EMARS: Efficient Management and Allocation of Resources in Serverless

Aakanksha Saha, Aisha Syed, Jacobus Van der Merwe, Sonika Jindal

University of Utah

Abstract: The latest serverless solutions are really server-hidden and built to host functions and hide that how the functions runs on servers or how scaling is done. Unlike SaaS or PaaS that are always running, but scale on-demand, serverless workloads run on-demand, and consequently, scale on-demand. There are mutual economic pressure as Cloud providers need to cost-compete by using datacenters resources more efficiently while Cloud customers seek to reduce cost by minimizing resource wastage. Both can be satisfied by better matching of application needs to allocated services. In this paper, we explore and compare various serverless platforms in terms of latencies incurred by them because of resource allocation. We analyse various serverless platforms and propose benchmark for performance measurements.

1 Introduction

Serverless cloud computing is a paradigm in which the cloud providers dynamically provisions machine resources for the user applications to run. And in turn, the user pays only for the compute resources utilized by their application. The applications are broken into modular functions which are stateless, event-driven and short lived.

The serverless architecture comes in useful for cases like bursty workloads where demand is not known upfront. Such scenarios require quick expansion and shrinking of resources. The need can be achieved if the functions can be spawned quickly in containers. To further reduce the invocation latencies, there is a need to efficiently manage the containers such that the startup time ($\approx$ 100ms) of containers can be avoided.

The following are the key challenges in a serverless platform concerning resource management:

1. *Cost:* Minimizing resource usage by serverless function, both when it is executing and when idle.

2. *Cold start:* A key differentiator of serverless is the ability to scale to zero, or not charging customers for idle time. Scaling to zero, however, leads to the problem of cold starts, and paying the penalty of getting serverless code ready to run. Techniques to minimize the cold start problem while still scaling to zero are critical.

3. *Resource limits:* Resource limits are needed to ensure that the platform can handle load spikes, and manage attacks. Enforceable resource limits on a serverless function include memory, execution time, bandwidth, and CPU usage. In additional, there are aggregate resource

4. *Scaling:* The platform must ensure the scalability and elasticity of users functions. This includes pro-actively provisioning resources in response to load, and in anticipation of future load.

Proactive allocation of resources remains a challenging problem. One of the solutions could be to train the system for predicting the requirements

based upon the data gathered. The useful information can be in the form of type of the requests received for certain period, type of resources needed to handle the request etc. In a nutshell, resource management can be done based upon the container type and application type by appropriately allocating the system resources.

In this paper, we make the following contributions:

1. Analysed various serverless platforms for latency in function invocations.
2. Build a benchmark for evaluating the performance.
3. Propose EMARS for efficient resource allocation.

2 Implementation

1. Current handling of containers in OpenLambda
2. Proposed container allocation in OpenLambda
3. Flow diagram with EMARS approach.

3 Evaluation

1. Test against the benchmarking suite
2. Compare performance with current platforms.

4 Future work

1. Apply machine learning to predict the application behaviour collected from logs and manage resources appropriately.
2. Strong isolation of containers running on the shared platform to enhance security.

5 Conclusion

In this work we present an analysis of resource allocation done inside various serverless platforms. We also present the effect of resource allocation techniques used in some open source serverless platforms. Further we propose a design based upon OpenLambda for resource allocation as per the learnings from variety of loads.

6 Related work

There have been work done in analysing the latencies due to container allocation [2]. To enable the serverless technology, there are still some gaps to be filled in terms of monitoring and analysis which is done by various tools. There are tools like [5] to monitor the function performance specific to AWS Lambda. OpenLambda [3] compares the response times of lambda functions vs elastic BS virtual machines.

To our knowledge, most of the platforms perform resource scaling using eager pooling and resource shrinking like in OpenWhisk[1] and Fission.io[4]. An initial pool of containers is allocated and based upon the demand the pool is expanded or shrunk.

References

[1] Apache/IBM OpenWhisk. IBM. `http://openwhisk.incubator.apache.org/`.

[2] G, M., and PR, B. Serverless computing: Design, implementation, and performance. `http://www.serverlesscomputing.org/wosc17/#p4`, 2017.

[3] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Serverless computation with openlambda. *HotCloud'16 Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing* (2016), 33–39.

[4] Kubernetes Fission. Serverless Functions as a Service for Kubernetes developed by Platform9. `http://fission.io`.

[5] Pipe, I. See inside your lambda functions. `https://www.iopipe.com`, 2017.