КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою "Інформатика"
спеціальності 122 Комп'ютерні науки на тему:

РЕАЛІЗАЦІЯ СИСТЕМИ ПАРКС ДЛЯ МОВИ PYTHON ЗГІДНО СУЧАНИХ СТАНДАРТІВ

Виконав студент 4-го курсу
Дмитро ТЄЛЬЦОВ _____

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Олексій ФЕДОРУС _____

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

Протокол № _____ _____ 2023 р.

Завідувач кафедри
В. М. Терещенко _____

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи: 40 сторінок, 14 ілюстрацій, 15 використаних джерел.

Ключові слова: ПАРКС, PYTHON, AIOHTTP, ПАРАЛЕЛЬНІ АСИНХРОННІ РЕКУРСИВНО КЕРОВАНІ СИСТЕМИ, GOOGLE CLOUD.

Об’єктом роботи є сукупність програмних засобів, що забезпечують процес розробки і реалізації алгоритмів паралельної обробки інформації.

Метою кваліфікаційної роботи є створення системи, яка дозволяла би реалізовувати паралелізацію алгоритмів з мінімальними зусиллями з боку користувача та, в разі потреби, легкого розширення цієї системи додатковим функціоналом.

Інструментом створення є безкоштовний, вільно поширюваний редактор коду Pycharm community edition та мова розмітки тексту Typst, мова програмування Python. Використано фреймворк aiohttp, бібліотеку google-cloud-compute та інструмент контейнеризації Docker.

Результат роботи: створено систему, яка дозволяляє реалізовувати паралелізацію алгоритмів.

ЗМІСТ

ВСТУП

Швидкий розвиток інформаційних технологій вимагає створення зручних засобів для паралельного програмування та суміжних технологій. Терміни «паралельний», «асинхронний», «конкурентний» все частіше використовуються по відношенню до архітектури обчислювальних систем, операційних систем, алгоритмів, мов програмування, структур даних і баз даних.

Все більшим і більшим стає попит на нейромержі, моделювання економічних процесів та інші задачі, які мають численні розрахунки на великих масивах даних.

Тому дуже важливо мати зручний інструмент для вирішення подібних задач з можливістю паралельних обчислень. Таку область задач покриває система ПАРКС, яка була розроблена на факультеті Комп'ютерних наук та Кібернетики Київського Національного Університету імені Тараса Шевченка.

Система ПАРКС [1] може сприяти підвищенню продуктивності та ефективності обчислювальних процесів у різних галузях, таких як:

- Наукові дослідження: Система може бути корисною у наукових дослідженнях, де потрібно виконувати складні обчислення, аналізувати великі обсяги даних або моделювати складні процеси.
- Фінансовий сектор: У фінансовій галузі система ПАРКС може бути використана для обробки та аналізу фінансових даних, розрахунків ризику, прогнозування ринкових трендів тощо.

- Інтернет речей: У цій галузі система ПАРКС може бути використана для обробки та аналізу великого обсягу даних, керування та оптимізації розподіленими мережами сенсорів, аналітики даних з пристроїв.

Вже існує реалізація ПАРКС на Python, яка використовує застарілі бібліотеки та має проблеми з архітектурою, що може викликати проблеми у використанні.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

КП – Керуючий простір;

GCP – Google Cloud Platform;

АМ – Алгоритмічний модуль;

REST – Representational State Transfer, Передача представницького стану;

RPC – Remote Procedure Call, Віддалений виклик процедури;

VMI – Virtual machine instance, об’єкт віртуальної машини;

РОЗДІЛ 1. ОГЛЯД СИСТЕМИ ПАРКС

1.1. Означення

ПАРКС - це набір програмних інструментів, які дозволяють розробляти та реалізовувати алгоритми паралельної обробки інформації. Головною особливістю ПАРКС є поєднання двох важливих алгоритмічних концепцій - рекурсії та паралельності [1].

За допомогою цих інструментів складний паралельний процес можна представити як виконавчу діяльність, яка розвивається та спілкується в деякому логічно пов’язаному просторі.

Кожна діяльність процесу має свої просторові координати, які визначають комунікаційні адреси. Така структура комунікації називається керуючим простором.

1.2. Керуючий простір

Керуючий простір у системі ПАРКС є структурою, яка складається з адресних точок та каналів [2]. Адресні точки можуть бути постійними або змінними і визначають місце розташування окремих елементів системи в просторі. Канали ж забезпечують можливість передачі даних та іншої інформації між різними елементами системи [3].

Важливою особливістю керуючого простору є те, що передача даних та іншої інформації може здійснюватися тільки через канали. Це дозволяє забезпечити максимальну ефективність та швидкість обміну даними в системі,

оскільки всі елементи системи знаходяться в просторі і мають доступ до каналів для обміну даними [4].

1.3. Алгоритмічний модуль

Алгоритмічний модуль є ключовою особливістю системи ПАРКС. Він представляє собою програму на базовій мові, розширеній спеціальними командами, які дозволяють взаємодіяти з КП. АМ дозволяє виконувати високорівневі операції зі збору, обробки та передачі даних, що дозволяє значно спростити розробку та підтримку системи [3].

Завдяки АМ, система ПАРКС може бути простішою та ефективнішою у порівнянні з іншими системами. АМ дозволяє автоматизувати багато операцій, що знижує ризик помилок та забезпечує швидше виконання завдань.

Додатково, АМ може бути змінений та доповнений відповідно до потреб користувачів. Це дозволяє забезпечити гнучкість та адаптивність системи до змінних умов та завдань. Крім того, АМ є важливим елементом системи для забезпечення безпеки та захисту від несанкціонованого доступу.

1.4. Порівняння ПАРКС з аналогами

Система ПАРКС є однією з найбільш ефективних систем для паралелізації алгоритмів на сьогоднішній день. Порівняно з аналогами, такими як MPI, OpenMP та іншими, ПАРКС має декілька переваг:

- Зручний інтерфейс для опису паралельних алгоритмів, що дозволяє програмістам швидко та ефективно перетворювати послідовні алгоритми на паралельні. Це зменшує час, необхідний для написання та налагодження паралельних програм.
- Ефективне використовувати розподілені ресурси для обчислень, такі як кластери та суперкомп'ютери. Це забезпечує збільшення швидкості обчислень та зниження часу, необхідного для вирішення складних завдань.
- Різні види паралельних обчислень, такі як розподілені та локальні обчислення, що дозволяє досягати оптимальної продуктивності та швидкості обчислень в залежності від вимог конкретного завдання.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧОЇ РЕАЛІЗАЦІЇ ПАРКС ДЛЯ МОВИ PYTHON

2.1. Загальна схема роботи

Користувач створює VMI master та VMIs worker. Далі він переходить за зовнішньою адресою об'єкту master. Через master можна взаємодіяти з системою:

- Дивитись на стан задач
- Створювати задачі на виконання, надсилаючи файли з алгоритмом та вхідними даними
- Дивитись на стан об'єктів worker
- Маніпулювати станом об'єктів worker

2.2. Структура проекту

Структура проекту - це спосіб організації файлів і папок, який визначає, як різні частини проекту будуть логічно пов'язані між собою та як вони будуть організовані у відповідності зі стандартами та кращими практиками. Важливість структури проекту полягає в тому, що вона дозволяє розробникам ефективно працювати з кодом, роблячи його легше зрозумілим, підтримуваним і розширюваним у майбутньому.

Але перше, з чим зіштовхується розробник, який відкриває репозиторій [5] з імплементацією ПАРКС мовою Python, є відсутність зрозумілої структури. Всі файли знаходяться в одній папці, що робить їх пошук, редагування складнішими та витрачає багато часу. Також, складно визначити, яка які модуль пов'язані логікою, а які ні.

Наприклад, в файлику node.py маємо реалізацію різних видів об'єкту Node, що є логічним, і зовні нелогічним є те, що там знаходиться імплементація трьох різних видів потоків. З одного боку зручно, що усе в одному файлу і не треба думати над компонуванням та імпортами, але з іншого боку в цьому файлі стає складніше орієнтуватись.

Крім того, відсутність документації та коментарів у коді робить розуміння функціоналу ще більш складним. Не зрозуміло, який саме код відповідає за яку функціональність, а назви не дають достатньої інформації про їх вміст.

Окрім цього, в одній папці присутні файли з імплементацією функцій вузла, апі, та окремих утіліт, що робить розуміння логічного взаємозв'язку між цими компонентами ще складнішим, а саме: який компонент є для внутрішніх задач модуля, а який пристосований до того, щоб використовуватись в інших модулях.

Необхідно досить детально вивчати код, щоб зрозуміти, які файли відповідають за який функціонал та які залежності між компонентами існують.

2.3. Архітектура системи

Загалом система побудована на веб-фреймворку Flask, що базується на архітектурі REST та фреймворку Pyro4, що базується на архітектурі RPC.

У REST архітектурі, веб-сервіси розглядаються як ресурси, до яких можна здійснювати доступ та взаємодіяти з ними за допомогою стандартних HTTP-методів [6], а у RPC основна ідея полягає в тому, що клієнтська програма і серверна програма мають спільний інтерфейс, який описує доступні функції

та параметри, що можуть бути передані. Клієнтська програма може викликати віддалені функції, передаючи необхідні параметри, а серверна програма виконує ці функції та повертає результат клієнту [7].

Таке поєднання архітектур є дуже вигідним для системи ПАРКС, бо RPC створена для розподілених систем, а REST підходить для обробки клієнтських запитів. Але, саме у цих фреймворків є свої недоліки:

- Відсутність вбудованої підтримки асинхронності: Flask [8] та Pyro4 [9] є синхронними, що означає, що вони використовують блокуючі операції вводу/виводу. У системі ПАРКС, де розподілені обчислення та взаємодія між елементами системи відбувається паралельно, асинхронність є важливою характеристикою.
- Обмежена масштабованість: Flask є легковаговим фреймворком, що дозволяє швидко створювати прості веб-додатки. Однак, для складних систем, як система ПАРКС, можуть знадобитися більш потужні інструменти для масштабування, такі як фреймворки, що підтримують горизонтальне масштабування, розподілені обчислення та навантажувальне тестування.Pyro4 був розроблений переважно для використання в невеликих до середніх масштабних додатках. Якщо система ПАРКС передбачає велику кількість вузлів та інтенсивний обмін даними, Pyro4 може стати обмеженням у плані масштабованості та продуктивності.
- Застарілість: Pyro4 не отримує таку активну підтримку та розробку. Це може призвести до відсутності нових функцій, виправлень помилок та підтримки нових версій Python або залежних бібліотек. Через це бібліотека має меншу спільноту користувачів. Це означає, що можливості знайти підтримку, документацію та приклади використання можуть бути обмеженими

2.4. Архітектура проекту

Основна мета архітектури проекту полягає в тому, щоб створити гнучку та масштабовану структуру, яка може витримати змінні вимоги. Добре розроблена архітектура дозволяє розробникам швидко розгортати нові функції та виправляти помилки, забезпечуючи високу якість та ефективність коду. Крім того, гарна архітектура проекту дозволяє знизити витрати часу на розробку та збереження програмного забезпечення, оскільки спрощує процес розробки та підтримки.

Правильна архітектура проекту також допомагає зменшити ризики, пов'язані зі збоїв та помилками. Вона дозволяє забезпечити високу надійність та безпеку системи, що важливо для захисту даних та забезпечення стійкості роботи програмного забезпечення.

У файлі parcs.py з репозиторію [5] знаходиться багато функцій-обробників запитів, як для веб-сторінок, так і для внутрішніх потреб. Це призводить до того, що весь функціонал, який стосується як об'єкта master, так і об'єктів worker, зібраний в одному місці.

Такий підхід йде з ідеї того, що користувач повинен використовувати один docker image при створенні об'єктів. Однак, незважаючи на це, користувач все одно повинен вказувати параметри при створенні об'єкту, щоб скрипт конфігурації VMI міг розрізнити, чи це об'єкт master, чи worker. Тому логічніше було б розділити функціонал об'єктів master та worker на два окремі модулі.

Це дозволило б досягти більшої чіткості та структурованості коду. Розділивши функціонал на два модулі, кожен з них міг би концентруватися

на своїх специфічних завданнях і нести відповідальність лише за обробку відповідних запитів. Такий підхід полегшив би розробку, тестування та зміну функціоналу кожного об'єкта окремо, а також зробив би код більш зрозумілим та підтримуваним.

Також, у системі існує велика кількість залежностей між класами, що ускладнює розширення функціоналу та розуміння роботи системи. Один з прикладів такої залежності можна знайти у класі WorkerNode, де ініціалізація цього класу залежить від об'єкту класу Config. Крім того, в самому класі WorkerNode ініціалізація внутрішнього RPC потоку RPCThread також залежить від Config.

Отже, якщо хтось вносить зміни до класу Config, існує ризик того, що не тільки клас WorkerNode, але й RPCThread можуть бути зламані. Це означає, що навіть незначні зміни у класі Config можуть мати далекосяжні наслідки та вимагати широкого рефакторингу коду у різних частинах системи.

Ця сильна залежність між класами ускладнює розширення та зміну функціоналу системи, оскільки невеликі зміни можуть впливати на велику кількість компонентів. Такий підхід утруднює розуміння та налагодження системи, а також збільшує ризик введення помилок при роботі з класами та їх взаємодією.

В жодному класі немає розділення на публічні, захищені та приватні методи та поля. Нехай для прикладу ми розглянемо клас WorkerNode. Усі поля та методи в цьому класі є публічними, хоча деякі з них, наприклад, поле reconnector, насправді не використовуються за межами класу. Це призводить

до втрати чіткості та розуміння щодо того, які поля слугують внутрішнім потребам класу, а які можуть бути використані зовні без впливу на логіку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ

3.1. Загальна схема роботи

Користувачу доступний docker image з реалізацією VMI master. Користувач створює об'єкт master. Далі він переходить за зовнішньою адресою об'єкту master. Через master можна взаємодіяти з системою:

- Дивитись на стан задач
- Створювати задачі на виконання, надсилаючи файли з алгоритмом та вхідними даними
- Дивитись на існуючі об'єкти worker
- Видаляти та створювати об'єкти worker

Головною особистістю цієї реалізації є те, що користувач ніяк не налаштовує об'єкти worker. Цим займається об'єкт master. Таким чином користувач витрачає набагато меньше часу на налаштування системи та може дуже швидко додавати нові об'єкти worker.

3.2. Структура проекту

У новій реалізації були створені дві окремі директорії для функціоналу об'єктів master та worker, що дозволяє краще розуміти роботу системи та забезпечує більш прозору структуру проекту.

Перша директорія, parcs_master, містить весь функціонал, пов'язаний з об'єктом master. Вона поділена на три окремі директорії, відповідно до їх функціонального значення:

- api, де знаходяться всі функції, пов'язані з мережевими задачами, такими як приймання та обробка HTTP-запитів від об'єктів worker та веб-клієнта.
- cloud, де зосереджено функції, пов'язані з контролем об'єктів worker.
- job, де знаходяться всі функції, пов'язані з контролем задач, які ставить користувач.

Друга директорія, parcs_node, містить весь функціонал, пов'язаний з конкретним об'єктом worker. Вона також розбита на дві окремі директорії:

- api, де знаходяться функції, пов'язані з мережевими задачами, такими як приймання та обробка HTTP-запитів від master.
- job, де знаходяться функції, пов'язані з виконанням та контролем задач, які ставить користувач.

Така структура проекту дозволяє мати чітку та логічну організацію всіх компонентів. Кожен модуль розташований у відповідній директорії, що спрощує пошук та зміну коду. Це також полегшує розширення функціоналу системи для майбутніх розробників, оскільки все знаходиться на своєму місці та має чіткі границі.

3.3. Архітектура системи

Вся система побудована за допомогою асинхронному веб-фреймворку aiohttp, який базується на архітектурі REST та плагіну для цього фреймворка aiohttp_rpc, який надає можливість будувати сервіси на основі RPC. Поєднання REST та RPC для реалізації системи ПАРКС має кілька переваг:

- Гнучкість: Використання REST дозволяє легко взаємодіяти з системою за допомогою HTTP-протоколу. REST API надає стандартизований спосіб отримання, створення, оновлення та видалення ресурсів у системі. Завдяки цьому, ви можете легко розширювати функціонал системи та взаємодіяти з нею за допомогою різних клієнтських програм або бібліотек.
- Простота інтеграції: RPC дозволяє викликати віддалені процедури із віддалених об'єктів. Це дає можливість реалізувати складну логіку на більш високому рівні абстракції і викликати ці процедури за допомогою RPC-викликів. Це спрощує розробку та інтеграцію різних компонентів системи ПАРКС.
- Ефективність: Використання RPC дозволяє здійснювати пряму взаємодію між компонентами системи без зайвого серіалізації-десеріалізації даних, що може бути витратним з точки зору продуктивності. Це особливо важливо у системах ПАРКС, де швидкодія та ефективність обробки завдань є критичними.
- Модульність: Поєднання REST та RPC дозволяє розділити функціонал системи на окремі сервіси або компоненти, які можуть бути реалізовані та масштабовані незалежно один від одного. Це дає можливість гнучко налаштовувати систему, додавати нові функціональні можливості та масштабувати окремі компоненти системи за потребою.

Сам фреймворк aiohttp має такі переваги [10]:

- Асинхронність та швидкодія: aiohttp базується на асинхронному програмуванні та використовує event loop, що дозволяє обробляти багато одночасних запитів без блокування інших процесів. Це дозволяє

забезпечити високу продуктивність та швидку відповідь системи, особливо при великому обсязі запитів та розподілених вузлах.

- Вбудована підтримка WebSocket: aiohttp має вбудовану підтримку WebSocket, що дозволяє взаємодіяти з веб-клієнтами в реальному часі. Це важливо для системи ПАРКС, яка може використовувати активний зв'язок з вузлами, обмінюючи даними та виконуючи контрольні операції.
- Простота розробки та розширення: aiohttp має простий та зрозумілий синтаксис, що спрощує розробку та підтримку системи. Він також має багатий набір функцій та розширень, які дозволяють легко налаштувати та розширити функціонал системи ПАРКС згідно з вимогами проекту.
- Підтримка масштабованості: aiohttp дозволяє горизонтальне масштабування системи шляхом розподілу навантаження між багатьма вузлами. Використовуючи механізми розподіленого обчислення та маршрутизації в aiohttp, можна створити масштабовану систему ПАРКС, яка забезпечує ефективне використання ресурсів та розподіл завдань між вузлами.

Також, для роботи з об'єктами worker було використано бібліотеку google-cloud-compute, бо за базову платформу було обрано GCP, але при бажанні можна додати підтримку інших платформ, як приклад, Amazon чи Azure. Використання цієї бібліотеки спрощує маніпуляції з об'єктами worker, бо надає публічний інтерфейс, який скриває від розробника всі тонкощі роботи.

3.4. Архітектура parcs_master

Основою всієї системи є REST веб-сервіс, який обробляє запити від користувача. Також, в цьому веб-сервесі є два класи-контроллери: один з них, GoogleCloudController, імплементує роботу з GCP та усього пов'язаного з VMIs, а другий, JobsController, займається контролем задач. Таким чином веб-сервіс не знає тонкощів роботи контролерів та користується виключно публічними методами. Спочатку, варто розібратись в роботі контролерів, щоб при розборі роботи веб-сервіса був контекст.

3.4.1. GoogleCloudController

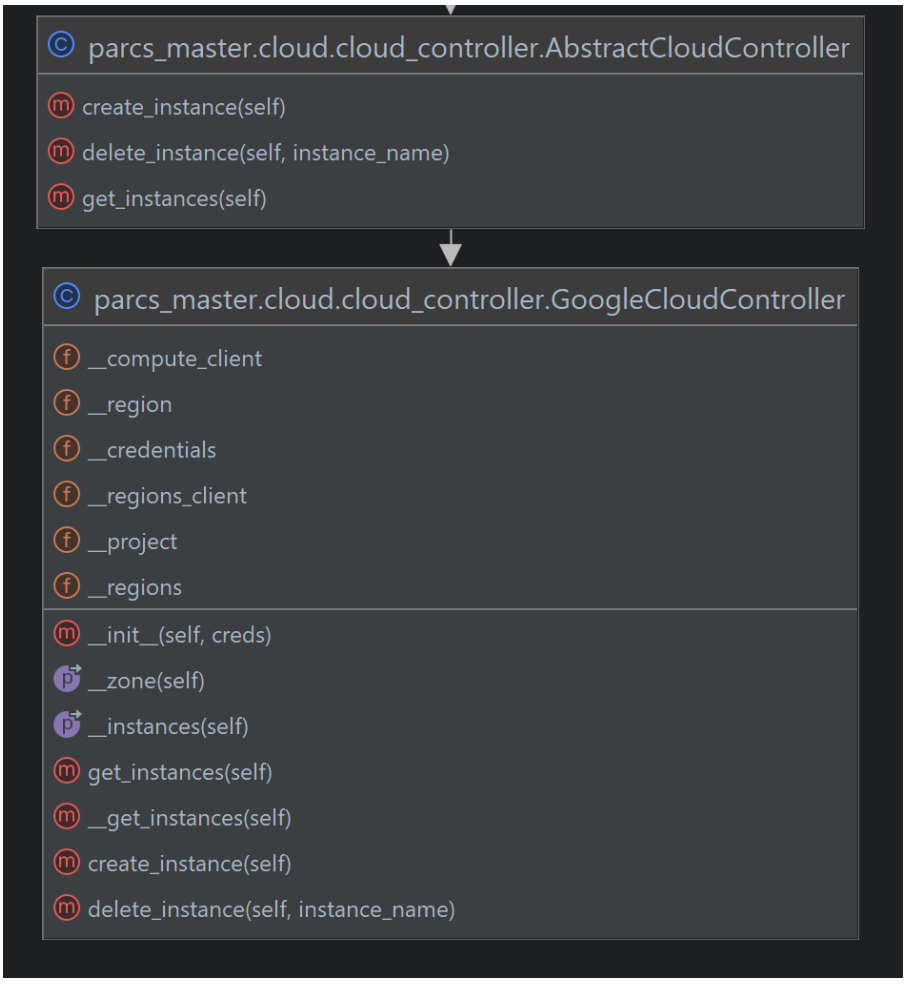

Рисунок 1 – UML-діаграма GoogleCloudController

У класі GoogleCloudController усі поля приватні, тобто поле може бути доступне лише в межах самого класу, а не в будь-яких інших класах або коді зовні [11]. Це зроблено для того, щоб скрити логіку від інших класів та об'єктів в коді, а надавати лише публічний інтерфейс взаємодії [12]. Поля наступні:

- credentials - зберігає необхідні, для авторизації, дані
- compute_client - зберігає об'єкт InstancesClient, який бібліотека google-cloud-compute надає для взаємодії з VMI
- regions_client - зберігає об'єкт RegionsClient, який бібліотека google-cloud-compute надає для отримання регіонів
- project - id проекту з яким працює користувач
- regions - генератор, який при виклику віддає наступний регіон
- region - зберігає поточний регіон
- zone - зберігає поточну зону
- instances - зберігає наявні VMI, які загорнуті в об'єкти класу Instance для зручного доступу до атрибутів з якими працюють методи

Сам клас Instance по своїй суті є дата-класом, тобто орієнтований на зберігання різної інформації, а не на якусь поведінку. Він має наступні поля:

- name - ім'я VMI
- ip - ip-адреса
- zone - зона, в якій запущено VMI
- upload_url - посилання по якому можна відправити файл до VMI
- rpc - посилання для створення JsonRpcClient

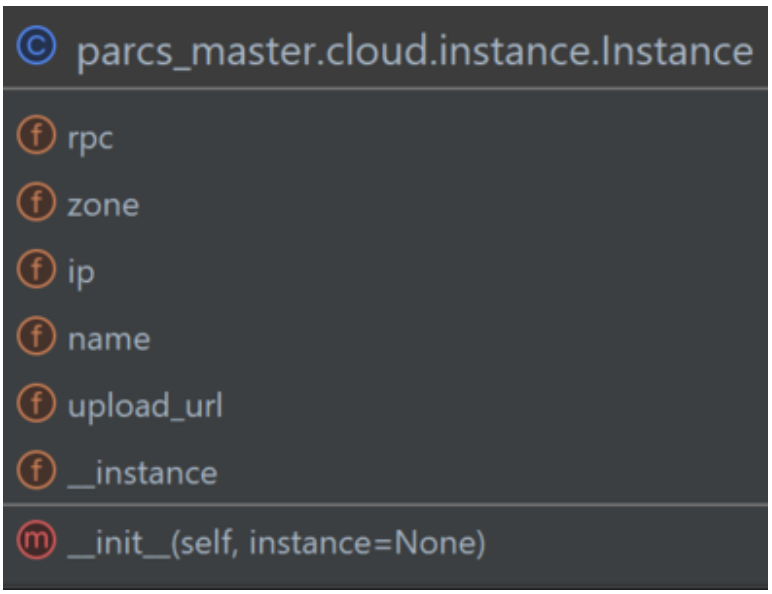

Рисунок 2 – UML-діаграма Instance

GoogleCloudController базується на основі абстрактного класу AbstractCloudController, який вимагає у класів-нащадків імплементації наступних методів, що дозволяє модульно додати імплементацію контролерів для інших сервісів:

- create_instance для створення об’єктів worker
- delete_instance для видалення об’єктів worker
- get_instances для отримання об’єктів worker

У GoogleCloudController ці методи імплементовані наступним чином:

a) create_instance - при виклику метод обирає ім’я нового VMI за порядковим номером, ініціалізує змінну config в якій ми прописуємо наступні параметри:
   1. ім’я VMI
   2. Характеристики VMI
   3. Мережеві налаштування

4. Скрипт, який буде виконано на старті

   Стартовий скрипт інсталює docker на VMI, затягує docker image parcs_node та запускає його з зовнішньою ip-адресою VMI. Після цього робиться виклик методу insert у об'єкту compute_client, який надсилає запит до GCP з параметрами зі змінної config, project та zone. Після цього робиться підрахунок існуючих об'єктів в поточному регіоні, якщо їх 4, то регіон змінюється на наступну зі списку доступних. Це зроблено через квоти GCP на кількість VMI в регіоні.

b) delete_instance отримує параметр instance_name та робить виклик методу delete у об'єкту compute_client, який надсилає запит до GCP з параметрами instance_name, project та zone.
c) get_instances повертає значення з instances

Також у цьому класі є приватний метод get_instances,який робить виклик методу aggregated_list у об'єкту compute_client, що повертає список всіх регіонів і, якщо є, список наявних VMI в регіоні. Отримавши цей список, йде цикл по кожному регіону. Якщо в регіоні є VMI та він має статус RUNNING або STARTING, то він попадає до результуючого списку. Після закінчення циклу, список повертається.

3.4.2. JobsController

Цей клас має тільки приватні поля:

- last_id - це поле потрібне для визначення id нової задачі
- jobs - лист об'єктів класу Job

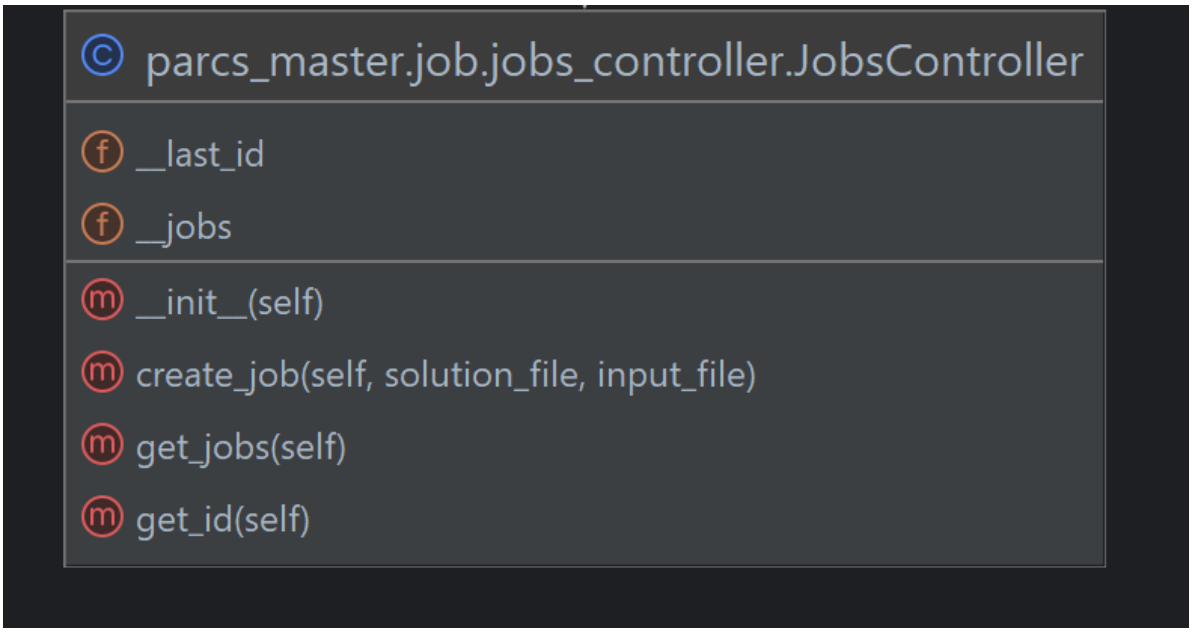

Рисунок 3 – UML-діаграма JobsController

Методи класу:

- create_job приймає на вхід два параметри: solution_file - шлях до файлу, де реалізовано алгоритм та input_file - шлях до файлу, де вхідні значення
- get_jobs повертає усі існуючі об’єкти класу Job.
- get_id повертає id наступної задачі.

В класі Job усі поля публічні, бо використовуються для відображення даних на сторінці:

- result - буде зберігати в собі помилку, якщо така відбудеться під час робити
- input_file - зберігає в собі шлях до файлу з вхідними значеннями
- job_id - id задачі
- solution_file - шлях до файлу з алгоритмом
- status - стан задачі

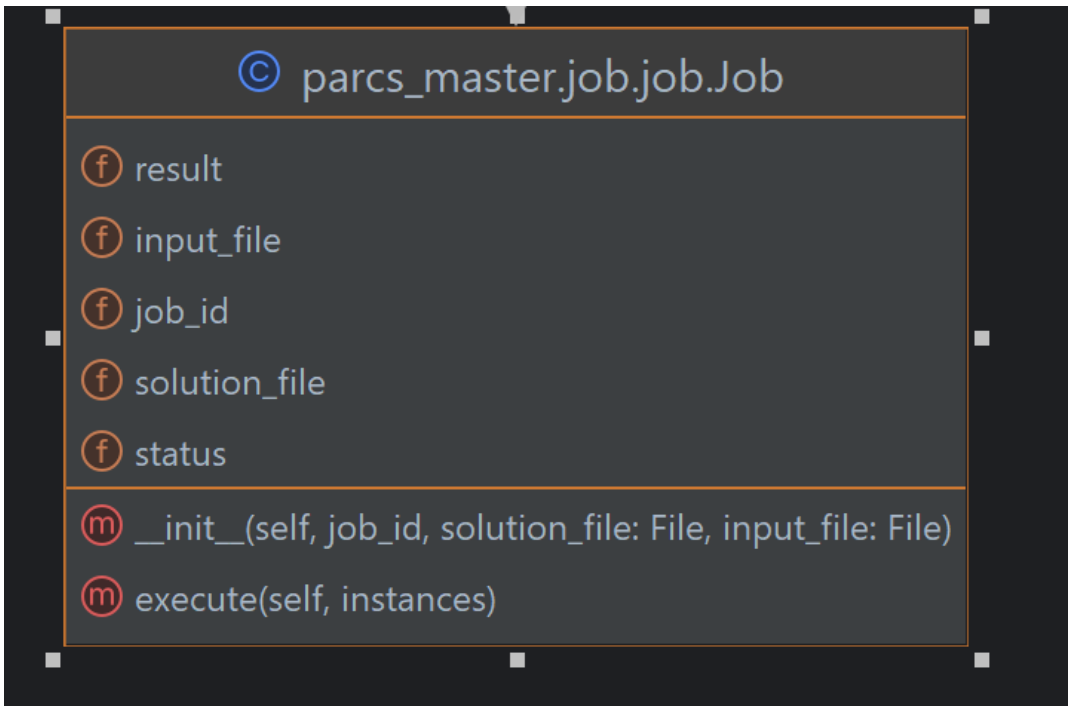

Рисунок 4 – UML-діаграма Job

Метод execute змінює статус задачі на Executing, після чого імпортує модуль з алгоритмом, який надіслав користувач, ініціалізує об’єкт класу Solver та викликає в нього метод solve. Якщо алгоритм виконався до кінця, то статус задачі змінюється на Success, якщо ні, то статус змінюється на Error та в змінну result потрапляє текст помилки, яка відбулась при виконанні алгоритму.

3.4.3. Веб-сервіс

При старті проекту ініціалізуємо об’єкт app класу Application з бібліотеки aiohttp, цей процес описується в методі create_app, в якому ми додаємо два контролери:

- GoogleCloudController, який відповідає за роботу з GCP
- JobsController, який відповідає за роботу з задачами

Також, окрім двох контролерів, ініціалізується шляхи, за якими може переходити користувач.

Перейшовши по зовнішній адресі master об’єкту, користувач потрапляє на індексну сторінку, де можна побачити список усіх об’єктів worker, де буде показано їх ім’я, ip, зону в якій запущений VMI та червона кнопка Remove, яка відповідає за видалення об’єкту. Також зверху є синя кнопка Create new instance, яка відповідає за створення нових VMIs.

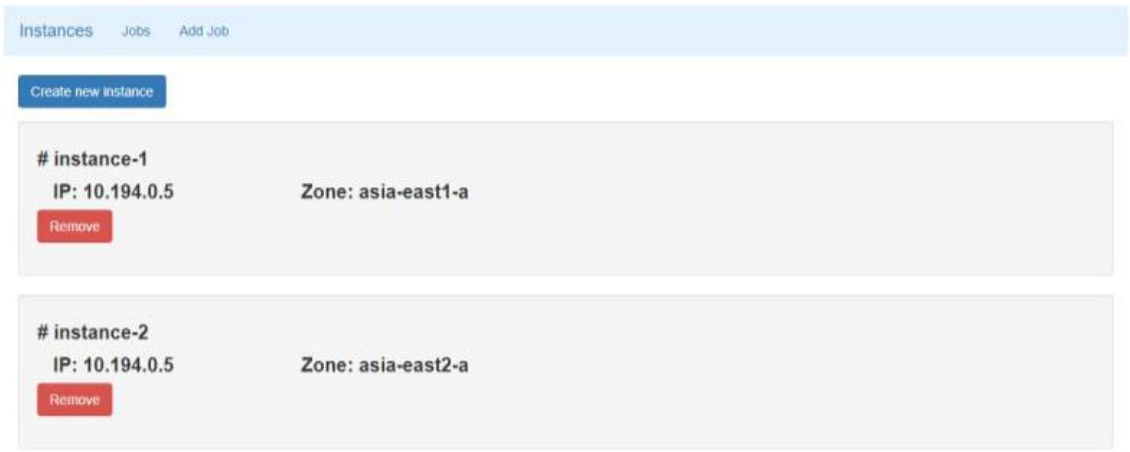

Рисунок 5 – Індексна сторінка зі списком існуючих VMIs

Після натискання кнопки видалення, веб-сервіс отримає HTTP-запит за посиланням /instances/delete/, де обробник remove_instance_view отримає з запиту параметр name, який відповідає ім’ю обраного об’єкту worker. Зробить виклик методу delete_instance з параметром name у класа-контролера GoogleCloudController. І зробить перенаправлення на сторінку зі списком об’єктів worker.

При натисканні кнопки створення нового об’єкту worker, веб-сервіс отримує HTTP-запит за посиланням /instances/create/, де обробник

add_instance_view зробить виклик методу create_instance у класа-контролера GoogleCloudController. І зробить перенаправлення на сторінку зі списком об’єктів worker.

Також, користувач може створювати об’єкт задачі. Для цього йому треба перейти на за посилання /add_job/. Тут буде форма з двома полями: одне для завантаження файла з алгоритмом, а друге для завантаження файлу з вхідними даними для алгоритму та зеленої кнопки Upload, яка відповідає за створення нової задачі

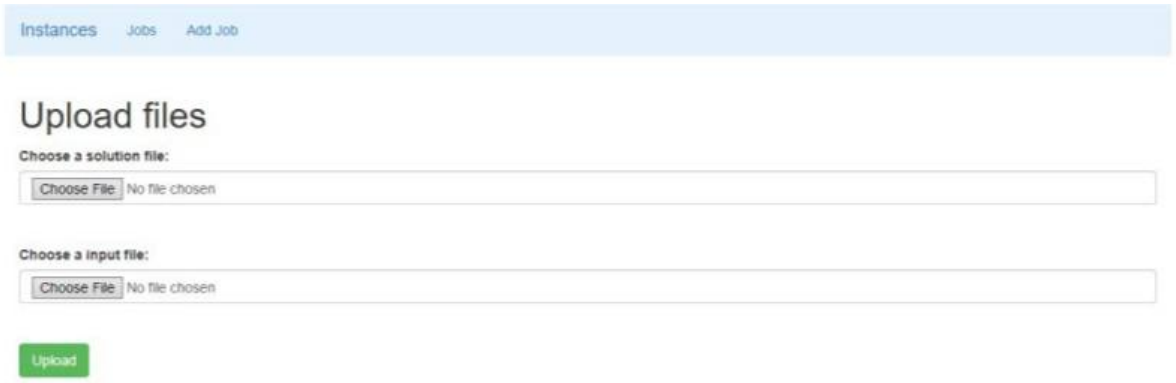

Рисунок 6 – Сторінка створення задачі

Після натискання кнопки Upload, веб-сервіс отримає HTTP-запит за посиланням /add_job/, але з методом не GET, як при звичайному переході на сторінку,а POST. Обробник запиту add_job_view отримає запит з файлами та збереже їх викликає метод create_job у класа-контролера JobsController, який поверне об’єкт класу Job. Після цього ці файли паралельно розішлються усім VMI і у об’єкту класу Job викликається метод execute, куди буде передано список об’єктів JsonRpcClient, які будуть підключатись до VMI по

їхнім зовнішнім ip-адресам. Після цього користувача буде направлено за посиланням /jobs/.

На сторінці, за посиланням /jobs/, користувач може побачити список всіх існуючих задач, їх стан та кнопку завантаження файла output, де буде відповідь на яку очікує користувач або лог помилки, якщо в реалізації алгоритму є проблеми.

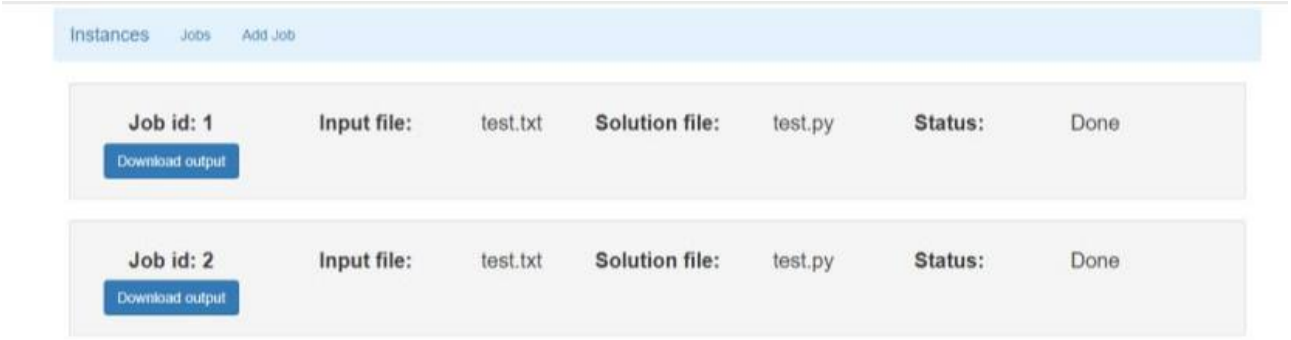

Рисунок 7 – Сторінка зі списком задач

При переході за посиланням /jobs/, веб-сервіс отримає HTTP-запит за посиланням. Обробник запиту get_jobs_view отримує запит та викликає метод get_jobs у класа-контролера JobsController.

3.5. Архітектура parcs_worker

Тут вже разом із REST веб-сервісом присутній RPC сервіс, який допомагає викликати методи напряму в об’єкта. RPC сервіс побудований на основі aiohttp_rpc.

aiohttp_rpc надає зручний інструментарій для виклику методів об’єкта VMI через RPC-ендпоінт /rpc/. При використанні aiohttp_rpc, реалізація

асинхронного RPC стає простішою завдяки зручному інтерфейсу та можливості додавати методи до списку доступних для виклику методів без необхідності вручну керувати окремими потоками, як це було з Pyro4.

У попередній реалізації, для кожного об’єкта VMI створювався окремий потік через синхронність Pyro. Але завдяки aiohttp_rpc, цей процес спрощується, оскільки можна просто додати необхідний метод до списку доступних для виклику, методів, й зосередитись на інших аспектах розробки, доки бібліотека самостійно керує всім необхідним. Це дозволяє зменшити складність реалізації асинхронного RPC та покращити продуктивність системи, забезпечуючи зручний спосіб виклику методів об’єкта VMI через RPC-ендпоінт.

Окрім цього є контроллер JobsController, який керує задачами, що надходять до VMI.

3.5.1. JobsController

У JobsContoller є лише одне, приватне, поле - jobs. Це список усіх об’єктів класу Job.

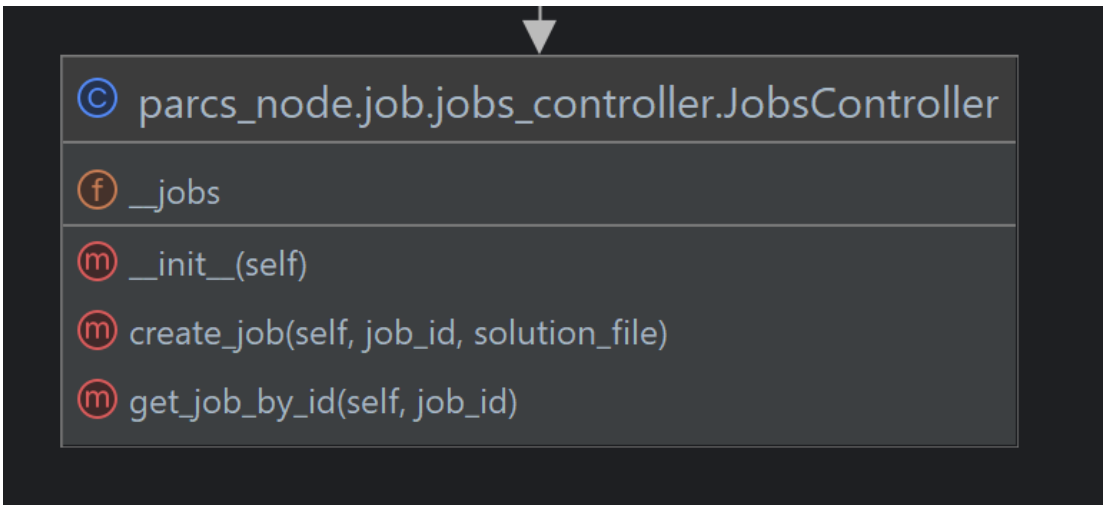

Рисунок 8 – UML-діаграма JobsController

Метод create_job створює об’єкт задачі. Для цього цей метод використовує ідентифікатор задачі та шлях до файлу з алгоритмом. Метод ініціалізує об’єкт задачі, додає його в список jobs та повертає об’єкт створеної задачі.

Метод get_job_by_id повертає задачу по id.

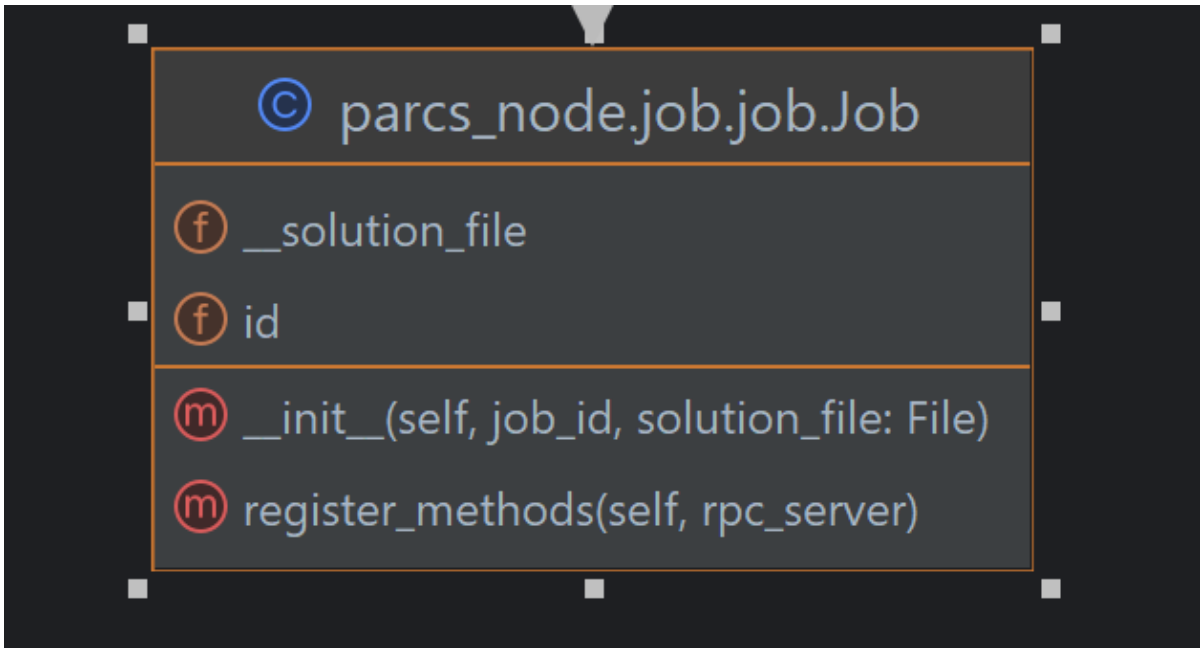

Рисунок 9 – UML-діаграма Job

Клас Job має наступні поля:

- solution_file - шлях до файлу з алгоритмом
- id - ідентифікатор задачі

Метод register_methods імпортує модуль з файлу, який надіслав користувач. На випадок, якщо модуль не буде знайдений, через конфлікти менеджера модулів в Python, присутній обробник помилки та прямий імпорт

файлу. Далі ініціалізується об’єкт класу Solver, створюється список з посилань на функції з цього об’єкту та цей список додається в об’єкт rpc_server, який дозволяє викликати ці методи з іншою машини напряму, ніби у об’єкта є цей метод.

3.5.2. Веб-сервіс

REST-ендпоінт /api/upload/ очікує сам файл та поле job_id та використовується для отримання файла з алгоритмом, створення задачі під нього та реєстрації методів з файлу з алгоритмом.

RPC-ендпоінт /rpc/ використовується для того, щоб підключитись до об’єкту VMI та викликати методи, ніби це локальний об’єкт

3.6. Налаштування та запуск

Для успішного налаштування системи ПАРКС для платформи Google Cloud потрібно виконати кілька кроків:

- Спочатку потрібно створити обліковий запис на платформі Google Cloud, якщо він відсутній.
- Потрібно створити проект
- Після створення облікового запису треба увімкнути білінг для проекту, в якому планується використовувати систему ПАРКС.
- Потрібно створити сервісний аккаунт у панелі керування Google Cloud. Для цього треба перейти у розділ «IAM і адміністрування» або «IAM» і перейти в розділ «Service Account», де аккаунту треба надати права «Compute Admin»

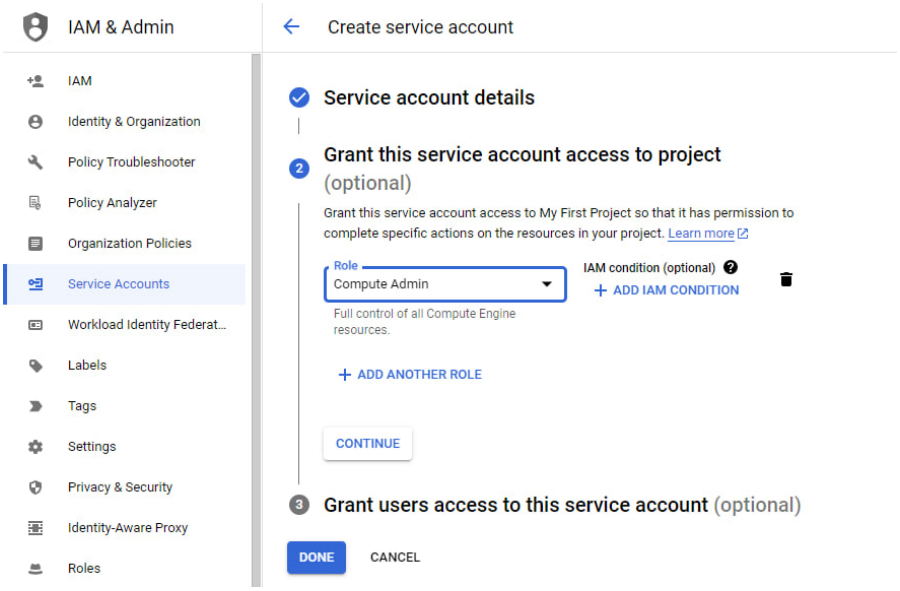

Рисунок 10 – Створення сервісного аккаунту

- Після створення сервісного аккаунту, користувач потрапляє на сторінку з усіма сервісними акаунтами. Потрібно перейти на сторінку створенного сервісного акаунту у розділ «Keys». Там натиснути «Add key», «Create new key» та згенерувати JSON-файл з ключем доступу. Цей файл містить інформацію, необхідну для автентифікації системи ПАРКС під час взаємодії з Google Cloud.

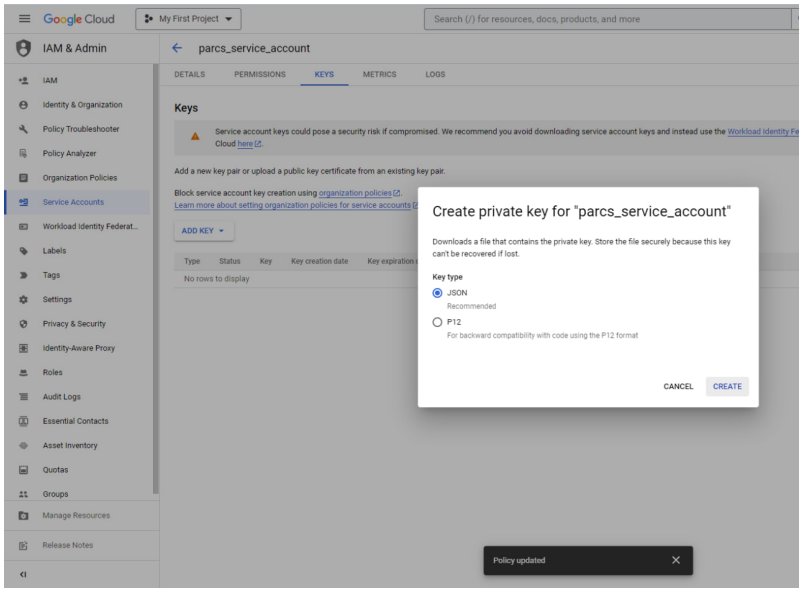

Рисунок 11 – Генерація json-файлу для автентифікації

- Для дозволу запитів до системи ПАРКС треба налаштувати фаєрвол Google Cloud, щоб не було проблем з доступом. Для цього потрібно зайти в консоль та виконати наступну команду: gcloud compute firewall-rules create allow-all –direction=INGRESS –priority=1000 –network=default –action=ALLOW –rules=all –source-ranges=0.0.0.0/0. Це дозволяє отримувати запиту від будь-якого джерела

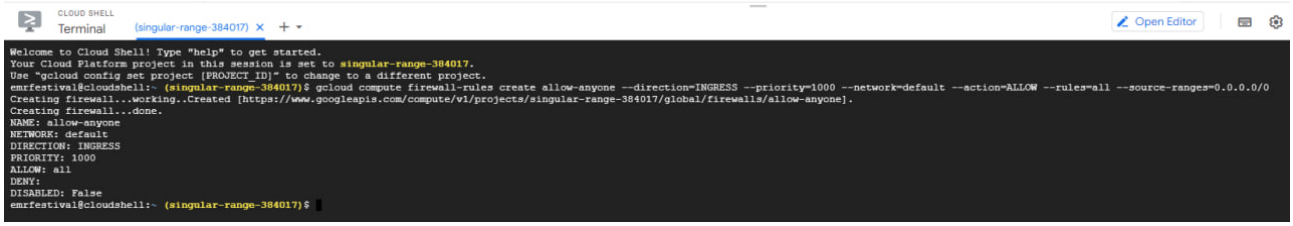

Рисунок 12 – Створення фаєрволу

- Для розгортання системи ПАРКС на Google Cloud потрібно створити об’єкт мастер. Для цього у розділі VM Instances треба натиснути на кнопку «Create VM Instance» та обрати стандартну конфігурацію VMI.

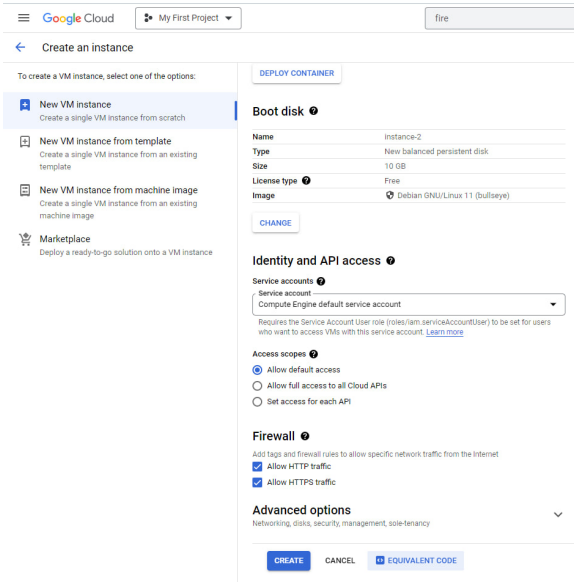

Рисунок 13 – Створення об’єкту master

- Після створення VMI мастер, потрібно підключитись до нього по ssh натиснувши відповідну кнопку в табличці з VMI. Відкриється нове вікно з консолью. Тут вікну можна побачити кнопку «Upload File». Потрібно натиснути на неї та завантажити JSON-файл з ключем доступу до об'єкта мастер.

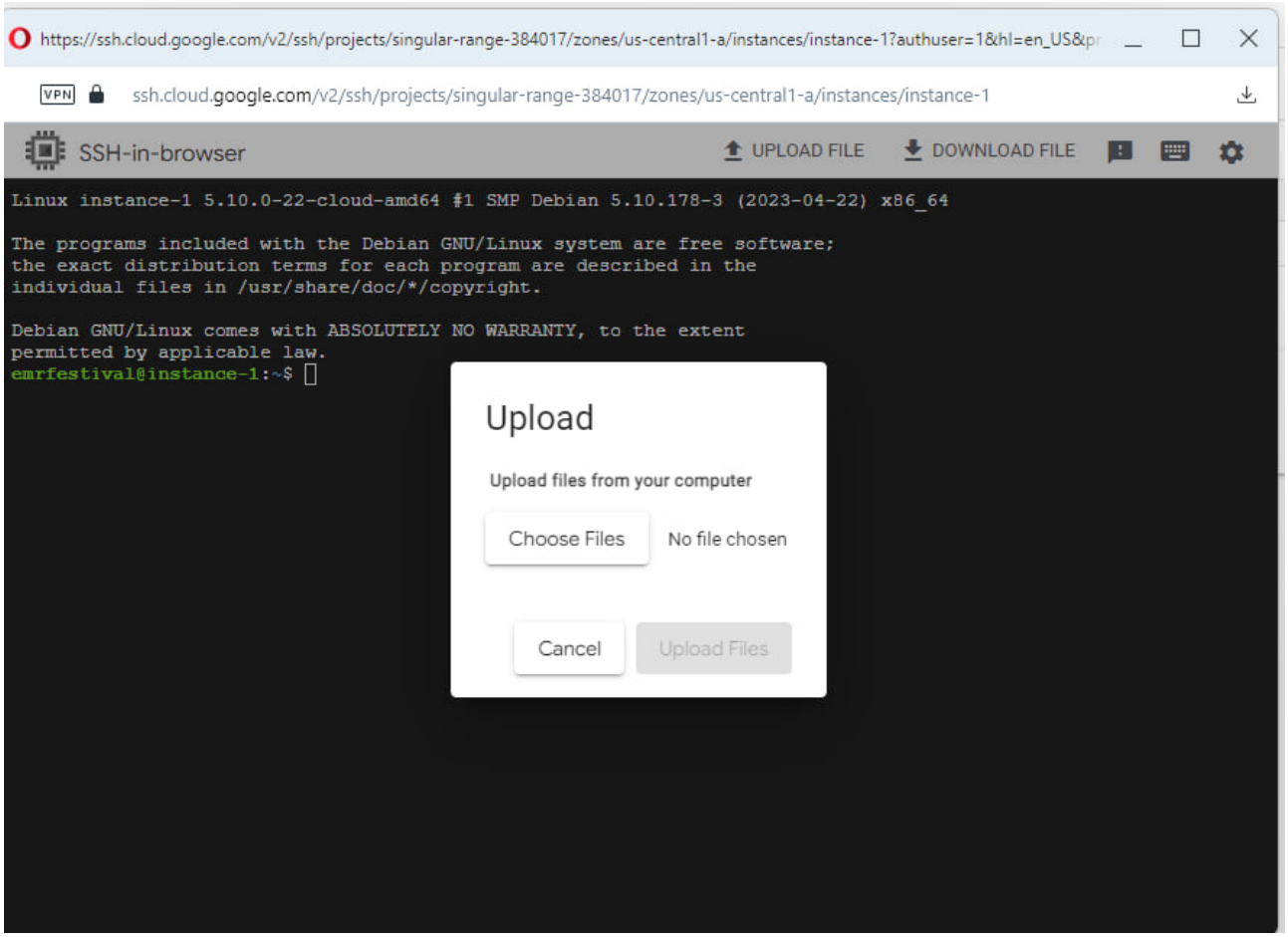

Рисунок 14 – Створення фаєрволу

- Останнім кроком є завантаження та запуск Docker-образу dtelt/parcs_master. Для цього необхідно встановити Docker, завантажити docker-image та запустити контейнер об'єкта мастер з використанням ключа -cc, в який треба передати шлях до файлу з ключем доступу. Це дозволить ініціалізувати систему ПАРКС на Google Cloud.

Після цих маніпуляцій можна переходити за зовнішньою ip-адресою VMI мастера та використовувати ПАРКС.

ВИСНОВКИ

У цьому дослідженні була реалізована нова версія системи ПАРКС з використанням мови програмування Python та сучасних технологій, таких як aiohttp, aiohttp_rpc та google-cloud-compute. За допомогою цих інструментів було усунуто недоліки попередньої реалізації.

Реалізація на базі aiohttp дозволила обробляти запити від користувачів асинхронно. Бібліотека aiohttp забезпечує швидку та легку реалізацію веб-сервера, що дозволяє обробляти багато запитів одночасно та забезпечує гнучкість у роботі з HTTP-протоколом.

Використання aiohttp_rpc дозволило забезпечити взаємодію між компонентами системи за допомогою RPC-викликів. Це зробило комунікацію між різними модулями та компонентами більш простою та зрозумілою.

Є інтеграція системи з Google Cloud Platform. Використання бібліотеки google-cloud-compute дало можливість створювати та видаляти віртуальні машини на основі вимог системи. Це відкрило широкі можливості для масштабування системи та забезпечило гнучкість у роботі з хмарами.

Одним з важливих досягнень нової реалізації було усунення недоліків попередньої версії системи:

- Розбиття файлів на логічні модулі по папкам дозволило зробити код більш організованим та зручним для розробки.
- Відокремлення публічних та приватних методів та полів у класах сприяє кращій модульності та забезпечує захист приватних деталей реалізації.

- Використання сучасних бібліотек дозволило зробити систему швидкою та надійною.

Існує потенціал для удосконалення системи ПАРКС, впровадження нових функціональностей та розширення її можливостей:

- Підтримка нових cloud-платформ
- Вдосконалення передачі алгоритмічного модуля від користувача до системи

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дерев'янченко О.В., “Моделювання паралельних програм за допомогою системи ПАРКС-JAVA // наукові записки Наукма, комп'ютерні науки,” 2005.
2. Анісімов А.В. Борейша Ю.Е. Кулябко П.П., “Технология параллельного программирования «ПАРУС» // автоматика и телемеханика,” 1990.
3. Анісімов А.В. Кулябко П.П, “Программирование параллельных процессов в управляющих пространствах. // кибернетика,” 1984.
4. Анісімов А.В. Дерев'янченко О.В., “Система ПАРКС-JAVA як засіб вирішення паралельних алгоритмів на комп'ютерній мережі // материалы четвертой международной научно-практической конференции Укрпрог'2004. – проблемы программирования.,” 2004.
5. Годованюк М., “Parcs-python.” [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/Hummer12007/parcs-python/tree/master
6. “Rest.” [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST
7. “Remote procedure call.” [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Remote_procedure_call
8. “Flask.” [Електронний ресурс]. Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.3.x/
9. “Pyro - python remote objects.” [Електронний ресурс]. Режим доступу до ресурсу: https://pyro4.readthedocs.io/en/stable/index.html
10. “Aiohttp.” [Електронний ресурс]. Режим доступу до ресурсу: https://docs.aiohttp.org/en/stable/
11. “Access modifiers.” [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Access_modifiers

12. “Information hiding.” [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Information_hiding/