Understanding Open Source Serverless Platforms: Design Considerations and Performance

Mr. Jeyhun Abbasov $UniversityofTartuComputerScience$

***Abstract*— Serverless computing is becoming more prevalent every day. The reason for that, this technology guarantees a lower cost and suitable development environment without focusing on server management. As a consequence, there are many proprietary and open-source serverless computing platforms. In this paper, we will figure out how the performance of open-source serverless computing platforms depends upon the design issues of the platform. We will look at how the throughput and latency metrics differ on different platforms and how the auto-scaling works, either resource-based or workload-based.**

I. INTRODUCTION

Serverless computing is almost indispensable for cloud application deployment in application architectures to containers and microservices. A serverless computing platform manages every step (lifespan, executions, scalability) of cloud applications without knowing prior knowledge of server management.

Serverless computing offers a cloud provider an extra chance to manage the complete development stack, cut operating costs through effective cloud resource optimization and administration, provide a platform that stimulates the adoption of other services in their ecosystem, and reduce the work necessary to design and operate cloud-scale applications.

Serverless computing is an industry phrase that describes a programming style and architecture in which short code snippets are performed in the cloud with no control over the resources on which the code runs. It is not an indicator that there are no servers; rather, the developer should leave most operational issues to the cloud provider, such as resource supply, monitoring, maintenance, scalability, and fault-tolerance.

A serverless paradigm is a programming model that is "stripped down" and based on stateless functions. Unlike PaaS, developers are not restricted to employ a prepackaged application and may create any code. Function-as-a-Service refers to the version of serverless that explicitly employs functions as the deployment unit (FaaS).

There are many proprietary and open-source serverless computing platforms. The AWS Lambda, CloudFlareWorkers, Google Cloud Functions, IBM Cloud Functions, Microsoft Azure Functions, and OracleFunctions are a few cloud service providers that offer serverless computing [2]. Nuclio, OpenFaaS, Knative, and Kubeless are among the most popular open-source serverless platforms we will investigate in this paper.

We fill find the answers for the following research questions:

- RQ 1. What are the architectural components used to design these systems?
- RQ 2. How do throughput and latency behaviors on various types' workloads?
- RQ 3. How do auto-scaling work?

II. BACKGROUND

Understanding Open Source Serverless Platforms that was published in 2019 by [J. Li, et al. 2019] is one of the main articles used in this paper. Their main focus was to investigate how the performance of serverless computing depends on a number of design issues using several popular open-source serverless platforms.

In the article, the authors did many assessments on several cloud serverless platforms (AWS Lambda, Microsoft Azure, and Google Cloud) and discovered that AWS was superior in terms of performance, scalability, and cold-start latency.

Furthermore, the works [5, 12] analyze the many elements that influence the performance of AWS lambda, namely the impact of the function's choice of language and memory footprint.

Finally, work [7] assesses the performance of the Fission, Kubeless, and OpenFaaS serverless frameworks, characterizing response time and the ratio of successfully completed requests for various loads.

III. CHARACTERISTICS

We will investigate the characteristics of Nuclio, OpenFaas, Knative and Kubeless open-source serverless platforms in this section.

a) Nuclio Serverless Platform

Nuclio Serverless Platform has unique architecture that is called 'Processor' architecture. The architecture enables work parallelism by allowing many worker processes to execute in each container. Another interesting functionality of Nuclio Serverless Platform is that the 'function' pod can be triggered directly from an external client. The function pod has the following processes:

- Event-Listener
- User Deployed Function

The Figure 1 shows the components of the Nuclio Serverless Platform:

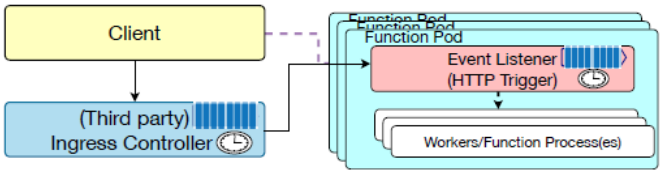


Figure 1: Nuclio Serverless Platform [1]

b) OpenFaas Serverless Platform

OpenFaas Serverless Platform function pod has the following processes:

- Of-Watchdog - entry-point for requests that written in Go
- User Deployed Function

The Figure 2 shows the components of the OpenFaas Serverless Platform:

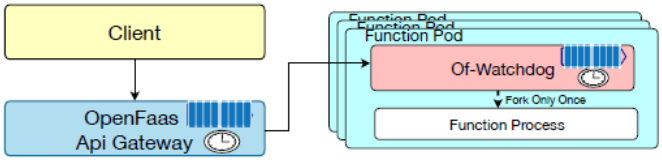
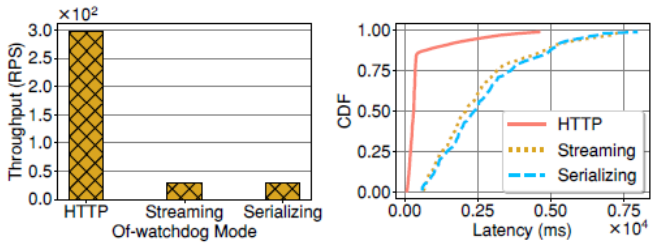


Figure 2: OpenFaas Serverless Platform [1]

The 'of-watchdog' has the following modes:

- HTTP - a function is only forked once at the start
- Streaming - a function is forged for every request
- Serializing - a function is forged for every request

The Figure 3 shows throughput and latency of different modes:



(a) Throughput in requests/second. (b) Latency in ms.

Figure 3: Throughput and Latency of different modes in OpenFaas [1]

In this paper, the HTTP mode of 'of-watchdog' is choosen for investigation of OpenFaas.

c) Knative Serverless Platform

Knative Serverless Platform function pod contains the following processes:

- Queue Proxy - queues incoming requests and forwarding them to execution
- User Deployed Function

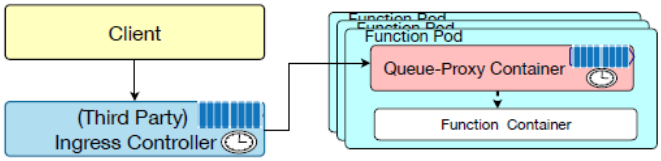


Figure 4: Knative Serverless Platform [1]

The Queue Proxy container immediately handles requests that are coming from the ingress controller. This approach has better throughput and latency rather than Nuclio and OpenFaaS approaches.

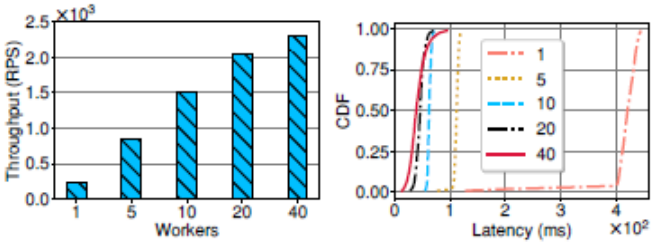


Figure 5: Throughput and latency for different number of workers within one Nuclio function pod

d) Kubeless Serverless Platform

The function pod of Kubeless Serverless Platform consists only user deployed function. This design has its own benefits such as gaining better performance in some conditions. The components of Kubeless Serverless Platform are shown in Figure 6:

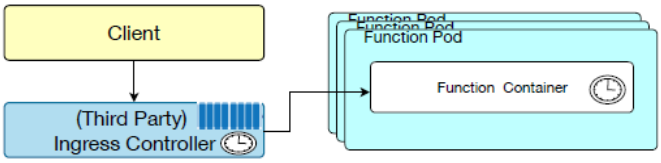


Figure 6: Kubeless Serverless Platform [1]

Impact of Ingress Controller and API Gateway

The API Gateway/Ingress controllers route the coming requests as follows: [13]

- Route the traffic, and manage the traffic load-balancing. For example, in OpenFaaS.
- Route the traffic immediately to the active pods. For example, in Knative.

We noticed that numerous connections are established with the service in the beginning and that these connections are subsequently used to redirect incoming requests (in OpenFaaS API Gateway). The scenarios are as follows:

- new connections are not created when the connections are not closed
- to the new pods, the traffic cannot be generated when the connections are not established after auto-scaling.

In the last condition, the controller keeps track of all active pods' health checks and status.

IV. EVALUATION

That section will investigate the impact of serverless platforms' design architecture and how they depend upon Kubernetes. Additionally, we will cover how auto-scaling capabilities distinguish among the serverless platforms.

3.1 Experimental setup [1]

The parameters of test environments (1 master and 2 worker nodes) are as follows:

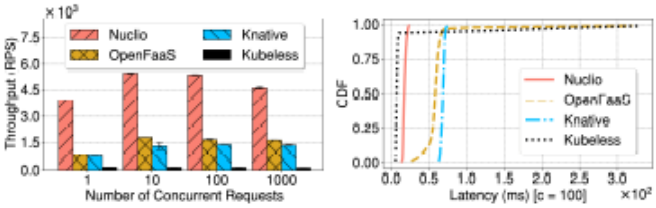
- CPU - Intel CPU E5-2640v4@2.4GHz with 10 physical cores
- OS - Ubuntu 16.04.1 LTS with kernel 4.4.0-154-generic
- Base Platform - Kubernetes v1.15.3
- Programming Language - Python 3.6

A simple function that returns 'HelloWorld' as a response is used for testing purposes and the **wrk** tool [7] is used for generating the HTTP workloads and invoke the serverless functions.

3.2 Performance - Throughput and Latency

3.2.1 Baseline Performance

A primary function that produces 4 bytes of static text ('Hello-world') as a response is used to evaluate the baseline performance of each serverless platform, such as throughput and latency. We set the same queue size and a single function pod to each serverless platform to achieve the same comparison parameters. Here, timeout parameters are 50K requests and 10s timeout in API Gateway components. Furthermore, we limited Nuclio to a single-worker process.

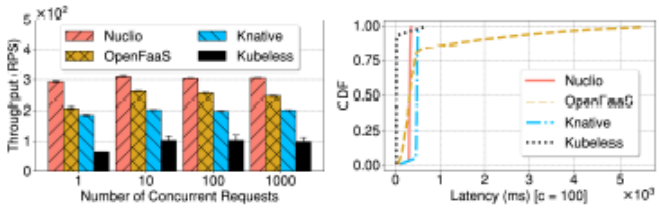


(a) Avg. throughput. (b) Latency for concurrency of 100.

Figure 7: Throughput and latency of 'hello-world' function. [1]

3.2.2 HTTP Workload

Following baseline performance testing, we proceed to http-workload performance testing. The testing settings remain the same as they were in the baseline trial. We discovered that Nuclio has the lowest latency in that trial (99% percent within 500ms). OpenFaaS and Knative, on the other hand, queue requests at the ingress/gateway components. OpenFaaS has a lengthy tail due to waiting at the gateway and watchdog components, with specific timeout limits. We discovered that the Kubeless loses connections at the API Controller while tested. As a result, it caused additional retries by the client and decreased throughput and latency.



(a) Avg. throughput (b) Latency for concurrency of 100.

Figure 8: Throughput and latency of 'http-workload' function across different serverless platforms at different concurrency levels. [1]

3.2.3 Variable Payload Size

First of all, we increase the payload size in response in order to test the data transfer difficulties of serverless platforms. Additionally, how was building, packing affected are investigated in this experiment. It is clear that from Figure 9, Nuclio is better for payload sizes that less than 1KB. However, OpenFaaS and Knative are better for large payloads.

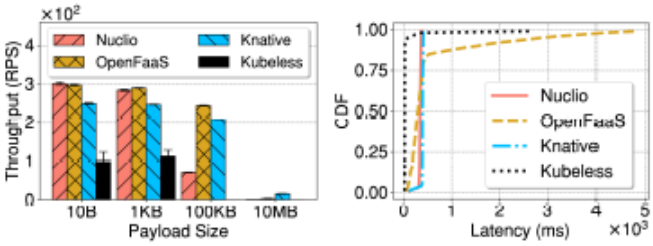


Figure 9: Throughput and latency of 'http-workload' function with different payload sizes for different serverless platforms. [1]

3.3 Auto-scaling

The serverless platforms that we are investigating support auto-scaling features. We experimented with the auto-scaling features of each serverless platform under different workload conditions. Knative and OpenFaaS have rate-based and Kubernetes-based horizontal pod-autoscaler (HPA) modes. Firstly, we set the configuration parameters of auto-scaling features on both platforms to have the exact interval for auto-scaling for a fair comparison. The autoscale feature is based on the average rate of incoming requests in OpenFaaS. However, the feature is based on the concurrency level in Knative. The same python function is used in that experiment. We will look at workload based and resource based auto-scaling in that section.

3.3.1 Workload based auto-scaling

1. Steady workload

We utilize the wrk tool, establish a constant rate for outstanding requests (100 concurrency), and run the experiment for 60 seconds. We also compel the OpenFaaS gateway to terminate and restart connections with the function pods to ensure correct traffic allocation among freshly generated pods. We check the amount of pod instances, CPU and memory utilization, and throughput every 2 seconds. According to the Figure 10, Knative scales several instances at a time to reach the maximum of ten (10) instances rapidly (in 12s), but OpenFaaS only scales one instance at a time, requiring 26s to reach the maximum of ten (10) instances. Although the CPU use for the scaled instances appears to be the same, Knative has a larger memory strain. This is due to the differences in python runtimes and overheads in the queue-proxy container component for Knative and the of-watchdog components in OpenFaaS.

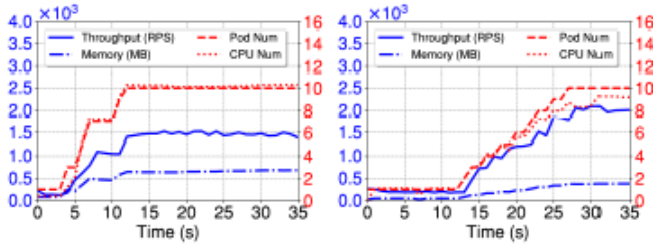


Figure 10: Auto-scaling with steady workload. [1]

2. Bursty workload

We also tried adjusting the http workload so that there were bursts of concurrent requests followed by significant periods of idle time. According to the Figure 11, Knative is more responsive to bursts and can scale quickly to a large number of instances, but OpenFaaS scales slowly and has a lower average throughput.

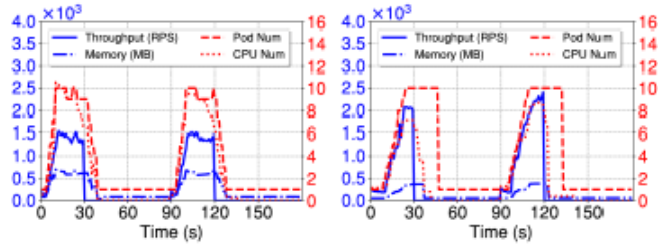


Figure 11: Auto-scaling with bursty workload.

3.3.2 Resource based auto-scaling

We utilize the same configuration (stationary state), set the CPU consumption limit to 50%, and auto-scale using Kubernetes HPA. It should be mentioned that function pod auto-scaling is exclusively the responsibility of Kubernetes. With the exception of Kubeless, the auto-scaling behavior is constant across all platforms, as seen in the Figure 12. In other words, auto-scaling seeks to double the number of instances at each stage until the maximum number is reached (10). However, the time of each step is defined by the CPU consumption factor, which is dictated by the serverless platform's specific components

(event-listener, of-watchdog, queue-proxy). Despite being more CPU intensive, Nuclio can scale quicker (in 40s) than Knative and OpenFaaS..

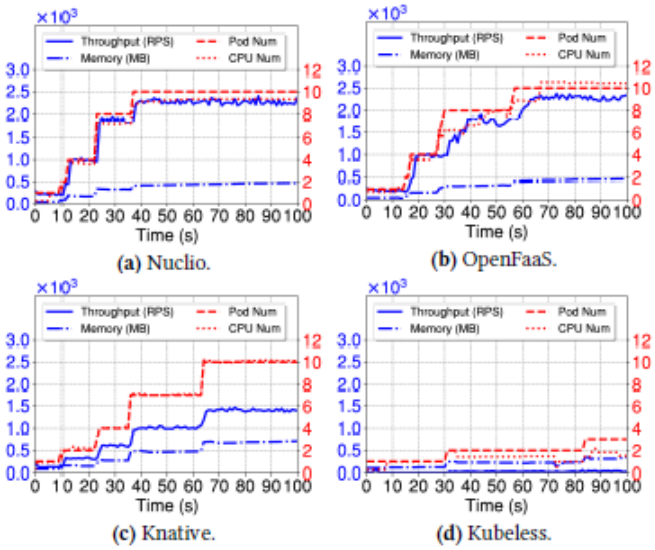


Figure 12: Auto-scaling issues with Knative and OpenFaaS. [1]

V. CONCLUSIONS

In this paper, we found out answers the research questions that what are the architectural components used to design these systems, how do throughput and latency behaviors on various types workloads, how do auto-scaling work. We investigated the open-source serverless plaforms using various measurements. First of all, we experimented the baseline performance of the open-source serverless plaforms. Then, we looked at how the throughput and latency changes when HTTP Workload and variable payload sizes. In addition, we experimented auto-scaling features of that open-source serverless plaforms, both workload based auto-scaling and resource based auto-scaling. Finally, we founnd out how steady workload and bursty workload is differs in each open-source serverless platforms.

- RQ 1. What are the architectural components used to design these systems?
  - (Third Party) Ingress Controller or API Gateway
  - Event listener or Queue-Proxy Container or Of-Watchdog or Nothing
  - (Function Container) User Defined Function
- RQ 2. How do throughput and latency behaviors on various types' workloads?
  - Nuclio has the least 99% latency within 500ms in HTTPWorkload.
  - Nuclio performs better for small payload sizes (i.e., less than 1KB), while OpenFaaS and Knative perform better for large payloads.
- RQ 3. How do auto-scaling work?
  - When it comes to workload based auto-scaling, the Knative scales multiple instances at a time to reach the max, while OpenFaaS just scales up one instance at a time to scale up to the max.
  - When it comes to resource based auto-scaling, Nuclio is more CPU hungry in order to able to scale more rapidly, than Knative and OpenFaaS.

REFERENCES

[1] Junfeng Li, Sameer G. Kulkarni, K. K. Ramakrishnan, Dan Li. Understanding Open Source Serverless Platforms: Design Considerations and Performance. https://dl.acm.org/doi/10.1145/3366623.3368139

[2] S3 Simple Storage Service. URL https://aws.amazon.com/s3/. Online; accessed December 1, 2016

[3] Bienko, C.D., Greenstein, M., Holt, S.E., Phillips, R.T.: IBM Cloudant: Database as a Service Advanced Topics. IBM Redbooks (2015)

[4] OpenFog Consortium. URL http://www.openfogconsortium.org/. Online; accessed December 1, 2016

[5] OpenFog Consortium. URL http://www.openfogconsortium.org/. Online; accessed December 1, 2016

[6] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Serverless computation with openlambda. 2016. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/hendrickson

[7] 2018. wrk: a HTTP benchmarking tool. https://github.com/wg/wrk. [online].

[8] Amazon. 2019. AWS Lambda. https://aws.amazon.com/lambda.

[9] Brendan Burns and et al. 2016. Borg, Omega, and Kubernetes. Commun. ACM 59, 5 (2016), 50–57.

[10] Wes Lloyd and et al. 2018. Serverless computing: An investigation of factors influencing microservice performance. In 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 159–169.

[11] Garrett McGrath and Paul R Brenner. 2017. Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 405–410.

[12] S. K. Mohanty, G. Premsankar, and M. di Francesco. 2018. An Evaluation of Open Source Serverless Computing Frameworks. In 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). 115–120.

[13] Andrei Palade, Aqeel Kazmi, and Siobhán Clarke. 2019. An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge. In 2019 IEEE World Congress on Services (SERVICES), Vol. 2642. IEEE, 206–211.