Tackling Cold Start in Serverless Computing with Container Runtime Reusing

Kun Suo[1], Yong Shi[1], Xiaohua Xu[1], Dazhao Cheng[2], and Wei Chen[3]
[1]*Kennesaw State University,*
[2]*University of North Carolina at Charlotte,* [3]*Nvidia*

ABSTRACT

During past few years, serverless computing has changed the paradigm of application development and deployment in the cloud and edge due to its unique advantages, including easy administration, automatic scaling, built-in fault tolerance, etc. Nevertheless, serverless computing is also facing challenges such as long latency due to the cold start. In this paper, we propose HotC, a container-based runtime management framework which leverages the lightweight containers to mitigate the cold start and improve network performance of serverless applications. HotC maintains a live container runtime pool, analyzes the user input or configuration file, and provides available runtime for immediate reuse. Our evaluation results show that HotC introduces negligible overhead and can efficiently improve performance of various applications in both cloud servers and edge devices.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Software and its engineering** → **Software design engineering**; • **Networks** → **Middle boxes / network appliances**;

KEYWORDS

Performance, Container, Serverless, Cold Start.

ACM Reference Format:
Kun Suo[1], Yong Shi[1], Xiaohua Xu[1], Dazhao Cheng[2], and Wei Chen[3] [1]*Kennesaw State University,* [2]*University of North Carolina at Charlotte,* [3]*Nvidia* . 2020. Tackling Cold Start in Serverless Computing with Container Runtime Reusing. In *Workshop on Network Application Integration/CoDesign (NAI'20), August 14, 2020, Virtual Event, USA.* ACM, Porto, Portugal, 2 pages. https://doi.org/10.1145/3405672.3409493

NAI'20, August 14, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8044-7/20/08…$15.00
https://doi.org/10.1145/3405672.3409493

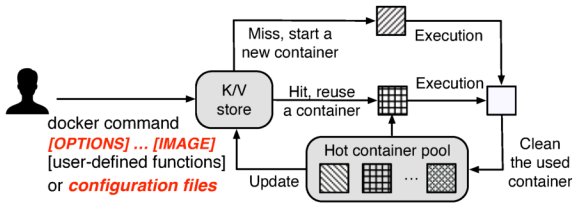

Figure 1: HotC Architecture.

1 INTRODUCTION & BACKGROUND

As the traditional market of cloud computing turns mature and user requirement for microservices keeps growing, serverless computing, such as Amazon Lambda, Microsoft Azure Function and Google Cloud Function, which provides high performance, high scalability, built-in availability and fault tolerance, is becoming increasingly popular in the public clouds [3, 4]. Serverless infrastructure allows developers to focus on the application and business logic itself instead of worrying about where to deploy their codes and how to tweak large number of servers. However, such the design might also introduce performance loss due to the cold start, especially to those I/O-intensive applications. For instance, Amazon has reported that every 100ms of latency costs them 1% in sales [1] and the page speed of websites is also treated by Google as one of the major ranking factors [2].

To address the cold start in serverless services, our key observation is that the runtime could be reused efficiently by leveraging the lightweight containers and the homogeneity of containerized serverless applications. Inspired by that, we proposed and developed HotC, a container-based runtime management framework which provides low-latency request handling while minimizing the performance overhead to applications. Different from existing solutions arbitrarily keeping container alive for certain amount of time (i.e., 15 minutes in AWS Lambda) or periodically waking up containers to keep warm (i.e., Azure Logic), HotC maintains a runtime pool inside the memory and efficiently reuses the containers based on the user requests. It is also transparent

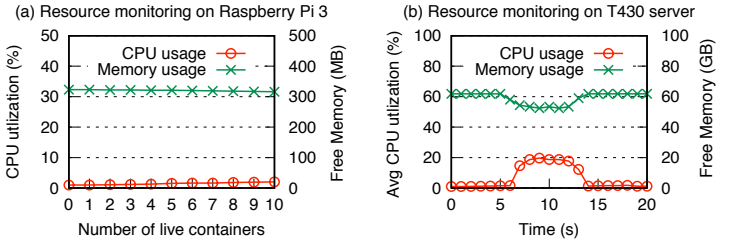

Figure 2: The resource consumption of live containers.

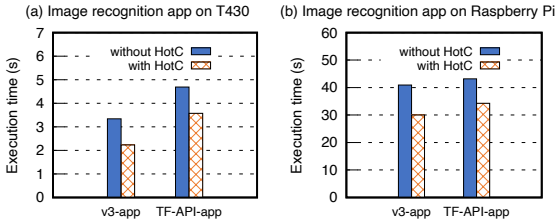

Figure 3: The image recognition application execution time w/o and w/ HotC.

to applications or serverless functions. Our preliminary results show substantial performance improvement of various applications in both cloud servers and edge devices.

2 HOW HOTC WORKS

Figure 1 shows the architecture of HotC. It acts as a middleware between the clients and the backend servers. When new requests come, HotC always attempts to execute the user code in an existing and free container. If it cannot find an available container, HotC just starts a new one as usual. After the container finishes execution, it returns the results back to the client side and HotC will then cleanup the container and prepare for the next request. Such a design has many benefits: First, it is simple and straightforward, which does not involve disruptive changes to the existing architecture. Second, as the resource consumption mainly comes from the application execution instead of container itself, it is lightweight to maintain a group of live containers without introducing much overhead. Lastly, reusing the same container runtime can offer hot cache and less translation lookaside buffer (TLB) flushing, which can significantly improve the resource utilization and application performance.

3 PRELIMINARY RESULTS

Overhead. We first analyzed the overhead of HotC on resource usage. Figure 2 plots the CPU and memory usage monitoring on Raspberry Pi and physical server. First, we varied the number of live containers and measured the resource consumption. As shown in Figure 2(a), the number of live containers does not have an obvious impact on the available resource. The CPU usage increased by less 1% (ten live containers) compared to that without containers. Similarly, the memory footprint due to the different number of live containers is also insignificant. For instance, the memory usage increased by 0.7MB for each individual live container. The majority of resource consumption comes from the applications instead of the container itself, which left immense potential to keep live containerized runtime to address the cold start latency. We also measured the resource change during a containerized application lifecycle. As shown in Figure 2(b), we started a *Cassandra* database in one container at 6_{th} second to handle some user requests and then stopped it at 13_{th} second while keeping the container still live. *Cassandra* database is a heavy workload which executes the database on the Java virtual machine. Compared to the application resource consumption, the cold container overhead cannot be neglected during execution.

Startup and Execution Time. Next, we evaluated the startup time of two image recognition applications with HotC. One was implemented in *Python* and built on Google inception-v3 model, which trained 1000 categories on the ImageNet dataset (denoted as *v3-app*). Another was implemented in *Go* through Tensorflow APIs to perform image recognition (denoted as *TF-API-app*). The version of Tensorflow is 1.13. All the applications were executed inside Docker containers. The results shown were the average of ten runs. As Figure 3(a) shows, the execution time of v3-app and TF-API-app reduced by 33.2% and 23.9% respectively compared to the that without HotC. The performance improvement is due to the efficient reuse of existing container runtime. Similarly, we also evaluated the performance on Raspberry Pi. Compared to the physical servers, Raspberry Pi has more resource constrains and is sensitive to the overhead. Compared to the physical servers, the normal execution time of the same application prolongs more than 10 times inside edge devices and makes the cold start impact less significant among the total execution time. However, as depicted in Figure 3(b), HotC still helped reducing the execution time of v3-app and TF-API-app by 26.6% and 20.6%, respectively.

REFERENCES

[1] 2012. *How one second could cost amazon 1.6 billion in sales.* http://bit.ly/1Beu9Ah.

[2] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. 2017. Improving User Perceived Page Load Times Using Gaze.. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[3] Kun Suo, Yong Zhao, Wei Chen, and Jia Rao. 2018. An Analysis and Empirical Study of Container Networks.. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*.

[4] Kun Suo, Yong Zhao, Wei Chen, and Jia Rao. 2018. vNetTracer: Efficient and Programmable Packet Tracing in Virtualized Networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*.