УДК 004.75:004.042
DOI: 10.15587/2312-8372.2020.205151

ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ З ВИКОРИСТАННЯМ ОРКЕСТРАТОРУ НА БАЗІ ПАРАДИГМИ ПОТОКІВ ДАНИХ

Куб'юк Є. Ю., Харченко К. В.

Об'єктом дослідження даної роботи є розподілені системи під управлінням оркестратору на базі парадигми керування потоками даних, а також методи управління мікросервісами. Одним з найбільш проблемних місць сучасних розподілених систем є вибір методу управління логікою роботи мікросервісів та процесами взаємодії між ними. Існуючі концепції оркестрації та хореографії мікросервісів не дозволяють в повній мірі ефективно використовувати та розподіляти навантаження рівномірно по всій системі, що пов'язано у першу чергу з наявністю гетерогенного характеру розподіленого середовища.

В рамках дослідження пропонується концепція гібридної оркестрації на основі парадигми керування потоками даних. Даний підхід дозволяє використовувати оркестратор лише для ініціації «хвилі» обчислень по дереву мікросервісів, а за подальше обчислення та розповсюдження даних несуть відповідальність самі мікросервіси. Даний підхід, на відміну від інших, поєднав у собі більш оптимальні якості оркестрації: просте та зрозуміле, на кожному етапі обчислення, управління системою, координованість дій мікросервісів. Також, використання спеціалізованого гібридного оркестратора усунуло один з головних недоліків, а саме – зменшило відповідальність та кількість обчислювального навантаження, покладених на оркестратор розподіленої системи, та вузли обчислень. В результаті проведення експерименту з використанням розподіленої системи з оркестратором на базі парадигми керування потоками даних було досягнуто зменшення у кілька разів навантаження на сам оркестратор. Це дало можливість використовувати мікроконтролери типу ESP8266, ESP32, Raspberry Pi у якості розподіленої системи. Такі мікроконтролери можуть виступати не тільки оркестраторами, але й вузлами обчислень (dataflow nodes). У той же час, парадигма управління потоками даних дозволяє рівномірно та максимально ефективно розподіляти навантаження по системі за рахунок того, що вхідні дані системи подаються у вигляді обчислювального графу, де кожен вузол представляє собою окремий мікросервіс.

Ключові слова: *парадигма керування потоками даних, розподілені системи, високопотужні обчислення, пристрої інтернету речей, хореографія мікросервісів.*

1. Вступ

Парадигма керування потоками даних з'явилася на початку 70-х років минулого століття та віднайшла себе у багатьох аспектах розподілених систем. В рамках даної парадигми обчислювальна задача має вигляд напрямленого графу, де

вузли представляють собою операції над даними, а зв'язки між вузлами показують вхідну та вихідну точку для обчислень. В той час як архітектура сучасної мікросервісної системи досягла того рівня, коли кількість сервісів сягає кількох десятків та навіть сотень мікросервісів [1]. До таких систем прийнято застосовувати більш інтелектуальні підходи при налаштуванні злагодженої взаємодії компонентів, а також у процесі управління ними [2–4]. У випадку з системами керування потоками даних [5], а також системами на базі парадигми потоків даних [6, 7] – питання оркестрації або управління мікросервісами стає ще більш актуальним. Адже від того, як проводиться комунікація та взаємодія між компонентами буде залежати роботоспроможність всієї системи. Наявність заплутаної комунікації у системі з багатьма компонентами призводить до зниження швидкодії самої системи, а також до уповільнення розробки та відладки такої системи. У даній роботі представлено реалізацію оркестратору на базі парадигми керування потоками даних, а також варіант побудови мікросервісної системи з використанням цього оркестратору. Отже, *об'єктом дослідження* є розподілені системи під управлінням оркестратору на базі парадигми керування потоками даних, а також методи управління мікросервісами. А *метою даної роботи* є створення оркестратору на базі парадигми керування потоками даних для розподіленої системи.

2. Методика проведення досліджень

Для досягнення поставленої мети було проведене дослідження існуючих рішень по оркестрації мікросервісів з використанням парадигми потоків даних. Серед проаналізованих рішень варто виділити наступні:

– *Netflix Conductor* [8]. Використовує центральний вузол керування для управління сервісами системи. Всі операції обчислення зав'язані на вузлі керування, в тому числі і взаємодія з базою даних.

– *ZeeBe* [9]. Використовує шину повідомлень для зв'язку із сервісами, а дані (або стан) системи зберігаються на самих вузлах системи та у випадку необхідності реплікуються на декілька сервісів.

– *Uber Cadence* [10]. Також, використовує шину повідомлень для міжсервісної комунікації, проте на відміну від ZeeBe, має централізоване сховище даних, як у Netflix Conductor, що зменшує відмовостійкість системи.

Використання шини повідомлення призводить до неконтрольованості процесу обчислення в системі, тому цей підхід було відхилено у користь централізованого вузла управління. Схему роботи системи з централізованим вузлом представлено на рис. 1.

Очевидним недоліком такої системи є те, що на оркестраторі зосереджено надто багато операцій, тобто він грає роль свого роду шлюзу, який не тільки контролює процес обчислення, а ще й пропускає через себе весь потік даних від мікросервісів. Для вирішення даної проблеми – зменшення навантаження на оркестратор, при умові наявності контрольованого потоку даних, пропонується таку модель оркестрації, яка б дозволила зменшити та перенести навантаження на інші сервіси. Дана модель оркестрації представлена на рис. 2.

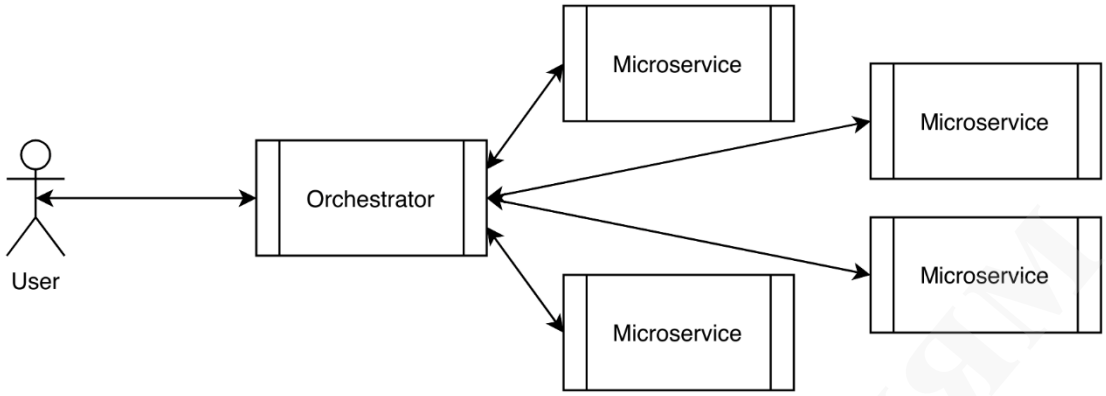

Рис. 1. Принцип роботи оркестратору з централізованим вузлом управління

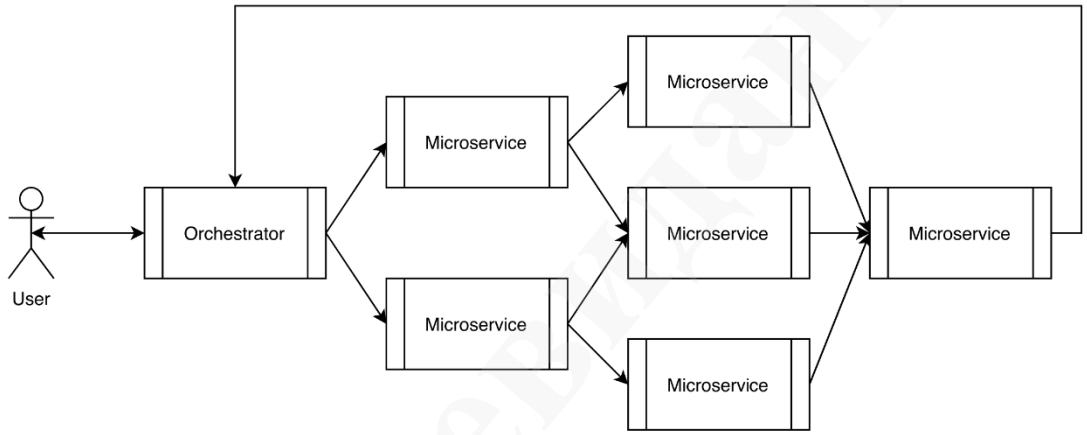

Рис. 2. Принцип роботи гібридного оркестратору на базі парадигми керування потоками даних

Як показано на рис. 2, оркестратор приймає участь в процесі обчислення лише на стадії ініціалізації та отримання результатів, в той час як кожен мікросервіс виконує по одній операції прийому та відправки даних. Особливістю даної моделі є те, що при збереженні підходу централізованого керування сервісами та здатності моніторингу стану обчислювальної задачі вдалося зменшити навантаження на вузол управління, що зробило роботу системи більш збалансованою в плані навантаження.

3. Результати дослідження та обговорення

В рамках дослідження роботи системи з оркестратором на базі парадигми керування потоками даних було створено 2 типи сервісів:

1) оркестратор;

2) сервіс-виконавець.

Оркестратор призначений для керування потоками даних в розподіленій системі, а також виконує роль шлюзу між користувачем та системою.

Сервіс-виконавець виконує роль обгортки над бізнес логікою системи, а також запроваджує функціонал для взаємодії з оркестратором та іншими сервісами-виконавцями.

UML діаграму класів спроектованої системи можна побачити на рис. 3.

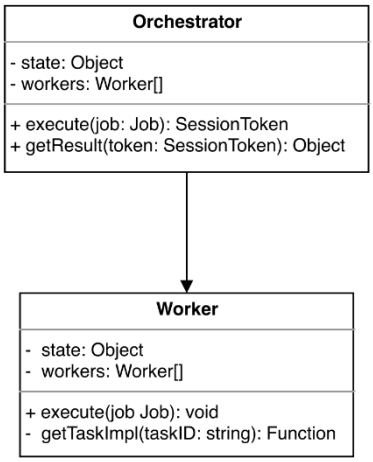

Рис. 3. UML діаграма класів системи з оркестратором на базі парадигми керування потоками даних

Кожен сервіс запускався в окремому Docker-контейнері. Структуру файлу конфігурації системи продемонстровано на рис. 4.

```
version: '3'
services:
  orchestrator:
    image: "porbs/df:orchestrator"
    ports:
      - "3000:3000"
  worker1:
    image: "porbs/df:worker"
    ports:
        - "3001:3000"
  worker2:
    image: "porbs/df:worker"
    ports:
        - "3002:3000"
  worker3:
    image: "porbs/df:worker"
    ports:
        - "3003:3000"
```

Рис. 4. Файл конфігурації системи з оркестратором на базі парадигми керування потоками даних

Згідно з файлом конфігурації системи з оркестратором на базі парадигми керування потоками даних, кількість сервісів, які приймали участь у експерименті, складає 4, з яких: 1 сервіс-оркестратор та 3 сервіси-виконавці.

Даний файл конфігурації відповідає наступній структурі системи (рис. 5).

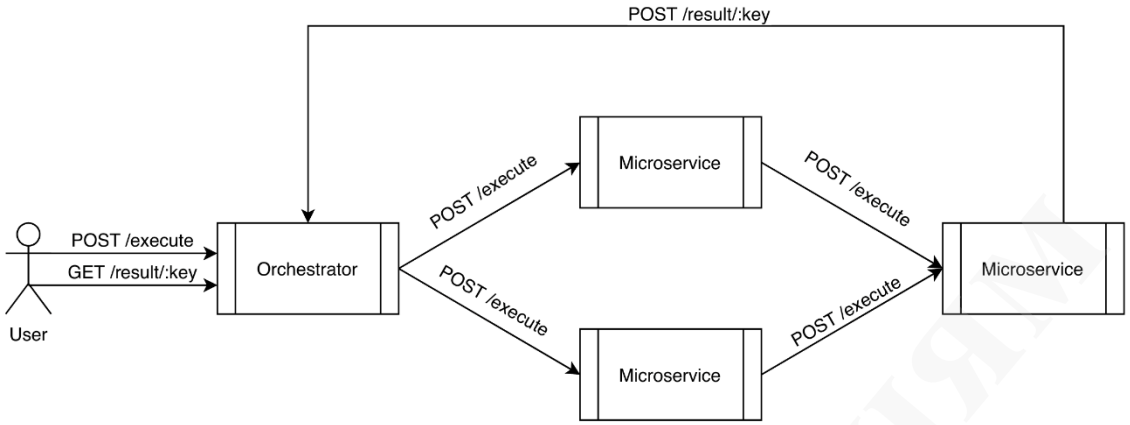

Рис. 5. Схема робочого процесу системи оркестратором на базі парадигми керування потоками даних

Система працює наступним чином:

1. Користувач робить POST/execute запит на оркестратор та передає файл конфігурації обчислювальної задачі (оформлений у вигляді напрямленого графу).

2. Як тільки оркестратор отримує конфігурацію він починає діяти за наступним протоколом:

1) оркестратор виконує процедуру призначення окремого сервісу до обчислювального вузла у файлі конфігурації за принципом round-robin;

2) оркестратор обчислює, які вузли (сервіси) системи мають всі вхідні дані для початку обчислення (фактично, оркестратор шукає «листки» дерева обчислень) та поширює файл конфігурації до них за допомогою POST/execute запиту.

3. Як тільки сервіс-виконавець отримує конфігурацію він починає діяти за наступним протоколом:

1) сервіс-виконавець перевіряє, чи всі вхідні дані доступні. Якщо ні, то сервіс буде чекати. Якщо всі дані для обчислення вже у наявності, то виконати обчислення;

2) сервіс-виконавець записує результат обчислення у файл конфігурації та поширює його за допомогою POST/execute запиту, згідно файлу конфігурації. Якщо вихідним вузлом для даного мікросервісу є сам оркестратор, то замість POST/execute запиту, виконується POST/result/:key запит, де key – параметр запиту, який містить ключ обчислювальної сесії.

Приклад файлу конфігурації обчислювальної задачі у форматі JSON представлений на рис. 6.

Робоча машина, на якій виконувалися всі тести має наступну конфігурацію:

– CPU: 3.4 GHz x 6;

– RAM: 8 GB.

```
{
    "$a": {
        "inputs": [3, 2],
        "outputs": ["$c"]
    },
   "$b": {
        "inputs": [3, 4],
        "outputs": ["$c"]
    },
    "$c": {
        "inputs": ["$a", "$b"],
        "outputs": ["$result"]
    }
}
```

Рис. 6. Приклад файлу конфігурації обчислювальної задачі

Споживання ресурсів описаної вище системи продемонстровано у табл. 1.

Таблиця 1

Споживання ресурсів системи оркестратора та сервісу-виконавця

Сервіс	CPU (%)	RAM (MB)	Розмір контейнера (MB)	Об’єм потоку даних (КБ)
orchestrator	0.44	42.8	55	0.235
worker	0.45–0.46	47–48	55	0.235–0.244

Таким чином, беручи до уваги результати тесту навантаження та споживання ресурсів системи, можна зробити висновок, що створений оркестратор можна використовувати на пристроях інтернету речей.

4. Висновки

В результаті проведення досліджень було спроектовано та реалізовано систему з оркестратором на базі парадигми керування потоками даних. Дана система має ряд переваг, порівняно з подібними системами оркестрації [8–10]:

1) відносно малий розмір та низький рівень споживання ресурсів;

2) рівномірне розподілення навантаження на систему;

3) контрольований потік даних;

4) неблокуючі операції обчислення (одночасно можуть виконуватись декілька незалежних задач).

Система була випробувана на прототипі коду та реалізована на мові JavaScript у середовищі Node.js. Дане середовище дозволяє виконувати JavaScript код та взаємодіяти з файловою системою.

Наступними етапами роботи планується дослідити та реалізувати такі задачі:

1) відсутність служби виявлення сервісів (задля усунення необхідності мануальної конфігурації сервісів);

2) відсутність механізму відмовостійкості сервісів;

3) відсутність програмного сервісу-шлюзу, який би дозволив шифрувати трафік між користувачем та системою.

В цілому, система показала, що реалізацію оркестратора для керування потоками даних можна швидко та ефективно реалізувати в середовищі скриптової мови, наприклад, у JavaScript. Це дає можливість використовувати даний підхід для побудови оркестраторів для малопотужних обчислювальних систем, наприклад, у пристроях Інтернету речей.

Література

1. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L. (2017). *Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering*. Cham: Springer, 195–216. doi: http://doi.org/10.1007/978-3-319-67425-4_12

2. Petrenko, A. I., Bulakh, B. V. (2018). Intelligent Service Discovery and Orchestration. *Proc. of 2018 IEEE First International Conference on System Analysis and Intelligent Computing (SAIC)*. Kyiv, 201–205. doi: http://doi.org/10.1109/saic.2018.8516723

3. Petrenko, A., Bulakh, B. (2019). Automatic Service Orchestration for e-Health Application. *Advances in Science, Technology and Engineering Systems Journal, 4 (4),* 244–250. doi: http://doi.org/10.25046/aj040430

4. Petrenko, O. O. (2015). Porivniannia typiv arkhitektury system servisiv. *Systemni doslidzhennia i informatsiini tekhnolohii, 4,* 48–62.

5. Kharchenko, K. V. (2016). An Architecture and Test Implementation of Data Flow Virtual Machine. *System Analysis and Informatin Technology Conference.* Kyiv: IASA NTUU-KPI, 268.

6. Kharchenko, K., Beznosyk, O., Romanov, V. (2018). Implementation of Neural Networks with Help of a Data Flow Virtual Machine. *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP),* 407–409. doi: http://doi.org/10.1109/dsmp.2018.8478455

7. Kharchenko, K., Beznosyk, O., Romanov, V. (2017). A set of instructions for data flow virtual machine. *2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON).* Kyiv, 931–934. doi: http://doi.org/10.1109/ukrcon.2017.8100385

8. *Netflix Conductor*. Available at: https://netflix.github.io/conductor/

9. *Zeebe*. Available at: https://docs.zeebe.io/index.html

10. *Uber Cadence*. Available at: https://cadenceworkflow.io/