НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Розрахункова робота

з предмету «Проектування розподілених систем»

Виконав:

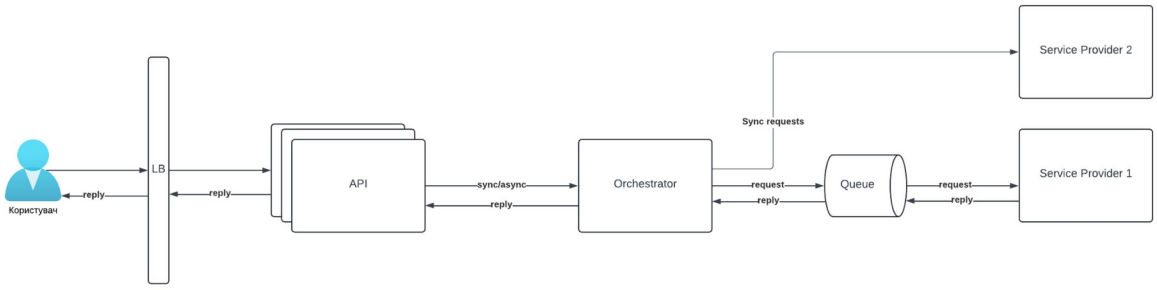
студент групи ІМ-31мн

Рекечинський Дмитро

Київ 2024

Завдання

- Реалізувати патерн оркестратор який буде керувати процесом розрахунку мат.моделі
- Побудувати математичну модель системи масового обслуговування (стая дронів) і розрахувати вплив кількості Постачальників сервісу на швидкість обробки завдань
- Зробити опис системи



Виконання завдання

Проект було створено з реалізацію паттерну Enterprise на основі моделі Event Sourcing.

Для виконання завдання було створено 6 сервісів:

- Load balancer load-balancer
- API api-service (3 реплікації)
- Оркестратор orchestrator
- Постачальник сервісу 1 queue-provider
- Постачальник сервісу 2 sync-provider
- Брокер повідомлень RabbitMQ (rabbitmq)

Сервіс load-balancer насправді є вхідною точкою до сервісу api-service. Конфігурація NGINX виглядає таким чином:

```
events {}

http {
    server {
        listen 80;

        location / {
            proxy_pass http://api-service:8000;
        }
    }
}
```

Якщо так, то за рахунок чого виконується балансування навантаження?

Docker Compose, починаючи із червня 2023 року підтримує налаштування реплікацій безпосередньо у файлі конфігурації docker-compose.yml. Налаштування api-service виглядає таким чином:

```
api-service:
  build:
    context: ./api
  deploy:
    mode: replicated
    replicas: 3
  environment:
    ORCHESTRATOR_URL: "http://orchestrator:9000/calculate"
    PORT: 8000
  networks:
    - my-network
```

Найбільш цікавою частиною є атрибут deploy. Саме в ньому визначається, в якому режимі запускається сервіс. В даному випадку встановлено режим «реплікований», і цих реплікацій — три. За умовчуванням, за реплікації та їх балансування відповідає Docker Swarm.

Якщо так, навіщо тоді вхідна точка у вигляді load-balancer?

Суть проста: для того, щоб надати доступ до сервісу api-service, треба надати доступ до порту, від контейнера до хоста. Втім, коли цей доступ надається для однієї реплікації, інша реплікація вже не може використати цей порт. Сервіс load-balancer допомагає уникнути цієї проблеми, оскільки він і так має доступ до сервісу api-service.

Оркестратор orchestrator виконує запити до queue-provider та sync-provider, тому не дивно, що в docker-compose.yml прописано залежність оркестратора від цих сервісів.

sync-provider запускається незалежно, в той час, як queue-provider залежить від того, чи запустився сервіс rabbitmq, чи ні. Для того, щоб перевірити, чи сервіс не просто запущений, а ще й готовий до підключення, використовується атрибут healthcheck:

```
healthcheck:
  test: rabbitmq-diagnostics check_port_connectivity
  interval: 10s
  timeout: 5s
  retries: 10
  start_period: 5s
```

Втім, цього лише недостатньо, щоб залежний сервіс спрацьовував лише тоді, коли сервіс rabbitmq готовий до підключень. За умовчуванням, depends_on вказує на те, що залежний контейнер запуститься тоді, коли потрібні контейнери просто запустяться, незважаючи на статус healthcheck.

Але і це можна виправити, якщо перетворити depends_on зі списку на об’єкт:

```
depends_on:
  rabbitmq:
    condition: service_healthy
```

Після цього, сервіс queue-provider буде запущений тільки тоді, коли healthcheck rabbitmq поверне успішний статус. Це, в свою чергу, впливає на сервіс orchestrator, який також залежить від rabbitmq, але оскільки orchestrator залежить і від rabbitmq, і від queue-provider, достатньо для orchestrator бути залежним від queue-provider.

Щодо суті математичної моделі, у цій системі реалізовано розв’язання множини систем лінійних алгебраїчних рівнянь методом Гауса. Приклад тіла запиту:

```
{
  "systems": [
    {
      "coefficients": [
        [10, -4, -3, 9],
        [-2, -4, -8, 2],
        [-7, 1, -8, 5],
        [-10, -4, -5, -6]
      ],
      "values": [21, -17, 17, 45]
    }
  ],
  "priority": 2,
  "format": "full"
}
```

- systems — це масив об’єктів СЛАР
    - coefficients — масив коефіцієнтів залежних змінних

    - values — масив результатів рівнянь
- priority — пріоритет виконання в черзі RabbitMQ
- format — формат виводу результатів

Демонстрація результатів

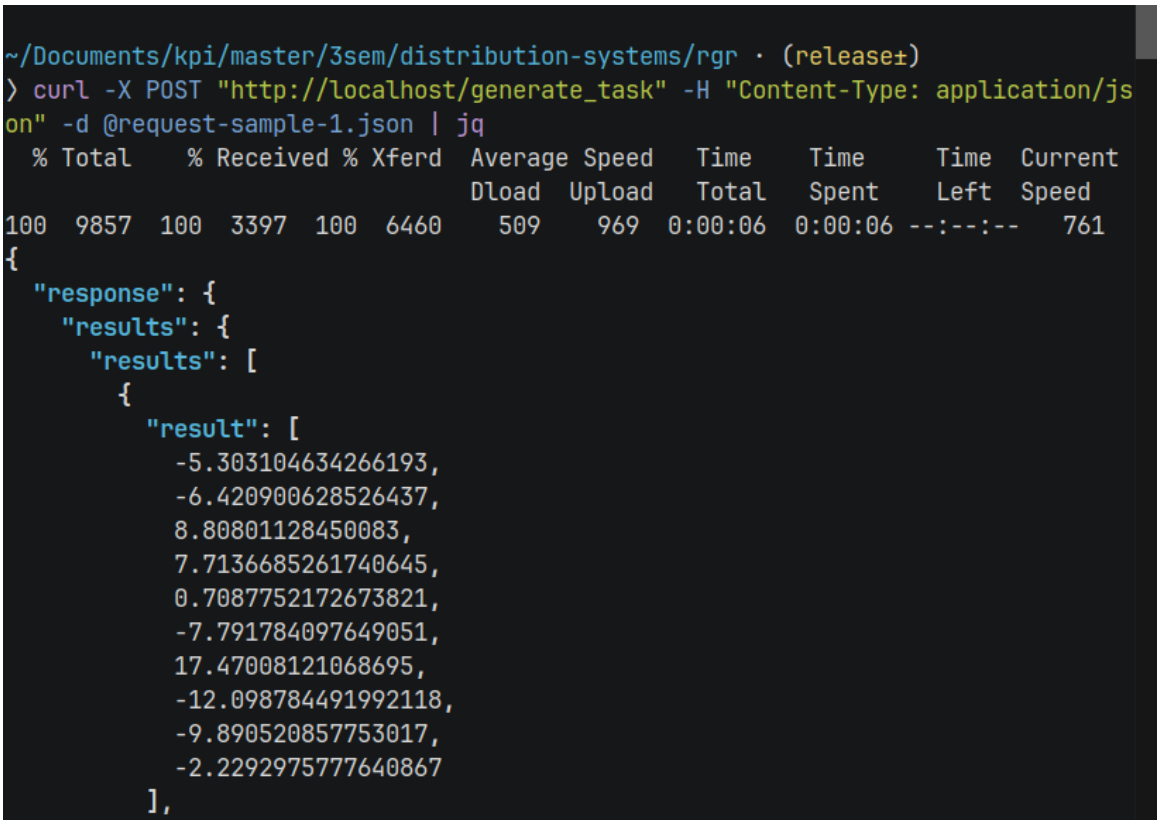


Рис. 1.1 — Запит 1 (пріотритет 2, виконувався першим)

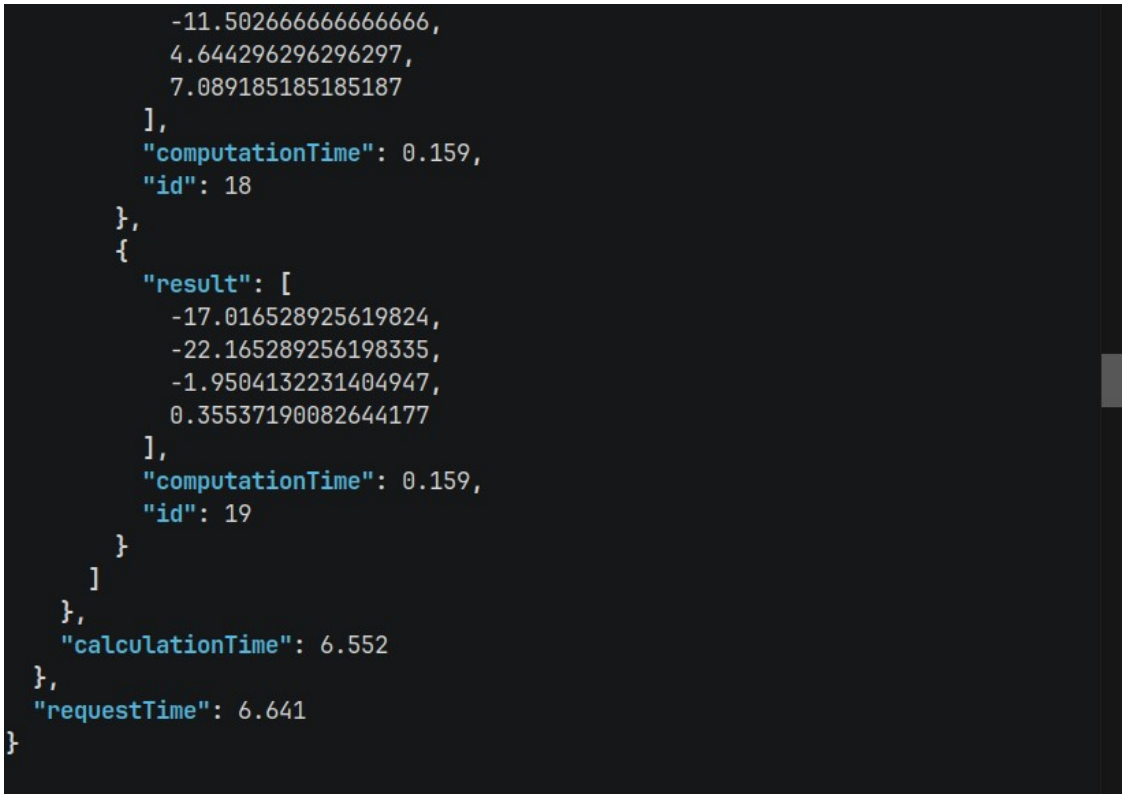


Рис. 1.2 — Запит 1 (пріотритет 2, виконувався першим)

Рис 2.1 — Запит 2 (пріотритет 1, виконувався другим)

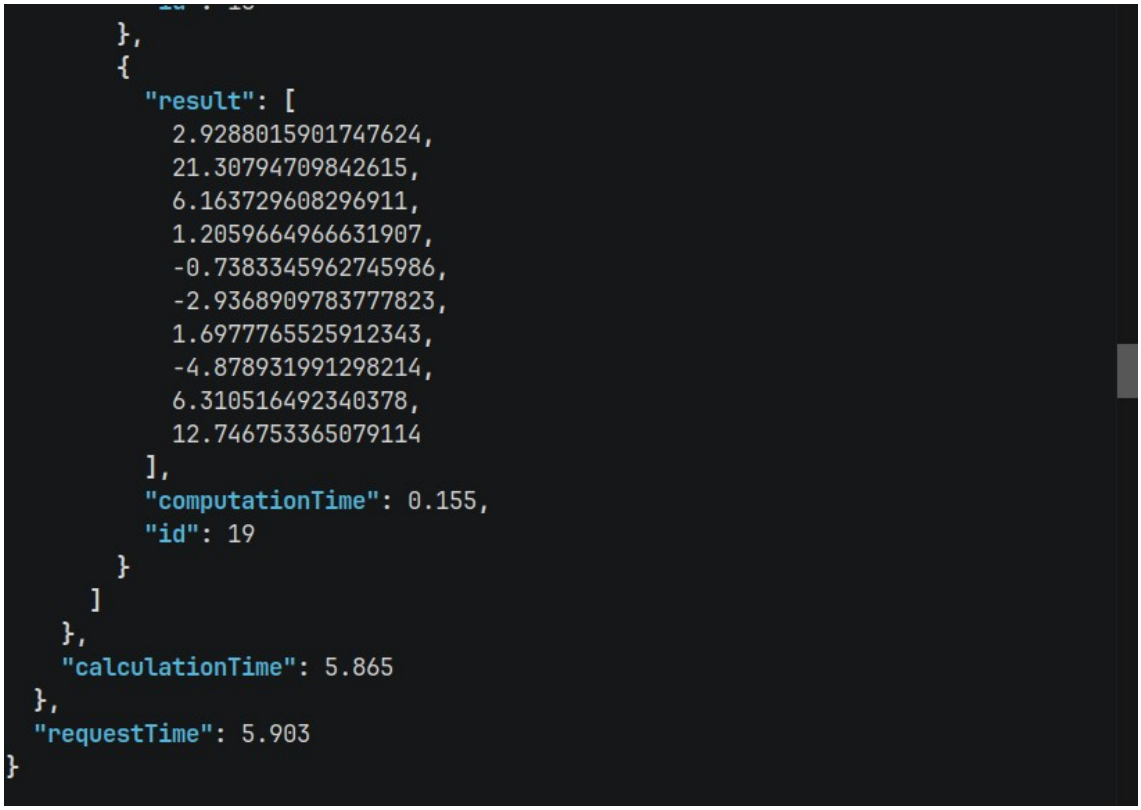


Рис 2.2 — Запит 2 (пріотритет 1, виконувався другим)

Повна версія коду проекту розміщена за веб-адресою:

https://github.com/rocket111185/distribution-systems/tree/release/rgr