. Specification, Object Management Group (OMG) (2003)

16. Aalst, W.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1) (1998) 21–66

17. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo (1985)

18. Billington, J., Christensen, S., Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In Aalst, W., Best, E., eds.: Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings. Volume 2679 of Lecture Notes in Computer Science., Springer (2003) 483–505

19. Beatty, J., Blohm, H., Boutard, C., Brodsky, S., Carey, M., Dubray, J.J., Ellersick, R., Ho, M., Karmarkar, A., Kearns, D., Brettevillois, R.L., Nally, M., PreotiucPietro, R., Rowley, M., Smith, S., Yan, H.: Service Data Objects For Java. Specification, Version 2.01, IBM, BEA, Oracle, SAP, Siebel, Xcalia, Sybase

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: "Методологія сервіс-орієнтованого балансування мобільних обчислень"

Обсяг пояснювальної записки: 66 аркушів

Дата закінчення дипломної роботи 13 червня 2021р.

Підпис студента _____