Proposal of inter-node communication method using P2P communication in multi-node computing platform

JUN LIANG[†1] YANZHI LI[†2]
SUGAYA MIDORI[†3]

Abstract: In recent years, serverless platforms such as OpenWhisk that provide FaaS computing services have become popular. These services improve computational efficiency by assembling and executing multiple functions in a pipeline. However, there are not enough proposed methods to reduce the network load of the controller due to the increasing of data transferred during the concurrent execution of multiple pipelines.
In this proposal, to reduce the load on the controller of a multi-node platform, we solve the problem by direct data transmission between the functions without going through the controller by using the Peer-to-Peer communication between the calculation nodes that execute each function. Simulated and compared network models through the controller confirm that the proposed method improves the performance of multi-node platforms.

Keywords: FaaS, Serverless, peer to peer,

1. Introduction

FaaS(Function as a Service) is a kind of cloud computing service that allows developers to build, compute, run, and manage application packages as functions without having to maintain their own infrastructure[1]. These days, the methodologies are widely used for the cloud since it makes possible to manage the various applications easily for the cloud computing users, who is not necessary care about the geological place that server placed on. For that meaning, FaaS is a subset of serverless. Serverless is focused on any service category, be it compute, storage, database, messaging, API gateways, etc. where configuration, management, and billing of servers are invisible to the end users of the FaaS service [2]. Some studies prove that serverless platform is cost saving in deploying microservices rather than building out traditional applications [3][4].

OpenWhisk is an open-source platform that provides FaaS services developed by Apache. Developers can write function programs for OpenWhisk in supported programming languages. These function programs are formed into the smallest execution unit called “action”, which operates according to a fixed rule. The inputting of an *action* must be a dictionary, and the execution result of an action also must be a dictionary [5]. It is possible to handle complex computational processes by combining multiple actions and eliminating the actions in a fixed order. A set of combined actions in a processing pipeline is called a ”sequence” in the OpenWhisk.

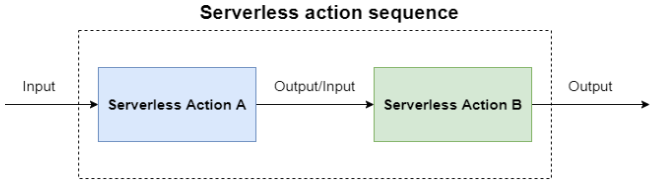

Figure 1. Sequence structure in OpenWhisk[6]

2. Problem

Generally, a FaaS platforms would have a controller module. In the OpenWhisk platform, the controller module consists of a multi-node system, and each node executing actions. The actions are separated and communicate through the network. Figure 2 is an example of the execution progress of OpenWhisk. All actions are executed in containers as shown by the arrows 15 to 19 in the pictures. The red line indicates that the result of executing a function inside the action, which is always transmitted to the controller. The controller which is pointed by the arrow 20 can communicate with the action execution node and pass input data to start the execution of the next action. The *invokeNextAction()* function about the controller in the OpenWhisk source code also shows the controller accepts the previous action’s response which contains the execution results and passes these results to the next action[7]. In other words, all output data from an action is transmitted as input data to the next action via the controller.

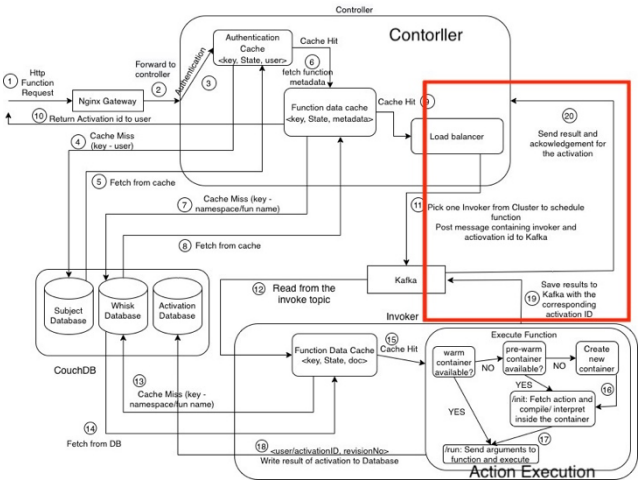

Figure 2. OpenWhisk execution progress[8]

To use the computing resources as much possible, it is common

†1, †2, †3 Shibaura Institute of Technology
3-7-5 Toyosu, Koto-ku, Tokyo 135-8548, Japan

ⓒ 2023 Information Processing Society of Japan

to execute multiple sequences simultaneously on the platform. When data transmission between Actions in multiple Sequences occurs simultaneously, the amount of communication increases. Since all data between Actions are transmitted to the next Action via the controller, when multiple Actions transmit data to the controller at the same time, the amount of communication in the controller section increases to an extreme level, and there is a possibility that this will cause a bottleneck in the system. This problem also causes communication delay problems, and the execution of an Action by communication with the controller may be terminated due to time out.

3. Proposal and Design

To reduce the amount of communication from the execution node of a function to the controller in a multi-node computation platform in the OpenWhisk, we propose a direct peer-to-peer communication function, which makes it possible to realize direct data transmission between the functions by building between the execution nodes of the functions. Due to the nature of the execution process of FaaS platforms in OpenWhisk, the data transmission is determined to be unidirectional and one-to-one, and the peer-to-peer communication function between function nodes for the FaaS platform is also assumed to be a single node unidirectional communication. The designed peer-to-peer communication data flow is shown in Figure 3.

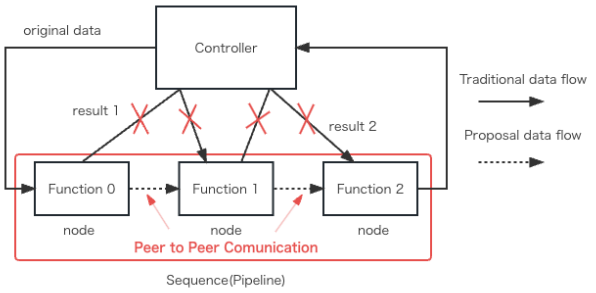

Figure 3. Distinguish between the proposed communication data flow and conventional method

3.1 Specific Design

The construction of a node-to-node peer-to-peer function for a multi-node computing platform also requires modification of the controller part.

According to the data flow shown in Figure 2, when the execution of a function in an action is completed, the controller returns an execution completion ack and the result to the requestor. All function results are always returned to the controller rather than passed directly to the next action node. The reason is that the function developer does not consider the combination of functions but concentrates on the function itself. Following this philosophy, the total peer-to-peer module is separated from the function execution module and becomes an independent module.

The function of the controller to return results and completion acks should be retained in the design of the proposal. Therefore, the proposal implements both communication model: via the controller and peer-to-peer communication. When the requestor writes the logic program, it should determine whether to use peer-to-peer communication between nodes or the traditional via-controller of the transmission mode. The purpose of peer-to-peer communication between nodes is to reduce the amount of data transmission to the controller. To reach this goal, the node which finished action would send only the calculation results to the next node via peer-to-peer communication. The controller can still receive the acks from the nodes that finished action and return acks to the requestors.

For information requiring peer-to-peer communication. It is necessary for a node belonging to a sequence (pipeline) to know the IP address of other nodes belonging to the same sequence. It is also necessary to know to which node a node will communicate next. This is determined by the controller at the beginning of the establishment of a sequence and sent to the nodes belonging to that sequence.

Peer-to-peer module is consisted by 2 individual programs. Every node has both client and server program. Data is sent from the client program to the server program by using HTTP Post method.

3.2 Implementation

This implementation is for the MEC-RM Platform, which is a FaaS platform that manages execution nodes with different architectures and provides a unified interface.

The peer-to-peer communication function between nodes is the C-Library for the Lua environment. The result of the function execution is passed to Lua virtual machine, and the Lua virtual machine calls the peer-to-peer communication module for transferring the execution result to the next node.

The client part of the peer-to-peer module uses the lib-curl C library, and the server part uses the mongoose C library.

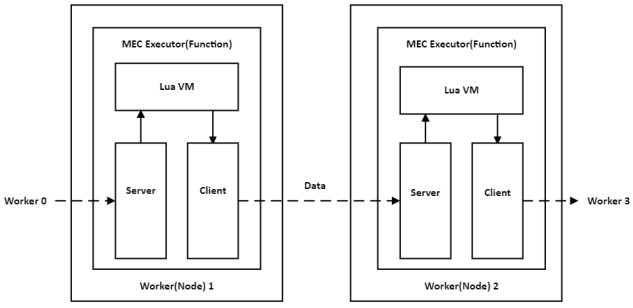

Figure 4. Design overview on MEC-RM Platform

4. Evaluation

Experiments were conducted to prove that peer-to-peer communication between nodes on a platform providing FaaS computation services can reduce communication bottlenecks.

This experiment was conducted by simulating and measuring the peer-to-peer communication and the situation via the controller with the program developed by myself.

The controller program uses I/O multiplexing to transmit data from node A to node B, which is specified in advance. The experiment is a unidirectional communication between two nodes with 2 or 3 sequences of simultaneous execution. The size of the

ⓒ 2023 Information Processing Society of Japan

transmitted data was 300 MB, and the time for the experiments via the controller was measured at the controller server, and the time for the peer-to-peer experiment was measured at the sender. Experiments were conducted on Raspberry Pi.

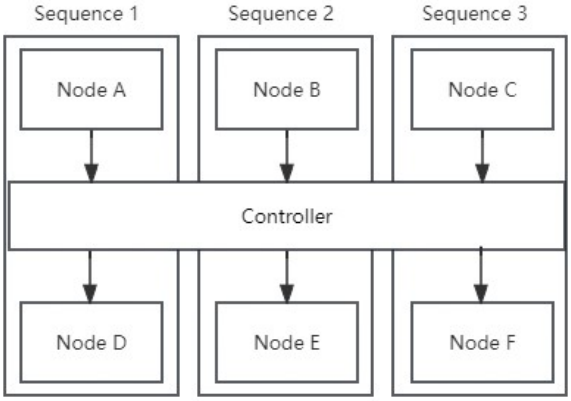

Figure 5. Via controller experimental configuration

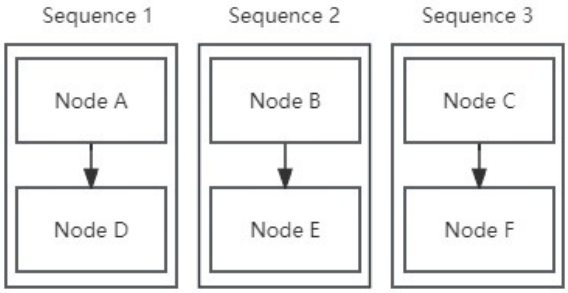

Figure 6. Peer-to-peer experimental configuration

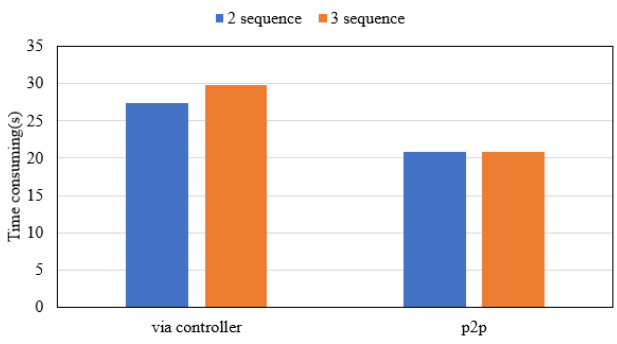

Figure 7. Transmission time of via Controller and peer-to-peer

Test results show that the data transferring time increases when the number of sequences is increased in the model via controller. In contrast, in the peer-to-peer communication, the increase in the number of sequences does not make a difference in the transmission time. In addition, by observing the transmission time, we can see that when the sequence is greater than or equal to 2, the transmission time of the peer-to-peer mode is smaller than that of the via-controller mode.

5. Conclusion

Peer-to-peer communication reduces the amount of data going through the controller and communication time between nodes is reduced because communication is not blocked by the controller performance.

This experiment was conducted on Raspberry Pi andsimulates a serverless platform executing multiple high-throughput sequences at the same time. This scenario of high-throughput computation is similar to certain computational tasks that require a large number of input parameters such as machine learning for large language models. This places great demands on the data transfer capabilities of serverless platforms. The performance representation of experiment was affected by the lack of hardware performance. It was not possible to set the number of threads in the program, nor to set different quantities of function nodes, nor different quantities of sequences (pipelines).

The next experiment will be implemented on the MEC-RM Platform. The experiment will compare the performance of this peer-to-peer model in the MEC-RM Platform with the traditional model on the OpenWhisk platform by testing in the realistic usage scenarios. The MEC-RM Worker client is still in development and then two modes (via controller and peer-to-peer communication) will be tested in the real environment of the MEC-RM Platform. The experiment will be conducted by changing the above unfinished environment settings.

Reference

[1] Red Hat. (2020.01.03). “What is Function-as-a-Service (FaaS)?”. https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas

[2] IBM. What is FaaS (Function-as-a-Service)? https://www.ibm.com/topics/faas

[3] Wagner, Brandon, and Arun Sood. "Economics of resilient cloud services." 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2016.

[4] Villamizar M, Garces O, Ochoa L, et al. Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures[C]//2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2016: 179-182.

[5] Apache/openwhisk (Last edited:2023.7.13). OpenWhisk Actions. github. https://github.com/apache/openwhisk/blob/master/docs/actions.md#creating-action-sequences

[6] Roberto Bandini. (2021.4.18). An OpenWhisk sequence example. Blog. https://www.robertobandini.it/2021/04/18/an-openwhisk-sequence-example/

[7] Apache/openwhisk (Last edited:2022.8.1). SequenceActions.scala. https://github.com/apache/openwhisk/blob/6f11d48b216a01b7c6fa3342d2b8b972109106f7/core/controller/src/main/scala/org/apache/openwhisk/core/controller/actions/SequenceActions.scala#L350

[8] Apache／openwhisk, (Last edited:2023.1.26). What happens on an invocation. Github. https://github.com/apache/openwhisk/tree/master

ⓒ 2023 Information Processing Society of Japan