UDC 004.75:004.042
DOI: 10.15587/2312-8372.2020.205151

Kubiuk Y.,
Kharchenko K.

DESIGN AND IMPLEMENTATION OF THE DISTRIBUTED SYSTEM USING AN ORCHESTRATOR BASED ON THE DATA FLOW PARADIGM

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.*

Received date: 12.02.2020
Accepted date: 18.03.2020
Published date: 30.06.2020

1. Introduction

The data flow management paradigm appeared in the early 70s of the last century and found itself in many aspects of distributed systems. Within the framework of this paradigm, the computational problem has the form of a directed graph, where the nodes represent operations on the data, and the connections between the nodes show the incoming and outgoing points for calculations. At the same time, the architecture of a modern microservice system has reached the level when the number of services reaches several tens or even hundreds of microservices [1]. It is customary to apply more intelligent approaches to such systems when setting up coordinated interaction of components, as well as in the process of controlling them [2–4]. In the case of data flow control systems [5], as well as systems based on the data flow paradigm [6, 7], the issue of orchestration or control of microservices becomes even more relevant. Indeed, the operation of the entire system will depend on how communication and interaction between the components is carried out. The presence of confusing communication in a system with many components leads to a decrease in the speed of the system itself, as well as to a slowdown in the development and debugging of such a system. This paper presents an implementation of an orchestrator based on the data flow control paradigm, as well as an option to build a microservice system using this orchestrator. So, *the object of research* is distributed systems under the control of an orchestrator based on the data flow control paradigm, as well as microservice management methods. And *the aim of research* is to create an orchestrator based on the data flow control paradigm for a distributed system.

2. Methods of research

To achieve this aim, a study is conducted of existing solutions for microservice orchestration using the data flow paradigm. Among the analyzed solutions, it is worth highlighting the following:

– *Netflix Conductor* [8]. It uses a central management node to manage system services. All calculation opera-

tions are tied to the control node, including interaction with the database.

– *ZeeBe* [9]. It uses a message bus to communicate with services, and the data (or status) of the system is stored on the nodes of the system itself and, if necessary, is replicated to several services.

– *Uber Cadence* [10]. It also uses a message bus for inter-service communication, however, unlike ZeeBe, it has a centralized data storage, as in Netflix Conductor, which reduces system fault tolerance.

The use of the message bus leads to an uncontrollability of the calculation process in the system, therefore this approach was rejected in favor of a centralized control node. The scheme of operation of the system with a centralized node is shown in Fig. 1.

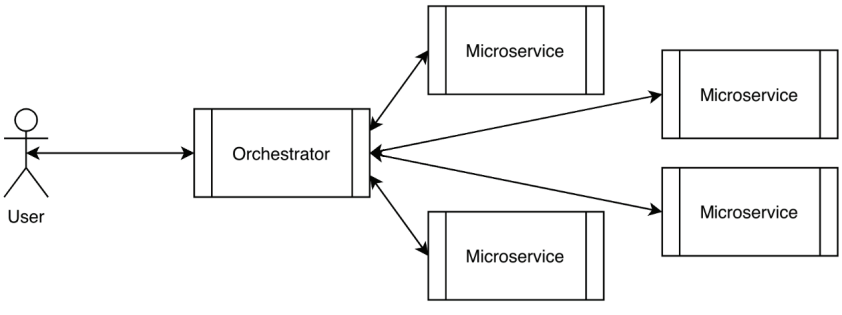

Fig. 1. The principle of orchestrator operation with a centralized control unit

The obvious drawback of such a system is that too many operations are concentrated on the orchestrator, that is, it plays the role of a kind of gateway that not only controls the calculation process, but also passes the entire data stream from microservices through itself. To solve this problem – reducing the load on the orchestrator, subject to the availability of a controlled data stream, let's propose such a model of orchestration, which made it possible to reduce and transfer the load to other services. This model of orchestration is presented in Fig. 2.

As shown in Fig. 2, the orchestrator takes part in the calculation process only at the initialization and receiving stage, while each microservice performs one operation of receiving and sending data. The peculiarity of this model is that while maintaining the centralized service management approach and the ability to monitor the state of the computational task, it is possible to reduce the load on the control node, making the system more balanced in terms of load.

3. Research results and discussion

As part of the study of the operation of the system with an orchestrator, based on the data flow control paradigm, 2 types of services are created:

1) orchestrator;

2) service worker.

The orchestrator is designed to control data flows in a distributed system, and also acts as a gateway between the user and the system.

The service worker acts as a wrapper over the business logic of the system, and also introduces functionality for interacting with the orchestrator and other service providers.

The UML class diagram of the designed system can be seen in Fig. 3.

Each service was launched in a separate Docker container. The structure of the system configuration file is shown in Fig. 4.

According to the system configuration file with the orchestrator based on the data flow control paradigm, the number of services that took part in the experiment is 4, of which: 1 service orchestrator and 3 service executors.

This configuration file corresponds to the following system structure (Fig. 5).

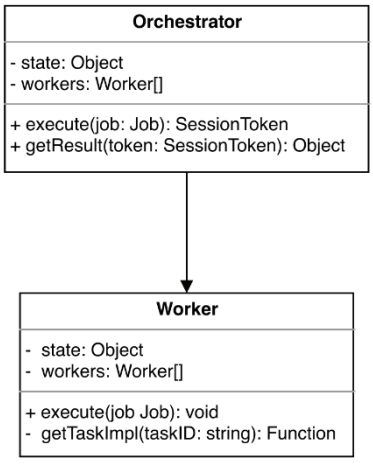

Fig. 3. UML class diagram of a system with an orchestrator based on a data flow control paradigm

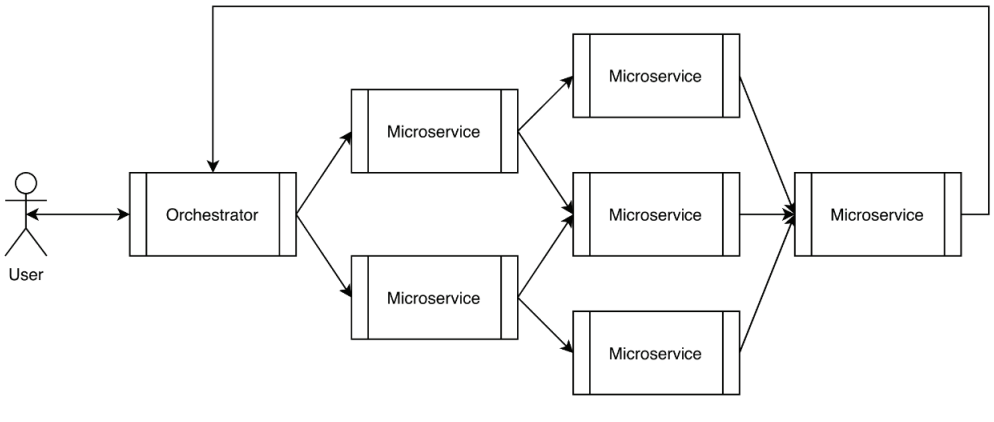

Fig. 2. The principle of the hybrid orchestrator based on the data flow control paradigm

 ISSN 2664-9969

```
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"
```

Fig. 4. System configuration file with an orchestrator based on the data flow control paradigm

The system works as follows:

1. The user makes a POST/execute request to the orchestrator and transmits the configuration file of the computational task (in the form of a directed graph).

2. As soon as the orchestrator receives the configuration, it begins to act according to the following protocol:

1) the orchestrator performs the procedure of assigning a separate service to the computing node in the configuration file according to the round-robin principle;

2) the orchestrator calculates which nodes (services) of the system have all the input data for starting the calculation (in fact, the orchestrator searches for the «leaves» of the calculation tree) and distributes the configuration file to them using the POST/execute request.

3. As soon as the service provider receives the configuration, it begins to act according to the following protocol:

1) the service provider checks that all incoming data is available. If not, the service will wait. If all the data for the calculation is already available, then perform the calculation;

2) the service worker writes the calculation result to the configuration file and distributes it using the POST/execute request, according to the configuration file. If the source node for this microservice is the orchestrator itself, then instead of POST/execute request, a POST/result/:key request is executed, where key is the request parameter that contains the key of the computing session.

An example of a configuration file for a computational task in JSON format is shown in Fig. 6.

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

Fig. 6. Example of a configuration file for a computational task

The working machine on which all tests were performed has the following configuration:

– CPU: 3.4 GHz×6;

– RAM: 8 GB.

The resource consumption of the system described above is shown in Table 1.

Table 1

Consumption of resources of the orchestrator system and service provider

Service	CPU (%)	RAM (MB)	Container size (MB)	Data stream volume (KB)
orchestrator	0.44	42.8	55	0.235
worker	0.45–0.46	47–48	55	0.235–0.244

Thus, taking into account the results of the test of load and consumption of system resources, it is possible to conclude that the created orchestrator can be used on devices of the Internet of things.

4. Conclusions

As a result of the research, a system with an orchestrator is designed and implemented based on the data flow control paradigm. This system has several advantages compared to similar orchestration systems [8–10]:

1) relatively small size and low level of resource consumption;

2) uniform distribution of load on the system;

3) controlled data flow;

4) non-blocking calculation operations (several independent tasks can be performed simultaneously).

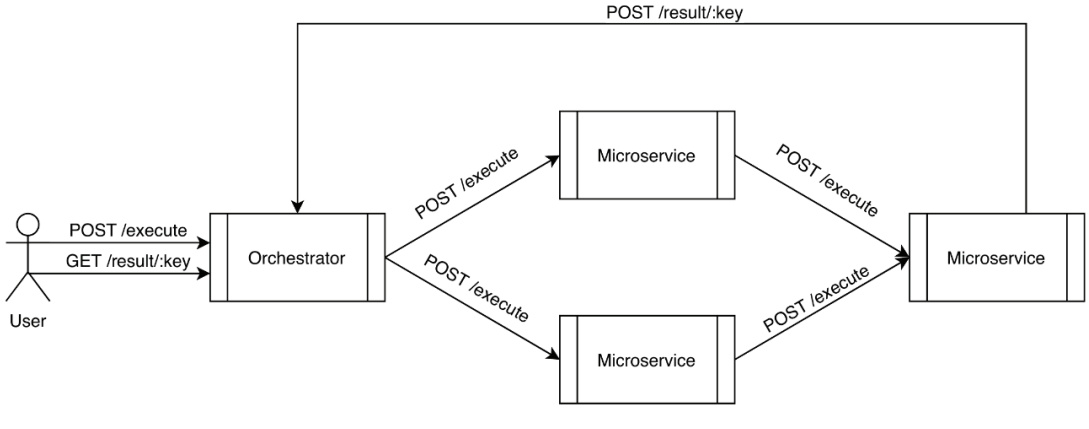

Fig. 5. Scheme of the work process of the orchestrator system based on the data flow control paradigm

The system was tested on a prototype code and implemented in JavaScript in Node.js. This environment allows to execute JavaScript code and interact with the file system.

The following stages of work are planned to investigate and implement the following tasks:

1) lack of service discovery services (to eliminate the need for manual configuration of services);

2) lack of fault tolerance services;

3) lack of a gateway software service that would allow traffic to be encrypted between the user and the system.

In general, the system showed that the implementation of an orchestrator for managing data flows can be quickly and efficiently implemented in a scripting language environment, for example, in JavaScript. This makes it possible to use this approach to build an orchestra for low-power computing systems, for example, in the devices of the Internet of things.

References

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/

Kubiuk Yevhenii, *Department of System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Ukraine, e-mail: eugen.kubiuk@gmail.com, ORCID: http://orcid.org/0000-0002-7086-0976*

Kharchenko Kostiantyn, *PhD, Department of System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Ukraine, e-mail: konst1970@gmail.com, ORCID: http://orcid.org/0000-0002-7334-8038*