Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ В СЕРЕДОВИЩІ PYTHON

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи
к.н., доц. Гороховський С.С.
_____ (Підпис)
“___” _____ 2023 року
Виконав студент
ІПЗ-3 Білокінь Є. Т.
“___” _____ 2023 року

Київ 2023

ЗМІСТ

АНОТАЦІЯ

Курсова робота присвячена проблемам та інструментам розподілених обчислень на Python. Для вирішення задачі розподілених обчислень було досліджено та оглянуто такі інструменти: Celery, Python-RQ та Pyro.

Розроблено програмний застосунок для демонстрації, порівняння та оцінки продуктивності розподілених обчислень на Python. Реалізована взаємодія між фреймворком Django та Redis Broker Transports. Також застосовано Docker для налаштування потрібних сервісів.

Досліджені інструменти розподілених обчислень та їх інтеграція у веб-застосунок з налаштуванням та коректною роботою серверу.

Текстова частина курсової містить опис дослідження проблем розподілених обчислень та методи їх реалізації. Текстова частина курсової також описує необхідні теоретичні відомості про використані інструменти та технології.

Ключові слова: Python, розподілені обчислення, Celery, Python-RQ, Pyro, Django.

ВСТУП

Розподілені обчислення є актуальним та важливим напрямом сучасного програмування, що дозволяє розподіляти обчислювальні завдання між різними комп'ютерами. Цей підхід забезпечує ефективне використання ресурсів та прискорює вирішення великих обчислювальних завдань.

Одним із популярних інструментів для реалізації розподілених обчислень є мова програмування Python. Python володіє простим та зрозумілим синтаксисом, широким спектром бібліотек та фреймворків, що дозволяють легко створювати та управляти розподіленими системами.

Актуальність даної теми полягає в необхідності швидкої та ефективної обробки великих обсягів даних, що стає все більш поширеним завданням сучасного інформаційного світу. За допомогою розподілених обчислень на Python можна досягти більшої продуктивності та швидкості виконання завдань, а також забезпечити масштабованість системи для розвитку у майбутньому. Також застосування цих технологій може стати у нагоді для швидкої та зручної взаємодії з користувачем, приховуючи виконання тривалих процесів. Інструменти розподілених обчислень можна застосовувати для завдань машинного навчання, створення швидких інтерфейсів для керування довготривалими завданнями та у інших не менш важливих сферах.

Метою дослідження є вивчення основних концепцій та практичного застосування розподілених обчислень на мові програмування Python. Буде розглянуто різні аспекти розподілених систем, включаючи архітектуру, комунікацію, синхронізацію та обробку помилок. Також будуть досліджені різні бібліотеки та фреймворки, доступні для розробки розподілених програм на Python.

Дана робота складається з трьох розділів.

В першому розділі розглянуто підходи для вирішення задач розподілених обчислень у середовищі Python. Визначення проблематики та застосування цих задач. Проаналізовано використання розподілених обчислень на прикладах реальних технологій.

У другому розділі досліджені методи вирішення поставленої задачі, порівняння та опис існуючих технологій для виконання завдання.

Третій розділ розгорнуто описує процес створення демонстраційного веб-застосунку та його тестування. Здійснюються заміри часу виконання та оцінка продуктивності різних програмних рішень для розподілених обчислень.

ПОСТАНОВКА ЗАДАЧІ

1. Виконати аналіз задач розподілених обчислень та розглянути приклади застосування систем, заснованих на принципах таких обчислень.
2. Розповісти про особливості та переваги використання Python для розподілених обчислень.
3. Зробити огляд відомих методів розробки розподілених обчислень у середовищі Python, зокрема:
   — Celery;
   — Python-RQ;
   — Pyro.
4. Розробити демонстраційний веб-застосунок з відповідними налаштуваннями для імплементації роботи розглянутих методів.
5. Виконати порівняльний аналіз використаних методів. Зробити висновки щодо продуктивності, проблем інтеграції та роботи з оглянутими технологіями загалом.

РОЗДІЛ 1 Розподілені обчислення

1.1 Загальні відомості та визначення

Для початку потрібно узагальнити визначення розподілених обчислень задля покращення розуміння розробки та використання цих програмних рішень. Згідно зі словами автора посібника «Distributed Computing with Python», розподіленні обчислення – це одночасне використання більш ніж одного комп'ютера для вирішення проблеми [1, с. 4]. Також він зазначає, що зазвичай визначення додатково обмежене правилом: розподілена система має відображатись користувачу як одна машина, приховуючи розподілену природу програми. Розподіленою системою є та, у основу якої закладено принципи РО. Надалі ми будемо відштовхуватись та розширяти це бачення досліджуваних систем.

Використання РО між кількома комп'ютерами є очевидною стратегією при використанні систем, які можуть спілкуватися одна з одною через мережу (локальну чи іншу). Існує багато задач для розробки систем або застосунків, що вимагають використання РО. Зазвичай, причиною є здатність вирішити доволі об'ємну проблему, яку жоден окремий комп'ютер не міг би вирішити взагалі, або, принаймні, не за раціональний проміжок часу. В окремих випадках сама направленість програми, що розробляється, вимагає розподіленої системи. Прикладом можуть слугувати програми обміну повідомленнями та застосунки для відеоконференцій. Для цих прикладів програмного забезпечення продуктивність не є основним фактором. Просто проблема, яку вирішує додаток – розподілена [1, с. 5].

На рисунку 1.1 зображено найпоширенішу реалізацію РС у розробці застосунку. Це така система, де кілька користувачів підключаються до веб-сайту через мережу. Одночасно з цим, застосунок

спілкується з системами (наприклад, сервер бази даних), що працюють на різних машинах у її локальній мережі:

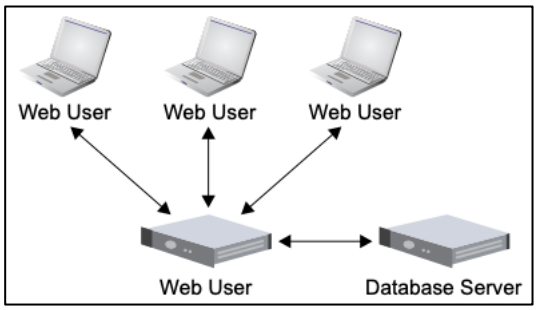

Рисунок 1.1 - приклад розподіленої системи

1.2 Аналіз сучасного стану питання

Дослідивши етапи розвитку технологічного процесу та загальної направленості сучасних інформаційних задач на опрацювання великих обсягів інформації [2], а також еволюцію парадигми РС[2, с. 7] (рисунок 1.2) можна зробити висновок про велике значення та важливість розвитку розподілених обчислень. На сьогоднішній день майже всі актуальні та перспективні галузі інформаційних технологій будуються на основі РС.

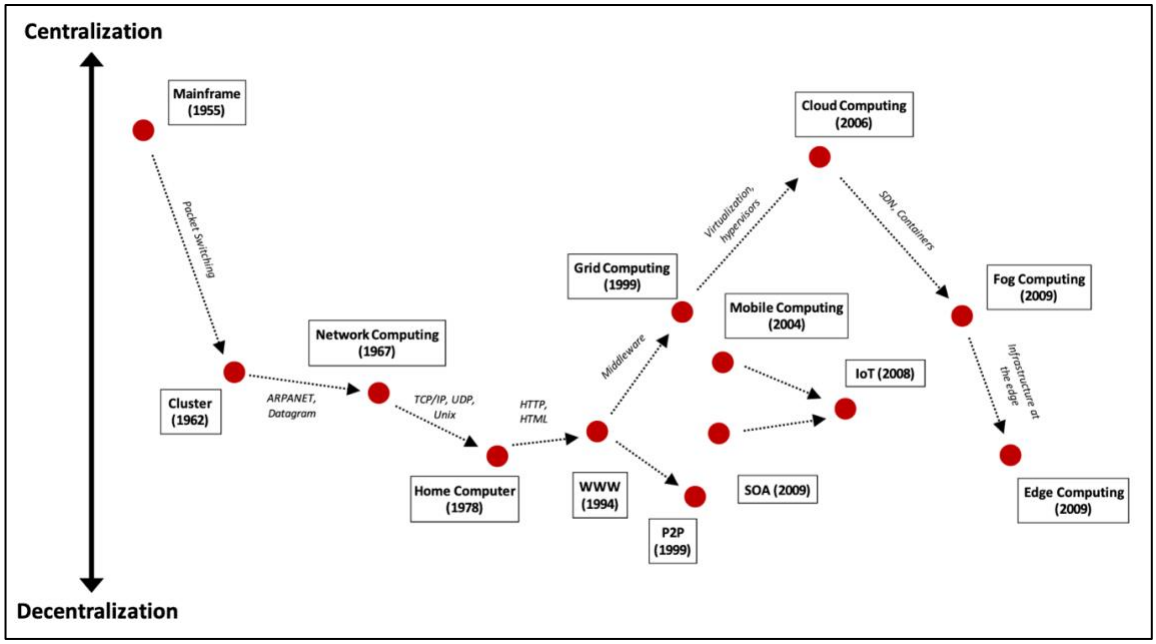

Рисунок 1.2 - Зображення еволюції парадигми розподіленої системи

Загалом можна визначити деякі сфери для застосування РО, щоб дослідити мету побудови такої системи та можливі труднощі її застосування:

— ***Обробка великих обсягів даних:*** З великим зростанням обсягу даних, розподілені обчислення стають необхідними для ефективної обробки та аналізу цих даних. Вони дозволяють розподіляти завдання обробки даних між багатьма вузлами, що прискорює процес та забезпечує масштабованість.

— ***Машинне навчання та інтелектуальний аналіз даних:*** Розподілені обчислення відіграють ключову роль у розвитку машинного навчання та інтелектуального аналізу даних. Вони дозволяють тренувати складні моделі машинного навчання на великих наборах даних та проводити інтенсивний аналіз даних для виявлення закономірностей та отримання цінної інформації.

— ***Великомасштабні системи та хмарні обчислення:*** РО використовуються для побудови великомасштабних систем та хмарних обчислень. Це дозволяє компаніям та організаціям отримувати доступ до обчислювальних ресурсів за запитом, масштабувати свої додатки та послуги, а також забезпечувати високу доступність та надійність.

— ***Інтернет речей (IoT):*** РО грають важливу роль у розвитку Інтернету речей. Зі зростанням кількості підключених пристроїв та сенсорів, необхідність ефективної обробки та аналізу даних стає на перший план. РС дозволяють обробляти та аналізувати дані з різних джерел, забезпечуючи обробку в режимі реального часу та взаємодію між пристроями.

Оглянувши основні напрямки розвитку РО в межах зазначених сфер, ми можемо продовжити дослідження, звертаючись до прикладів

реалізації РС, що допоможе нам правильно обрати технології розробки та уникнути помилок на рівні логіки та структури такої системи.

Також варто зазначити, що будь-яка розроблена РС, яку ми використаємо, швидше за все буде працювати на сучасних комп'ютерах, оскільки зазвичай вони є багатоядерними, а часто навіть і багатопроцесорними. Ця парадигма[1, с. 12] дозволяє нам використовувати РО у нашому коді, що будуть доступні для опрацювання засобами сучасних користувачів.

1.3 Огляд підходів до вирішення задач РО

Проаналізувавши розвиток РО та приклади розробки РС задля вирішення конкретних задач[3, с. 3-16], можна зазначити, що використання РС підвищує ефективність програмного забезпечення шляхом делегування та координації процесів між кількома пристроями. Також вона підвищує надійність програм, використовуючи резервні системи та пристрої.

Оскільки РС вимагає налагоджування звя'зку між пристроями та розподілом пам'яті, існує декілька типів РС. Тип РС стосується архітектури та стилю зв'язку, які використовує мережа для виконання завдань і обробки інформації, що зберігається на кількох серверах і пристроях.

Розглянемо типи архітектур РС[3, с. 66] та їх приклади, щоб дослідити залежність типу архітектури з виконуваними задачами:

— ***Client-server:*** У підході client-server система складається з клієнтів, які надсилають запити до централізованого сервера, який обробляє ці запити та надає відповіді клієнтам. Цей підхід забезпечує централізоване керування та контроль над обчисленнями, дозволяючи

ефективно виконувати завдання та забезпечувати безпеку даних. Приклади: файловий обмін, криптовалюта.

— ***Peer-to-peer (P2P):*** У підході peer-to-peer вузли мережі співпрацюють на рівних позиціях і мають можливість взаємодіяти один з одним без централізованого сервера. Кожен пір може функціонувати як клієнт і сервер одночасно. Цей підхід дозволяє розподіляти завдання та ресурси між учасниками мережі, що забезпечує високу масштабованість та стійкість до помилок. Приклади: веб-додатки, електронна пошта.

— ***Three-tier:*** У архітектурі three-tier система розділяється на три рівні: клієнтський рівень, рівень бізнес-логіки та рівень доступу до даних. Клієнти взаємодіють з рівнем бізнес-логіки, а рівень бізнес-логіки отримує доступ до даних через рівень доступу до даних. Цей підхід дозволяє розділити функціональність, полегшує розробку та підтримку системи, сприяє модульності та перевикористанню компонентів. Приклади: веб-додатки, електронна комерція.

— ***N-tier:*** Архітектура n-tier подібна до three-tier, але може містити більше ніж три рівні. Цей підхід дозволяє розгортати систему на більшу кількість рівнів, що дозволяє краще організувати функціональність та підвищує масштабованість та гнучкість системи. Наприклад, в архітектурі n-tier можуть бути рівні для обробки бізнес-логіки, кешування, інтеграції з іншими системами та інші. Приклади: хмарні додатки.

Ці підходи до вирішення задач розподілених обчислень мають свої переваги та використовуються в різних сценаріях. Вибір підходу залежить від вимог проєкту, розмірів системи та інших факторів. Успішне використання розподілених обчислень вимагає детального розуміння цих підходів та їхніх особливостей для вибору оптимального рішення.

1.4 РО у середовищі Python. Переваги та особливості

Мова програмування Python є популярним вибором для реалізації розподілених систем. В цьому розділі ми розглянемо переваги та особливості розподілених обчислень у середовищі Python.

Переваги включають у себе простоту та зручність використання Python для побудови РС. Широкий вибір фреймворків та бібліотек є критично важливим, оскільки це дозволяє обирати потрібні інструменти для визначених завдань та використовувати їх переваги.

Не менш важливим є підтримка асинхронного програмування. Python надає потужні механізми для асинхронного програмування для роботи з асинхронними задачами. Це дозволяє ефективно використовувати розподілені ресурси та забезпечує високу продуктивність системи при обробці багатьох запитів одночасно [4, с. 39].

Також великою перевагою слугує багатий інструментарій для обробки даних. Це дозволяє ефективно використовувати розподілені обчислення для обробки великих обсягів даних та виконання складних аналітичних завдань, про що ми говорили у попередніх пунктах.

Особливістю побудови РС у середовищі Python є підтримка різних форматів збереження та обробки даних. Це дозволяє зручно та ефективно обробляти дані в розподілених системах. Однак, важливо враховувати, що передача великих обсягів даних між вузлами може бути витратною операцією, тому оптимізація та розподіл обробки даних може бути важливими аспектами.

1.5 Проблематика

Опираючись на дослідження розробки розподілених систем, варто дослідити проблеми задля ефективного та швидкого процесу створення застосунку. Хід побудови РС – складний та багатоетапний, тому варто знати, яким аспектам варто приділити особливу увагу.

Далі ми розглянемо проблематику створення РС при розробці застосунку [5] та можливі шляхи вирішення. Потрібно виділити наступні проблеми побудови РС:

— ***Надійність***: РС повинні бути стійкими до помилок окремих вузлів або мережевих з'єднань. Додатково, розподілені системи повинні забезпечувати можливість відновлення даних в разі втрати.
Шляхи вирішення: розробити механізми для виявлення помилок та автоматичного відновлення системи, щоб забезпечити безперебійну роботу.

— ***Масштабованість***: РС повинні бути здатними ефективно працювати зі зростанням обсягу даних, навантаження та кількості вузлів.
Шляхи вирішення: розробити механізми горизонтального та вертикального масштабування, щоб забезпечити продуктивність та розширюваність системи.

— ***Конфігурація та керування:*** управління РС потребує налагодження та керування конфігурацією багатьох вузлів та компонентів. Це включає управління розгортанням нових вузлів, налаштування параметрів системи.
Шляхи вирішення: використовувати інструменти для композиції застосунку та належним чином програмно передбачити можливі сценарії.

— ***Тестування та налагодження:*** розробка та налагодження РС можуть бути складними завданнями.

Шляхи вирішення: створення тестових середовищ та сценаріїв, які відображають реальні умови використання системи.

— ***Теоретична підготованість:*** створення РС потребує гарного знання теорії задля коректного вибору архітектури та інструментів.

1.6 Висновок

У цьому розділі проаналізовано та досліджено розвиток РО, їх значимість у сучасних технологіях та визначенні основні сфери застосування. Також було досліджено шляхи побудови РС задля коректного вибору архітектури для розробки додатків, що дозволяє правильно підійти до процесу створення застосунку.

У наступних розділах ці відомості допоможуть у розробці та виборі технологій.

РОЗДІЛ 2 Огляд наявних технологій для РО на Python

2.1 Celery

2.1.1 Загальні відомості

Celery є однією з найпопулярніших технологій для розподілених обчислень у середовищі Python. Вона надає фреймворк для асинхронного виконання завдань та управління розподіленими задачами у розподіленому середовищі. Основною концепцією Celery є розподілені черги повідомлень, де можна визначити задачі, які будуть виконуватись асинхронно на вузлах мережі [6].

Черги завдань використовуються як механізм для розподіленої роботи між потоками або машинами. Вхідні дані черги завдань — це одиниця роботи, яка називається завданням. Виділені робочі процеси постійно відстежують черги завдань для виконання нової роботи. Celery спілкується за допомогою повідомлень, зазвичай використовуючи брокера для посередництва між клієнтами та працівниками. Щоб ініціювати завдання, клієнт додає повідомлення до черги, а потім брокер доставляє це повідомлення обробнику (worker). Система Celery може складатися з кількох обробників і брокерів, що забезпечує високу доступність і горизонтальне масштабування. Celery написаний на Python, але протокол можна реалізувати будь-якою мовою.

Взаємодія мов також може бути досягнута, відкриваючи кінцеву точку HTTP та маючи завдання, яке її запитує (веб-хуки).

2.1.2 Переваги та недоліки

Перевагами Celery для реалізації розподілених обчислень у середовищі Python є наступні показники [7]:

— ***Масштабованість***: Celery дозволяє легко масштабувати ваші розподілені обчислення шляхом додавання нових обробників (worker) або вузлів. Це дозволяє збільшувати потужність обчислень та обробляти більше завдань при зростанні обсягу даних або навантаження на систему.

— ***Асинхронність***: Завдяки асинхронному виконанню, Celery дозволяє вам виконувати завдання паралельно та ефективно використовувати ресурси. Ви можете надсилати задачі до черги та продовжувати роботу без очікування їх виконання, що підвищує продуктивність та зменшує час очікування.

— ***Гнучкість***: Celery дозволяє вам гнучко визначати та налаштовувати завдання. Ви можете передавати параметри до задач, керувати пріоритетами та часом виконання, налаштовувати повторну спробу виконання у випадку невдачі. Це дозволяє вам точно налаштовувати роботу системи під ваші потреби.

— ***Легка інтеграція***: Celery можна легко інтегрувати з існуючими проектами на Python. Можна огорнути функції декораторами @task та використовувати їх для асинхронного виконання. Це дозволяє використовувати потужність розподілених обчислень у проєкті без необхідності переписування коду.

— ***Тестування та обробка помилок***: Celery дозволяє використовувати інструменти, такі як Celery Flower або Celery Events, для візуалізації та моніторингу стану задач, черг та робітників. Конфігуруйте журнал виконання, щоб відстежувати помилки та події в системі.

Недоліками можуть слугувати наступні чинники використання Celery:

— ***Складність налаштування:*** Налаштування Celery може бути складним завданням, особливо для початківців. Потрібно правильно

налаштувати брокер повідомлень та обробників (workers), встановити параметри та визначити правильні налаштування для досягнення оптимальної продуктивності та надійності. Це може вимагати додаткових знань та часу на вивчення документації та налагодження.

— ***Залежність від зовнішніх сервісів***: Для коректної роботи Celery потрібно мати належно налаштований та доступний брокер повідомлень, такий як RabbitMQ або Redis. Це означає, що ви залежите від стороннього сервісу, і його недоступність або проблеми зі з'єднанням можуть вплинути на роботу вашої системи розподілених обчислень.

— ***Витрати ресурсів***: Розподілені обчислення з використанням Celery можуть вимагати додаткових ресурсів, таких як пам'ять, процесорний час та мережевий пропуск. При збільшенні кількості обробників (workers) і обсягу завдань, може знадобитися масштабування інфраструктури та збільшення ресурсів, що може вплинути на витрати та ефективність системи.

2.1.3 Використання

Одним з ключових аспектів успішного використання Celery у середовищі Python є правильна конфігурація. Під час налаштування Celery, визначаються параметри, які описують поведінку системи РО.

Встановлення виконується за допомогою менеджера пакетів Python, наприклад, pip. Інтеграція Celery у Python-програму здійснюється у включенні всіх необхідних залежностей та імпортуючи необхідні модулі.

Для коректної роботи Celery потрібно налаштувати брокер повідомлень. Celery використовує брокер повідомлень для обміну повідомленнями між задачами та робітниками. Треба обрати та

використати підходящий брокер повідомлень (RabbitMQ або Redis), а також встановити необхідні параметри підключення до брокера.

Для подальшої роботи з Celery потрібно визначити та налаштувати задачі, які будуть виконуватися розподілено. Необхідно включити до налаштувань час очікування, пріоритет та повторюваність.

Останнім важливим елементом використання Celery є налаштування обробників задач (worker), що будуть виконувати визначенні розподілені функції.

2.2 Python-RQ

2.2.1 Загальні відомості

Python-RQ (Redis Queue) є іншою технологією для розподілених обчислень у середовищі Python. Вона базується на Redis, який використовується як брокер повідомлень для обміну задачами між робітниками. Python-RQ пропонує простий та прямолінійний підхід до розподілених обчислень, зосереджуючись на простоті використання та інтеграції з Python [8].

Python-RQ є одним з варіантів для розподілених обчислень у середовищі Python. Він надає простоту використання та гнучкість, хоча має певні обмеження щодо масштабування. Вибір Python-RQ залежить від конкретних потреб вашого проєкту та обмежень, з якими ви стикаєтеся.

2.2.2 Переваги та недоліки

Python-RQ був натхненний хорошими частинами у Celery та був створений як легка альтернатива для масового обслуговування з низьким бар’єром для входу [8].

Переваги цього рішення (без вже перерахованих у попередньому підрозділі) є наступними:

— ***Простота*** використання: Python-RQ має простий та зрозумілий API, що дозволяє швидко почати використовувати його в проекті.

— ***Легка інтеграція***: Python-RQ побудований на основі Redis, який є потужним та широко використовуваним інструментом. Це дозволяє легко інтегрувати Python-RQ усередину існуючого стеку технологій.

— ***Гнучкість***: Python-RQ надає можливість налаштування пріоритетів задач, розподілу завдань між обробниками (workers), встановлення часу виходу та інших параметрів.

Недоліками при спрощенні технології слугують наступні чинники:

— ***Обмежені можливості масштабування:*** Python-RQ не надає вбудованих механізмів для масштабування та балансування навантаження між обробниками (workers). Це може бути обмеженням для великих та високонавантажених систем.

— ***Залежність від Redis***: На відміну від Celery, Python-RQ використовує лише Redis як брокер повідомлень, що означає, що він потребує наявності та правильної конфігурації Redis-сервера. Це може бути не зручно або недоцільно в деяких проєктах.

2.2.3 Використання

Встановлення Python-RQ відбувається за допомогою менеджера пакетів Python, такого як pip. Після встановлення треба імпортувати необхідні модулі у Python-програму.

Python-RQ використовує Redis як брокер повідомлень. Для роботи з цією технологією треба встановити Redis-сервер та налаштувати параметри підключення до нього.

Для подального використання варто визначити функції, які представляють задачі, та помістити їх у чергу через Python-RQ. Задля цього треба використовувати декоратор `@job` для позначення функцій як задач. Далі потрібно надіслати задачі до черги за допомогою методу *enqueue*.

Наступним кроком є запуск обробників (workers) Python-RQ, які виконуватимуть задачі. Використання команди *rq worker* для їх запуску з командного рядка або налаштування автоматичного запуску у середовищі допоможе запустити обробники.

Python-RQ надає інтерфейс командного рядка *rq* для моніторингу та налагодження черг та обробників (workers). Команди, такі як *rq info, rq pause, rq requeue* дають доступ до інформації про стан черг та керування задачами. Також у Python-RQ можна налаштувати візуальне відображення Python-RQ Dashboard для моніторингу та тестування у веб-форматі.

Після виконання задач обробниками, результат може бути доступний для подальшої обробки. Отримати результат задачі можна за допомогою об'єкту *Job* та методу *result*.

2.3 Pyro

2.3.1 Загальні відомості

Pyro (Python Remote Objects) є фреймворком для розподілених обчислень у середовищі Python. Він надає засоби для створення розподілених систем та взаємодії між об'єктами на різних вузлах мережі. Pyro базується на концепції віддалених об'єктів, де об'єкти можуть бути передані і викликані між процесами або навіть на різних комп'ютерах [9].

Pyro є потужним і корисним інструментом для розподілених обчислень у середовищі Python. З правильним розумінням його

функціональності та відповідними уміннями у розробці, Pyro може бути використаний для побудови ефективних та масштабованих розподілених систем.

2.3.2 Переваги та недоліки

Перевагами використання Pyro для створення РС на Python є:

— ***Простота використання*:** Pyro надає простий API для визначення та виклику віддалених об'єктів, що полегшує розробку розподілених систем.

— ***Прозорість*:** Підхід Pyro дозволяє приховати складності мережевої комунікації та передачі об'єктів. Розробнику не потрібно безпосередньо працювати з протоколами мережі.

— ***Масштабованість*:** Pyro підтримує масштабування розподілених систем за допомогою можливості використання багатьох вузлів та серверів.

— ***Підтримка серіалізації*:** Pyro надає можливість серіалізувати та десеріалізувати об'єкти для передачі через мережу.

— ***Підтримка безпеки*:** Pyro надає можливості безпеки для захисту передачі даних та автентифікації. Це важливо для розподілених систем, де забезпечення безпеки є пріоритетом.

Недоліки Pyro для побудови РС складаються з таких елементів:

— ***Швидкодія***: Виклик віддалених об'єктів через мережу може мати деякий штраф до швидкості. Затримки мережевих операцій можуть впливати на продуктивність розподіленої системи.

— ***Залежність від мережі***: Робота з віддаленими об'єктами вимагає активного мережевого з'єднання. Якщо мережеве з'єднання недоступне

або непостійне, це може призвести до проблем з доступністю та стабільністю системи.

— ***Складність налаштування***: Налаштування Pyro може бути складним завданням, особливо для розподілених систем зі складною архітектурою. Вимагається розуміння концепцій віддалених об'єктів, серіалізації даних та безпеки, що може бути викликом для новачків.

— ***Обмежена підтримка***: Pyro, як будь-який інший фреймворк, має свої функціональні обмеження. Деякі розширені функції, які можуть бути необхідні в певних сценаріях, можуть вимагати додаткової розробки.

2.3.3 Використання

Для використання Pyro, необхідно налаштувати середовище та створити сервер, який надає віддалені об'єкти. Потім клієнти можуть підключатись до сервера, викликати методи віддалених об'єктів та отримувати результати виконання.

Для налаштування Pyro необхідно встановити необхідні залежності, визначити класи віддалених об'єктів, вказати правила серіалізації та десеріалізації об'єктів для передачі через мережу. На серверній стороні потрібно створити і налаштувати Pyro-об'єкт, який виконуватиме роль сервера. Цей об'єкт повинен мати методи, які будуть доступні віддаленим клієнтам.

Наступним кроком є створення клієнтського додатку, який підключатиметься до сервера Pyro і викликатиме методи віддалених об'єктів. Для цього необхідно налаштувати з'єднання з сервером та викликати методи, передаючи необхідні параметри. Клієнт отримує результати виконання методів і може їх обробити за потреби.

2.4 Інші аналоги

Поміж популярних технологій розподілених обчислень у середовищі Python, крім Celery, Python-RQ та Pyro, існують інші аналоги, які також варто розглянути для розуміння застосування кожного з них:

Dask є фреймворком для паралельних обчислень та розподілених обчислювальних задач у Python. Він надає зручний інтерфейс для створення паралельних обчислень на основі Pandas, NumPy та інших популярних бібліотек. Dask підтримує як розподілені обчислення на кластерах, так і обчислення на вузлах однієї машини.

Ray є фреймворком для РО та машинного навчання у Python. Він надає можливості для виконання паралельних та розподілених обчислень, реалізації складних графів обчислень та виконання асинхронних завдань. Ray також має підтримку для використання ресурсів розподілених кластерів.

Joblib є бібліотекою у Python, яка спрощує паралельні обчислення та збереження результатів. Вона надає інструменти для розпаралелювання обчислень на рівні функцій, методів та петель, що дозволяє прискорити виконання завдань, особливо в області обробки даних та машинного навчання.

MPI4PY є бібліотекою для розробки розподілених програм на основі стандарту MPI (Message Passing Interface). Вона надає інтерфейс для взаємодії між процесами у розподіленій системі та виконання обчислень, які вимагають обміну повідомленнями та синхронізації між процесами.

Ці інші аналоги розподілених обчислень у середовищі Python варто враховувати, оскільки вони надають різноманітні можливості для паралельних обчислень та розподілених систем. Кожен з цих інструментів має свої переваги та особливості, які можуть відповідати конкретним вимогам та сценаріям використання.

Важливо зазначити, що вибір конкретного аналогу для використання в розподілених обчисленнях залежить від потреб проекту, його масштабу, типу задач, доступних ресурсів і багатьох інших факторів. Важливо провести достатній аналіз і порівняти ці аналоги, щоб вибрати найбільш підходящий для конкретної ситуації.

У цьому підрозділі було наведено лише кілька прикладів інших технологій розподілених обчислень в середовищі Python. Цей список не є повним, і існують ще багато інших інструментів та бібліотек, які можуть бути корисними для розподілених обчислень.

2.5 Порівняльний аналіз та висновок

У цьому пункті буде проаналізована поведінка та робота оглянутих технологій у демонстраційному застосунку, який детально описаний у третьому розділі. Для порівняння інтеграції, складності застосування та налаштування, швидкодії та інших показників було обрано методи, що обґрунтовані та пояснені у наступному розділі, а саме: завдання для виконання на окремому сервері, щоб користувач міг продовжувати використовувати програму (фонове завдання) та завдання на оптимізацію обчислень завдяки РО (завдання на сортування злиттям списку з n-розмірністю).

Розглянувши та застосувавши на дії перелічені методи впровадження РО та побудови РС, було виконане порівняння використаних засобів. Порівняння містить у собі швидкодію для обраного завдання, складність роботи та інтеграцію для обраних технологій.

Швидкодія: у наведеному нижче графіку (див. рис. 2.1) зафіксовані заміри часу виконання у побудованій системі. Дослідивши ці показники

та проаналізувавши роботу цих технологій, було зроблено наступні висновки:

— ***Celery*** має потужний механізм планування завдань, підтримує асинхронну обробку та може масштабуватися горизонтально за допомогою розподіленої архітектури. Це дозволяє досягти високої швидкодії виконання завдань. У створеному проєкті був підключений лише один обробник (worker), що підтримував паралельне виконання завдань, тому що цього було достатньо для поставленої задачі.

— ***Python-RQ*** також надає асинхронну обробку завдань, але в порівнянні з Celery він може бути трохи повільнішим через меншу підтримку розподіленої обробки та масштабування**,** але через те, що у цій технології гірше реалізовані паралельні обчислення, у проєкті використовувалось 4 обробники для РО, тому результати дещо кращі від Celery, але вимагають більше ресурсів.

— ***Pyro*** є більш загальною бібліотекою для розподілених обчислень, а не просто для планування завдань. Швидкодія може бути залежна від конкретної реалізації та архітектури системи. У даному випадку – підхід з використанням даного засобу РО має найбільшу продуктивність.

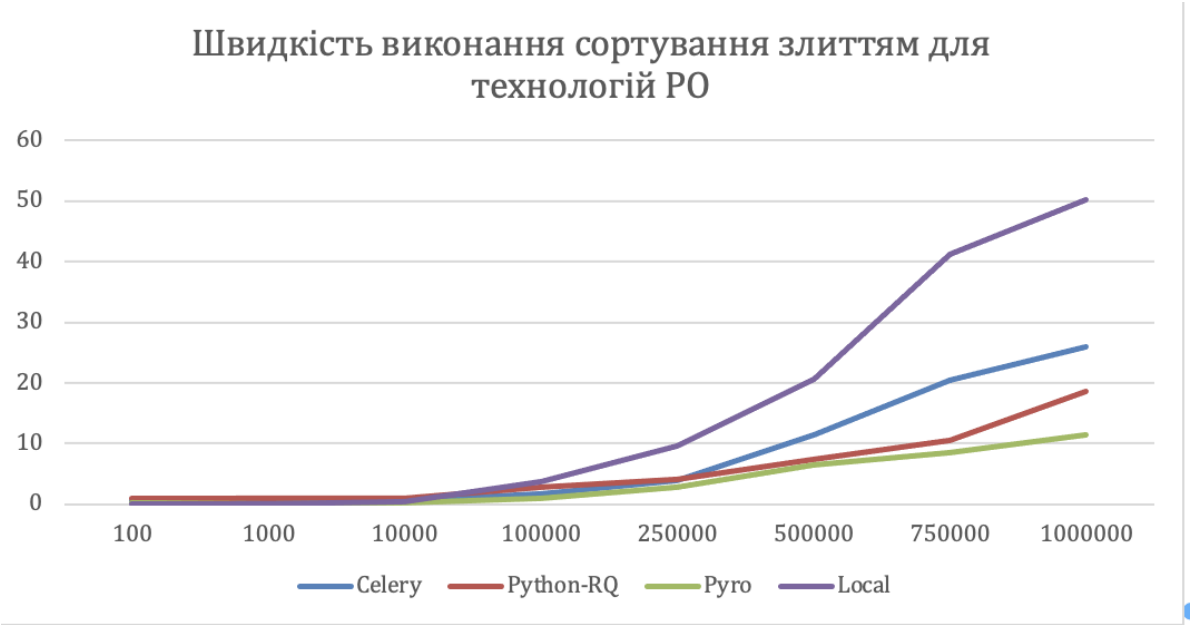

Рисунок 2.1 – Порівняння швидкодії технологій PO для Python у проєкті

Складність роботи:

— **Celery** *має високий рівень налаштування, що може вимагати додаткових зусиль для конфігурації та налагодження системи. Однак, документація та підтримка спільноти є досить широкими, що полегшує процес розробки.*

— **Python-RQ** *має просту конфігурацію та використання. Він призначений для простіших сценаріїв розподілених обчислень і має меншу кількість функцій порівняно з Celery, що робить його легким у використанні, але обмеженим у функціоналі. У цьому проєкту суттєвим недоліком стало незручна реалізація паралельних обчислень.*

— **Pyro** *може бути складним у використанні через свою загальну природу та більш широкий спектр можливостей. Він вимагає детального розуміння розподіленого програмування та побудови систем.*

Інтеграція:

— ***Celery*** має широку підтримку та інтеграцію з різними фреймворками та технологіями Python, зокрема з Django, Flask, SQLAlchemy, Redis

тощо. Його API добре документовано і дозволяє легко інтегрувати з існуючими проектами.

— ***Python-RQ*** також має підтримку для інтеграції з фреймворками та іншими бібліотеками Python, але його екосистема менш розгалужена порівняно з Celery.

— ***Pyro*** може бути інтегрований з існуючими системами Python, але вимагає додаткового коду для налаштування з'єднань та комунікації між сервером та клієнтом. Його інтеграція з іншими фреймворками виявилась найскладнішою через нестачу інформацію у документації та відсутності потрібних інструментів для інтеграції.

Загалом, вибір між Celery, Python-RQ та Pyro залежить від конкретних потреб проєкту. Celery є потужним та широко використаним рішенням з багатим функціоналом, але може вимагати більше зусиль для конфігурації і налаштування. Python-RQ є простішим у використанні, але має обмежений функціонал для більш складних сценаріїв. Pyro, з своєю загальною природою, може забезпечити більшу гнучкість та розширюваність, але вимагає більшої експертизи в розподіленому програмуванні та детальнішого налаштування.

РОЗДІЛ 3 Опис реалізації демонстраційного веб-застосунку

3.1 Обґрунтування вибору методу демонстрації та засобів розробки

Проаналізувавши побуду РС було обрано архітектуру three-tier для побудови веб-застосунку. Даний тип архітектури дозволить належним чином застосувати на практиці отримані у ході дослідження знання та порівняти продуктивність та складність інтеграції. Також це передбачає поглянути на зручність та доцільність використання таких технологій очима кінцевого користувача.

Основна ідея архітектури three-tier полягає в розділенні системи на три рівні: клієнтський, рівень бізнес-логіки та рівень доступу до даних. Давайте розглянемо, як це відображається у обраному методу демонстрації:

— ***Клієнтський рівень***: Клієнтами системи є веб-браузери, що звертаються до Django веб-додатку за допомогою HTTP запитів. Клієнти взаємодіють з додатком для отримання та відображення сторінок, виконання дій і надсилання запитів.

— ***Рівень бізнес-логіки*:** Django веб-додаток виступає в ролі рівня бізнес-логіки. Він обробляє запити від клієнтів, виконує логіку додатку, включаючи обробку форм, перевірку даних, взаємодію з базою даних та інші бізнес-операції.

— ***Рівень доступу до даних***: Щоб забезпечити асинхронне розподілене обчислення, використовється Celery та Redis. Redis виступає в ролі рівня доступу до даних, де він використовується як брокер повідомлень для передачі задач між Django додатком та обробниками (workers) Celery. Django додаток генерує задачі, які передаються до Redis, а потім обробники Celery забирають ці задачі для виконання.

Для виконання РО у додатку використовуються оглянуті у попередньому розділі технології. Було обрано саме ці рішення проблеми

РО тому, що кожна з них є доцільною та зручною для конкретних цілей розробки. Також ці технології підтримуються належним чином, є популярними та мають детальну документацію, зокрема і про зовнішню інтеграцію. На прикладі використання цих рішень у повному обсязі можна проаналізувати проблеми, інтеграцію та зручність використання РО у Python проєктах.

Для розгортки проєкту було обрано такий потужний інструмент як Docker. Docker — програмне забезпечення для автоматизації розгортання та керування додатками в середовищах з підтримкою контейнеризації, контейнеризатор додатків. Використання Docker для Django веб-застосунку з використанням технологій для РО дозволяє забезпечити ізольоване та передбачуване середовище, спростити розгортку та масштабування, забезпечити сумісність та переносимість між середовищами, спростити керування залежностями та легко інтегрувати з іншими сервісами. Всі ці переваги допомагають покращити розробку, тестування та виробничу експлуатацію Django веб-застосунку з іншими технологіями.

3.2 Розробка програми

3.2.1 Налаштування серверу та необхідних технологій

Спочатку створюємо Django застосунок на базі Python. Далі створюємо файли для налаштування Docker — Dockerfile та docker-compose.yml. У Dockerfile конфігуруємо звичне середовище для зберігання та обробки контейнерів. Також варто згенерувати requirements для продовження роботи. Це файл, що містить необхідні для роботи застосунку залежності. У docker-compose.yml треба налаштувати потрібні для обраних технологій РО брокери та сервери:

Redis – вказуємо стандартний порт та стартуємо сервер.

```
redis:
        image: redis:6-alpine
        ports:
                - 6379:6379
```

Celery – встановлюємо залежності та запускаємо обробник задач для Celery. Також вказуємо підлючення до брокеру.

```
celery:
        build: ./
        command: celery -A demo worker --loglevel=INFO
        volumes:
                - ./:/usr/src/app
        environment:
                - CELERY_BROCKER=redis://redis:6379
                - CELERY_BACKEND=redis://redis:6379
        depends_on:
                - web
                - redis
```

Python-RQ – визначаємо залежності на налаштовуємо підключення. Також конфігуруємо чергу, яка буде використовуватись.

```
python-rq:
        build: ./
        command: rq worker -u redis://redis:6379 default
        volumes:
                - ./:/usr/src/app
        environment:
                - REDIS_HOST=redis
                - REDIS_PORT=6379
                - REDIS_DB=0
                - RQ_QUEUE=default
                - LOG_LEVEL=DEBUG
        depends_on:
                - web
                - redis
```

Pyro – для Pyro потрібно визначити два контейнери. Один з них запускає сервер для обробки звернень, а інший реєструє обробник на сервері та

моніторить нові звернення завдяки циклу, визначеному у файлі pyro.py, який і виконується у другому сервісі.

```
pyro:
        build: ./
        command: pyro4-ns --host 0.0.0.0
        volumes:
                - ./:/usr/src/app
        depends_on:
                - web


pyro-connection:
        build: ./
        command: sh -c “cd pyroDemo && python3 pyro.py”
        volumes:
                - ./:/usr/src/app
        depends_on:
                - web
                - pyro
```

Web – налаштування сайту, підключення до інших сервісів та запуск сайту.

```
web:
        build: ./
        command: python manage.py runserver 0.0.0.0:8000
        volumes:
                - ./:/usr/src/app
        ports:
                - 8000:8000
        environment:
                - DEBUG=1
                - SECRET_KEY=<key>
                - DJANGO_ALLOWED_HOST=<host>
                - REDIS_HOST=redis
                - REDIS_PORT=6379
                - REDIS_PASSWORD=<password>
        depends_on:
                - redis
```

Тепер, коли налаштовано коректну роботу серверів та технологій, можна починати створювати методи обробки задач кожної з технологій. Розгорнути застосунок потрібно за допомогою команди у терміналі (MacOS): docker-compose up -d –build.

3.2.2 Створення методів реалізації демонстраційного завдання

3.2.2.1 Celery

3.2.2.1.1 Налаштування

Для початку роботи з Celery ми маємо визначити та додати екземпляр Celery. Ми називаємо це додатком Celery або скорочено просто додатком. Оскільки цей екземпляр використовується як точка входу для всього, що ви хочете зробити в Celery, інші модулі повинні мати можливість імпортувати його[]. Розглянемо файл celery.py, що виконує цю задачу:

```
import os
from celery import Celery

os.environ.setdefault("DJANGO_SETTINGS_MODULE",
"demo.settings")
app = Celery("demo")
app.config_from_object("django.conf:settings", namespace="CELERY")
app.autodiscover_tasks()
```

Спочатку завантажується конфігурація, що визначена у файлі налаштувань settings.py. Далі потрібно створити екземпляр Celery для подальшої роботи. Також виконується відслідковування задач за допомогою методу autodiscover_tasks(), що знаходить для Celery завдання, позначені декоратом @shared_task для РО.

Відтепер є можливість створювати завдання для опрацювань РО за допомогою Celery.

3.2.2.1.2 Використання

Для демонстрації визначено такі задачі: фонові обчислення та обчислення, які можна оптимізувати завдяки РО.

Розглянемо функцію для першої визначеної задачі, що реалізовує фонові обчислення, щоб користувач міг продовжувати використовувати сайт. Як завдання обрана генерація моделей студентів у локальній базі даних, що являє собою ім’я студента, його бал та чи є цей бал прохідним. Модель визначена у основному додатку таким чином:

```
from django.db import models

class StudentRate(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    mark = models.IntegerField()
    status = models.CharField(max_length=10)
```

Сама функція у застосунку має такий вигляд та буде використана в усіх розглянутих технологіях:

```
@shared_task
def create_students(total):
    for i in range(total):
        first_name = '{}'.format(get_random_string(10, string.ascii_letters))
        second_name = '{}'.format(get_random_string(10,
string.ascii_letters))
        mark = random.randint(0, 100)
        if mark >= 61:
            status = 'PASSED'
        else:
            status = 'NOT PASSED'
        s = StudentRate(first_name=first_name, last_name=second_name,
mark=mark, status=status)
        s.save()
    return '{} students created!'.format(total)
```

Як вже зазначалось раніше, для того, щоб Celery визначило задачу для виконання, потрібно використати декоратор @shared_task перед визначенням функції.

Завдання зберігатимуться в програмах для багаторазового використання, а програми для багаторазового використання не можуть залежати від самого проекту, тому неможливо імпортувати свій екземпляр програми безпосередньо.

Декоратор @shared_task дозволяє створювати завдання без конкретного екземпляра програми. Завдяки методу відслідковування у файлі створення екземпляру, Celery знайде та зареєструє це завдання на своєму сервері для обробників.

Викликається завдання для обробки Celery за допомогою методу delay(): create_students.delay(total).

delay() зручний метод, але якщо треба встановити додаткові параметри виконання, потрібно використовувати apply_async.

Для виконання другої визначеної задачі написано функцію сортування злиттям, щоб оцінити продуктивність PO для оптимізації, що також буде використовуватись у подальших технологіях:

```
@shared_task
def sort(list):
    listlen = len(list)
    if(listlen <= 1):
        return(list)
    half_listlen = listlen // 2
    left = list[:half_listlen]
    right = list[half_listlen:]
    return (merge(sort(left), sort(right)))


def merge(left, right):
    lenleft = len(left)
    lenright = len(right)

    merged = []
    i = 0
    j = 0
    while i < lenleft and j < lenright:
        if (left[i] < right[j]):
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    return merged + left[i:] + right[j:]
```

Для обробки цієї задачі варто використати метод group Celery. Цей метод створює групу завдань, які потрібно виконувати паралельно. Він містить у собі 2 параметри: масив завдань та параметри виконання. Тож для реалізації сортування злиттям було розбито масив на 4 частини та кожній задачі було визначено по одній частині. Отже для виклику та

отримання результатів використовуємо такий виклик методу group: partials = group(sort.s(part) for part in sublist)().get().

3.2.2.2 Python-RQ

3.2.2.2.1 Налаштування

Для початку роботи з цією технологією у реалізації веб-застосунку треба встановити django_rq та додати його у список встановлених застосунків (INSTALLED_APPS). Django-RQ — це проста програма, яка дозволяє налаштовувати черги в django settings.py і легко використовувати їх у своєму проекті. Налаштування для django_rq у settings.py виглядають так:

```
# RQ settings
RQ_QUEUES = {
    'default': {
        'HOST': 'redis',
        'PORT': 6379,
        'DB': 0,
    }
}

RQ_EXCEPTION_HANDLERS = ['rq.handlers.move_to_failed_queue',]
```

У RQ_QUEUES можна визначати власні черги та їх конфігурацію. Для цього проєкту буде достатньо звичайної черги. Також налаштування включають у себе конфігурацію обробників для винятків RQ_EXEPTION_HANDLERS, де можна визначити власні класи.

3.2.2.2.2 Використання

Як було зазначено у попередньому розділі, Python-RQ є більш простою аналогією Celery, тому їх застосування дуже схоже.

Для використання Python-RQ задля РО треба використовувати декоратор над функцією @job. Цей декоратор приймає аргументи для налаштування, наприклад назву черги, в якій завдання має бути виконано та час збереження результатів виконання.

Розглянемо реалізацію першої задачі засобами Python-RQ. Щоб викликати функцію з РО у Python-RQ потрібно використовувати метод enqueue, що має такі параметри: саму функцію та її аргументи. Тому виклик описаної вище функції буде виглядати так: q.enqueu(create_students, total), де q – черга, яка була налаштована.

Розглянемо реалізацію другої задачі засобами Python-RQ. Оскільки у даній технології є місце деяким спрощенням від реалізації на Celery, є і певні недоліки цього рішення. Наприклад, для паралельних обчислень нема зручного та швидкого рішення, а також немає механізмів очікування результату, тому потрібно звертати увагу на це при написанні реалізації цієї задачі. Отже, функція виклику методів для РО має таку структуру:

```
q = django_rq.get_queue()
q.empty()

partials = q.enqueue_many(
        [
            rq.Queue.prepare_data(sort, (sublist[0],), job_id=<jon_id1>),
            rq.Queue.prepare_data(sort, (sublist[1],), job_id=<jon_id2>),
            rq.Queue.prepare_data(sort, (sublist[2],), job_id=<jon_id3>),
            rq.Queue.prepare_data(sort, (sublist[3],), job_id=<jon_id4>),
        ],
)
while (partials[0].result == None):
    pass
```

Спочатку отримано поточну чергу, що була налаштована до цього. Потім вона очищується, щоб запобігти помилки. Далі потрібно скористатись методом enqueue_many(), що приймає масив завдань та параметри конфігурації. Також потрібно зачекати результатів перед

виконанням іншої частини програми. Через це втрачається продуктивність для таких задач засобами Python-RQ.

3.2.2.3 Pyro

3.2.2.3.1 Налаштування

Для початку роботи з Pyro потрібно встановити потрібні залежності та визначити клас Worker, який реєструється у пункті 1 цього підрозділу. Для цього був створений файл pyro.py:

```
@Pyro4.behavior(instance_mode="percall")
class Worker(object):
@Pyro4.expose
    def create_students(self, total):
            …

@Pyro4.expose
    def sort(self, xs):
            …

daemon = Pyro4.Daemon()
uri = daemon.register(Worker)
Pyro4.locateNS().register('StudentWorker', uri)

print('Accepting connections')
try:
    daemon.requestLoop()
except KeyboardInterrupt:
    daemon.shutdown()
print('All done')
```

Спочатку треба створити клас Worker, що буде містити в собі потрібні для реалізації функції. Наступним кроком буде реєстрація цього Worker у поточному сервері, який був налаштований.

Кожен екземпляр Pyro Daemon може стежити за будь-якою кількістю класів, відкриваючи їх функціональні можливості в мережі. Внутрішньо об’єкт Daemon буде створювати екземпляри класів, які він

приховує, за потреби. За замовчуванням об’єкти Daemon створюють екземпляри своїх зареєстрованих класів один раз на мережеве з’єднання, чого не хочеться, якщо ми маємо виконувати дії одночасно. Поведінку за замовчуванням можна змінити, використовуючи декоратори зареєстрованого класу @Pyro4.behavior(instance_mode=...). Підтримується три значення для instance_mode: single, session і percall.

Використання single означає, що Daemon створить один екземпляр класу та використовуватиме його для всіх запитів між клієнтами.

Використання session дає поведінку за замовчуванням: кожне підключення клієнта отримує новий екземпляр, який використовується для всіх викликів методів цим клієнтом.

Використовуючи instance_ mode="percall", для кожного віддаленого виклику методу створюється новий екземпляр класу.

3.2.2.3.2 Використання

Для реалізації першої задачі потрібно звернутись до зареєстрованого класу Worker на сервері та викликати потрібну функцію.

Коли визначається розташування об’єкта Pyro, створюється для нього Proxy. З Pyro віддалені виклики методів об’єктів Pyro проходять через проксі.

Проксі-сервер можна розглядати так, ніби він був фактичним об’єктом, тому ви пишете звичайний код python для виклику віддалених методів і роботи з поверненими значеннями або навіть винятками:

```
worker = Pyro4.Proxy("PYRONAME:StudentWorker")
worker.create_students(total)
```

У Proxy передбачені зручні типи атрибутів, що дозволяють знайти потрібний клас за назвою у поточному сервері.

Для реалізації другої задачі потрібно просто передати у worker 4 задачі для сортування, оскільки клас був налаштований на паралельне виконання.

3.2.3 Тестування програми

3.2.3.1 Тестування оглянутих технологій для РО

Робота з технологіями для РО складна та вимагає коректної та детальної конфігурацію задля правильного використання. Саме тому кожен крок побудови потрібно тестувати та перевіряти. У кожній з оглянутих технологій, таких як Celery, Python-RQ та Pyro, є деякі спеціальні засоби для тестування, які допомагають перевірити їх функціональність та забезпечити якість коду.

Для Celery існує зручний веб-інтерфейс Flower для моніторингу та керування Celery. Flower використовується для перевірки стану черги, задач та workers у реальному часі. Він також надає інформацію про завдання, їх стан, час виконання та інші показники, що допомагають аналізувати роботу Celery. У проєкті тестування роботи Celery здійснювалось завдяки Flower, що допомогло у правильному налаштуванні та виправленні помилок, що виникали про розробці.

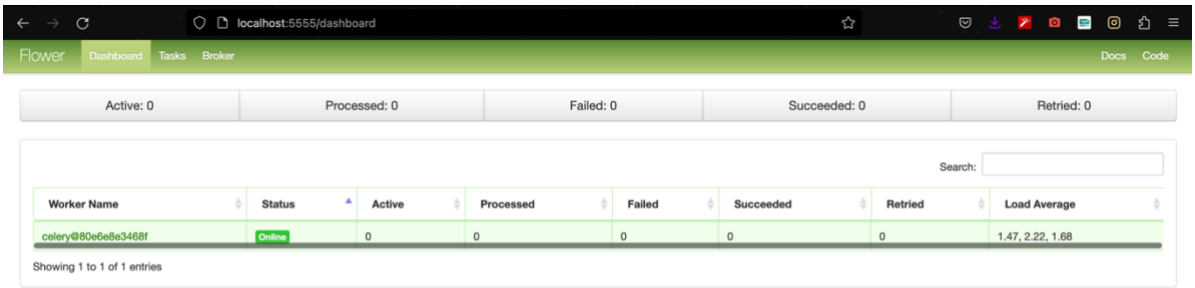

Рисунок 3.1 – Веб-інтерфейс Flower для тестування Celery

У демонстраційному застосунку було використано django_rq. Це пакет для інтеграції Python-RQ з Django. Він також надає зручні інструменти для тестування Python-RQ у контексті Django-проєктів. Завдяки django_rq можна перевірити, чи правильно налаштовані черги Python-RQ у Django-проекті, перевірити стан черги та виконання задач.

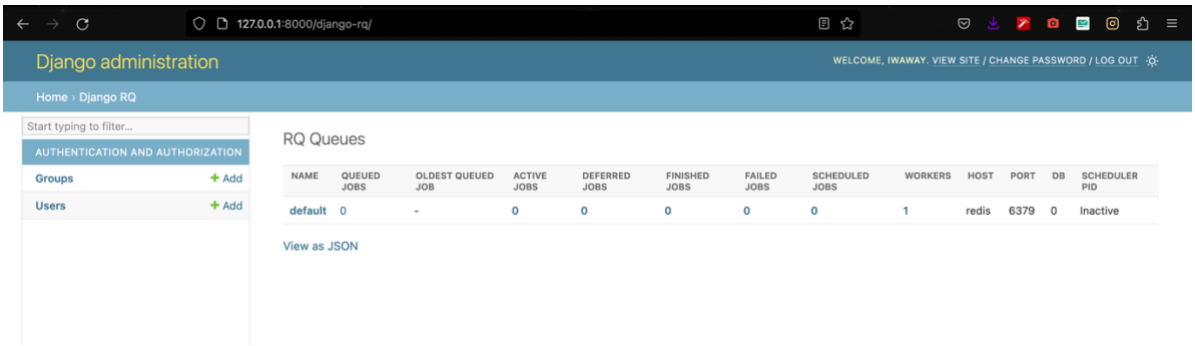

Рисунок 3.2 – Веб-інтерфейс django_rq для тестування Python-RQ

Pyro наразі не має зручного веб-інтерфейсу для моніторингу та тестування, тому всю потрібну для обробки помилок інформацію можна отримувати з вікна терміналу та налаштувавши поведінку Pyro при виникненні помилки.

3.2.3.2 Робота програми

Веб-застосунок складається з трьох вкладок для роботи з кожною технологією (Celery, Python-RQ, Pyro). Кожна сторінка містить посилання на задачу для фонового обрахування (Upload students: генерації списку студентів), задачу для оптимізації (Sort list: сортування злиттям) та

перегляд згенерованих моделей студентів (Students list: список студентів).

Рисунок 3.3 – Інтерфейс користувача для роботи з РО

Форма для введення кількості студентів, яку потрібно згенерувати для кожної реалізації містить у собі поле для введення та кнопку відправки форми.

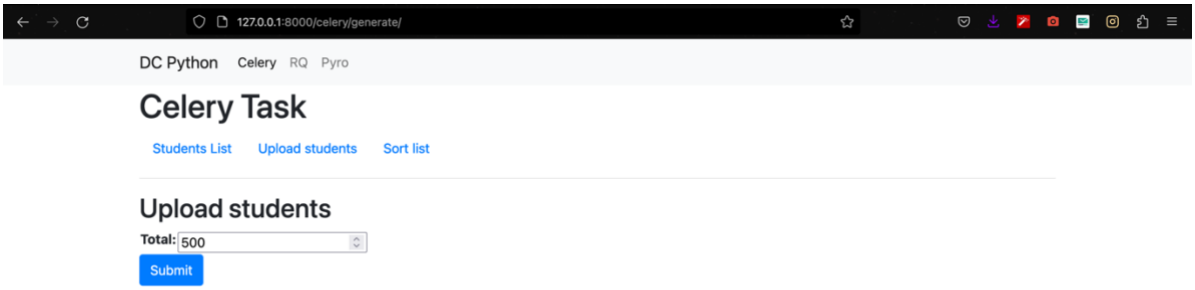

Рисунок 3.4 – Форма введення для задачі фонового обчислення

Після відправки форми відбувається переадресація на сторінку з списком студентів. З’являється повідомлення про успішне отримання завдання та його опрацювання. Поки генеруються моделі студентів користувач може продовжувати користуватись сайтом.

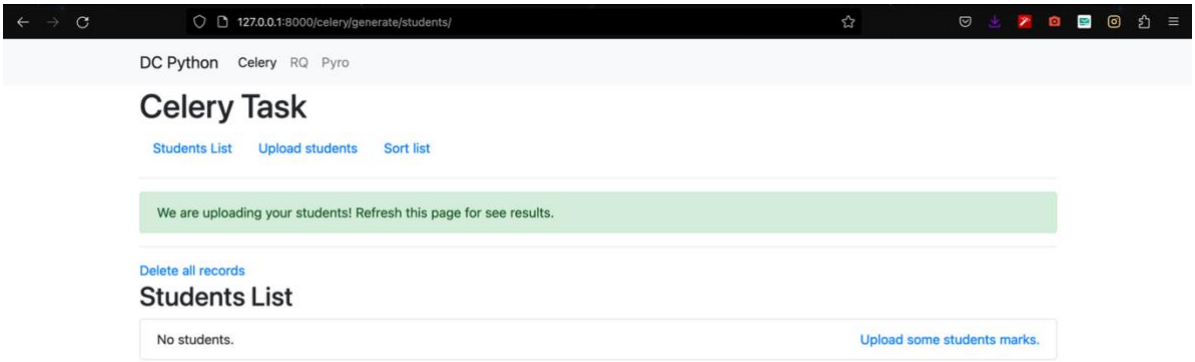

Рисунок 3.5 – Успішне отримання завдання та початок фонового обчислення

Якщо оновити сторінку – вміст сайту буде поступово поповнюватись новостворенними записами студентів (Ім’я, оцінка, статус). Також у користувача є можливість видалити всі записи з бази даних для переходу до наступної технології. Програмою передбачена поведінка у разі видалення під час виконання завдання.

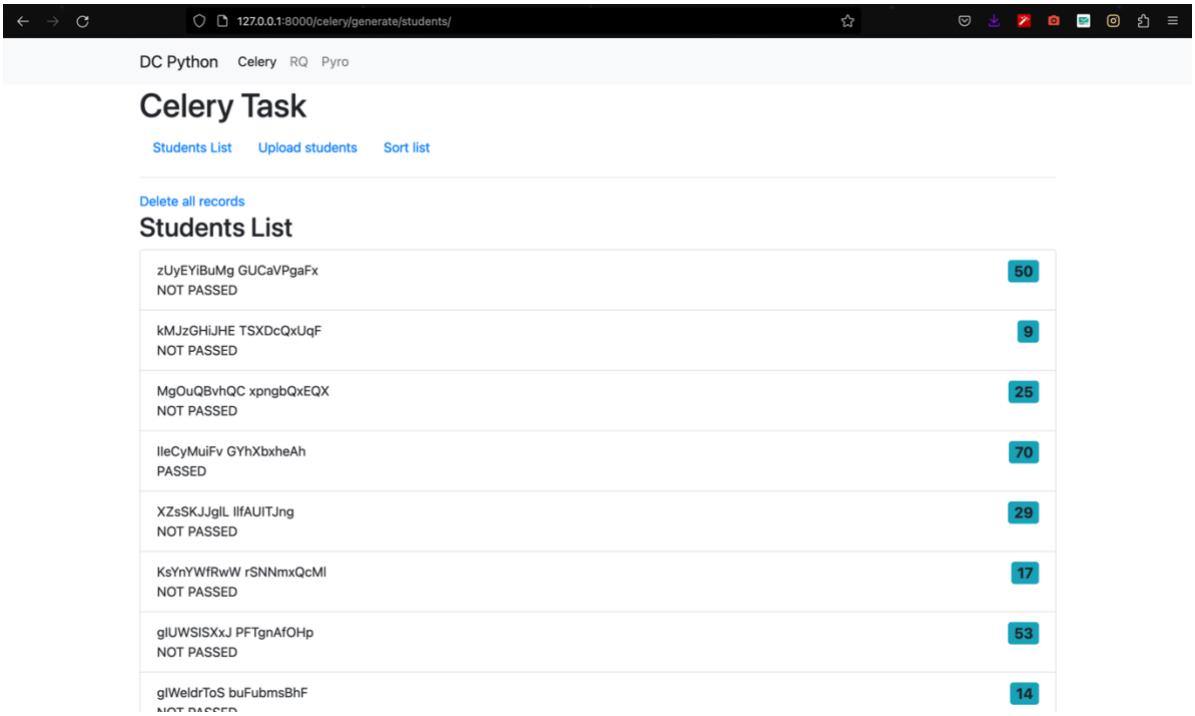

Рисунок 3.6 – Оновлений зміст сторінки списку студентів

Форма для опрацювання задачі на оптимізацію (сортування злиттям) практично така ж, як і для першої задачі. Користувач має ввести у форму розмір списку випадкових чисел для опрацювання за допомогою РО.

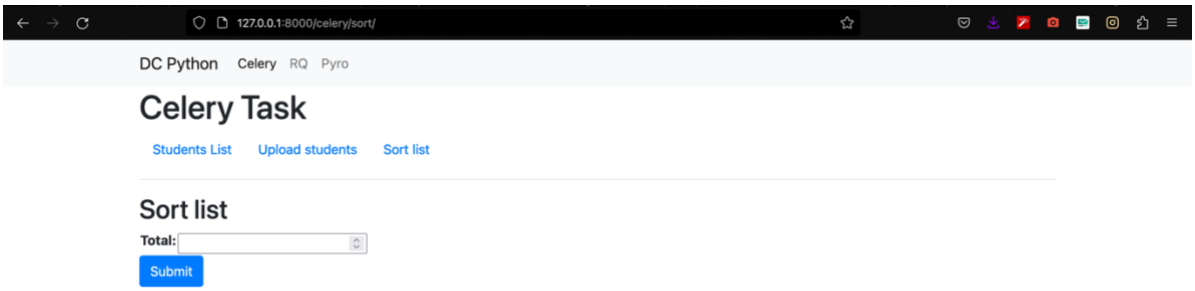

Рисунок 3.7 – Форма для опрацювання задачі на оптимізацію для РО

Після відправки форми користувач отримує повідомлення з підрахунком часу для розподіленого обрахування та для локального.

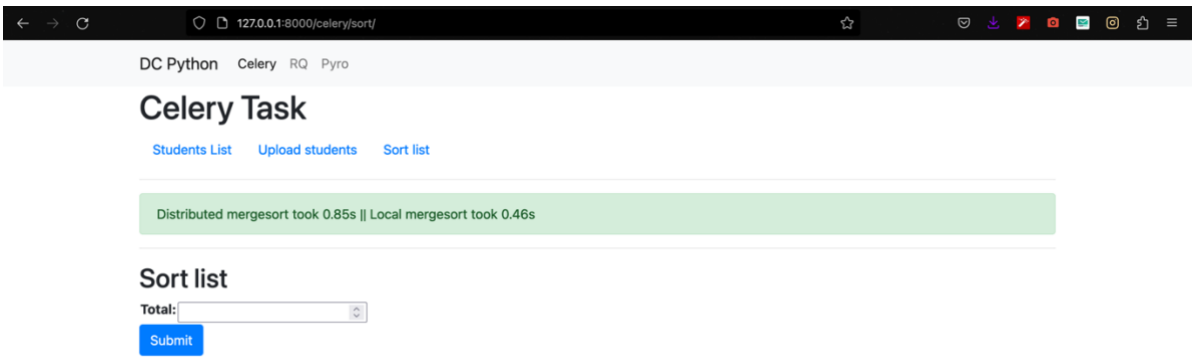

Рисунок 3.8 – Результат опрацювання задачі для розміру списку 10 000

Деталі щодо продуктивності окремих методів були описані у попередньому розділі.

ВИСНОВОК

Під час збору інформації та її вивченні було прийнято рішення реалізувати демонстраційний веб-застосунок для аналізу та порівняння технологій РО у середовищі Python, який можна надалі розширяти для нових завдань та фреймворків.

В результаті виконаної роботи мета була досягнута, а постановка задачі дотримана. Була детально розглянута побудова РС та реалізована програмно. Також був реалізований простий веб-інтерфейс для демонстрації роботи РО.

СПИСОК ДЖЕРЕЛ

1. Pierfederici F. Distributed Computing with Python / Francesco Pierfederici. – Birmingham: Packt Publishing Ltd., 2016. – 155 с.
2. The evolution of distributed computing systems: from fundamental to new frontiers / P. Garraghan, D. Smirnova, S. G. Singh, D. Lindsay. // Computing. – 2021. – №103. – С. 4–11.
3. Tanenbaum A. S. Distributed systems: principles and paradigms / A. S. Tanenbaum, M. V. Steen. – USA: Pearson Education. Inc., 2007. – 705 с.
4. Parallel distributed computing using Python / D. D. Lisandro, R. R. Paz, P. A. Kler, A. Cosimo. // Advances in Water Resources. – 2011. – №34. – С. 1124–1139.
5. Sheel K. M. Some Issues, Challenges and Problems of Distributed Software System / K. M. Sheel, A. T. Kumar. // International Journal of Computer Science and Information Technologies. – 2014. – С. 4922–4925.
6. Celery User Manual [Електронний ресурс] // Ask Solem. – 2016. – Режим доступу до ресурсу: https://docs.celeryq.dev/en/stable/index.html
7. Python Celery Software: Pros & Cons and Reviews [Електронний ресурс] // The Iron.io Blog. – 2020. – Режим доступу до ресурсу: https://blog.iron.io/python-celery-pros-cons-and-reviews/
8. RQ: Simple job queues for Python [Електронний ресурс] // Vincent Driessen. – 2012. – Режим доступу до ресурсу: https://python-rq.org/
9. Pyro Documentation [Електронний ресурс] // Uber Technologies. – 2017. – Режим доступу до ресурсу: https://docs.pyro.ai/en/stable/

ДОДАТОК А

Перелік прийнятих скорочень

РО – розподілені обчислення;

РС – розподілена система;

DSM – Distributed Shared Memory.