FaasCache: Keeping Serverless Computing Alive With Greedy-Dual Caching Extended Abstract

Alexander Fuerst, Prateek Sharma
Indiana University Bloomington

1. Motivation: Keep-alive in FaaS

Functions as a Service (FaaS) is an emerging and popular cloud computing model, where applications use cloud resources through user defined "functions" that execute application code [2]. FaaS is being used by different applications such as web services, API services, parallel and scientific computing, and in machine learning pipelines. The execution time of each function is typically short—in the range of a few milliseconds to a few seconds.

While Functions as a Service (also called serverless computing) promises to revolutionize how applications use cloud resources, the tight latency requirement and the wide diversity in function characteristics raises new challenges in resource management for FaaS. Specifically, current FaaS systems suffer from cold-start problems due to the overhead of initializing code and data dependencies before a function can start executing. Our empirical analysis of popular FaaS applications shows that the initialization overhead can as much as 80% of the total running time.

Keeping the functions alive and warm after they have finished execution can alleviate the cold-start overhead. However, keeping the execution environment alive and running, instead of immediately terminating it, has some drawbacks. Keeping a container or a VM alive consumes computing resources on the physical servers (such as memory). Keep-alive algorithms and policies thus need to balance the latency requirements of applications and the resource utilization of FaaS backend servers. In this paper, we focus on how diverse FaaS workloads can be efficiently executed, by developing a new class of resource management techniques that balance this fundamental latency vs. utilization tradeoff.

2. Limitations of the State of the Art

Current clouds and popular FaaS platforms (such as OpenWhisk) use simple keep-alive policies (such as keeping functions warm for a fixed amount of time). However, ideally, keep-alive policies must keep functions alive based on their resource and usage characteristics, which is challenging due to the diversity in FaaS workloads. Primitive/lack of keep-alive policies has resulted in application developers resorting to brute-force techniques such as polling their functions so that they are kept warm. Recent keep-alive policies [3] also do not take into cognizance all function attributes such as their resource footprints and other initialization overheads.

3. Key Insight: Use Caching for Keep-alive

Our primary insight is that the resource management of functions is equivalent to object caching. Keeping a function warm is equivalent to caching an object, and a warm function execution is equivalent to a cache hit. Terminating a function's execution environment means that the subsequent invocation will incur a cold-start penalty, and is thus equivalent to evicting an object from a cache. The objective is to keep functions warm such that the effective function latency is reduced—which is equivalent to caching's goal of reducing object access time. *By mapping keep-alive to the exhaustively studied caching problem, we can leverage principles and techniques from caching, and apply them to serverless computing.*

The caching analogy allows us to use the vast set of caching algorithms and analytical models, and provides a new way to approach resource provisioning for FaaS platforms. We use hit-ratio curves to determine the ideal size of servers required for handling FaaS workloads, and develop a new vertical autoscaling approach that dynamically adapts server size based on the workload characteristics. The dynamic scaling uses proportional control and hit-ratio curves to minimize both the required server resources, and the cold-start overheads.

4. Artifacts

We implement all our keep-alive and provisioning policies in our FaasCache system, which is based on the popular OpenWhisk platform. We evaluate all our techniques with the recently released Azure function trace [3]. We also use a custom discrete-event keep-alive simulator for large-scale workload analysis involving millions of invocations of up to 1000 distinct functions.

5. Key Results and Contributions

The rise of serverless computing and the challenges posed by its heterogeneity, workload diversity, and latency requirements, will require a new class of approaches to FaaS resource management. We argue that the vast collection of algorithms, analytical models, practical optimizations, and hard lessons from one of the most well studied fields in computer science, caching, can be customized to address many of these challenges. While bespoke solutions to serverless resource management will continue to be developed, our intent is to show the natural equivalence of caching and FaaS, and to highlight how naturally and easily caching techniques can be adapted.

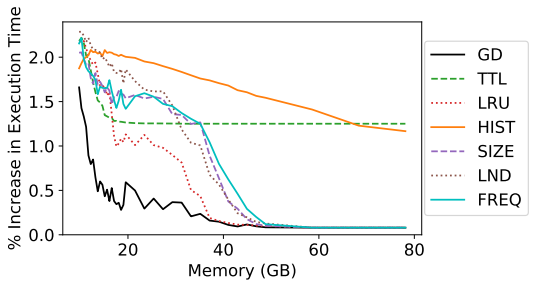

Figure 1: Our caching-based keep-alive policies (such as Greedy-Dual) can reduce the cold-start overhead by more than 3× compared to current (TTL) and state of the art approaches (HIST [3]). The results are shown for a representative sample of the Azure function trace comprising of over 100 million function invocations.

5.1. Greedy-Dual Keep-Alive

Our keep-alive policy is based on Greedy-Dual-Size-Frequency object caching [1], which was designed for caches with objects of different sizes, such as web-proxies and caches. Classical caching policies such as LRU or LFU do not consider object sizes, and thus cannot be completely mapped to the keep-alive problem where the resource footprint of functions is an important characteristic. As we shall show, the Greedy-Dual approach provides a general framework to design and implement keep-alive policies that are cognizant of the frequency and recency of invocations of different functions, their initialization overheads, and sizes (resource footprints).

For each container, we assign a *keep-alive priority*, which is computed based on the frequency of function invocation, its running time, and its size:

$$\text{Priority} = \text{Clock} + \frac{\text{Freq} \times \text{Cost}}{\text{Size}} \quad (1)$$

This allows us to capture the function invocation recency (Clock), the inter-arrival-time (frequency), its memory footprint size, and its cold-start cost. Because this approach considers these different function attributes, it can improve the keep-alive "cache hit rate". We also develop and evaluate specialized versions of Greedy-Dual such as LRU, LFU, etc. We evaluate all keep-alive policies on the Azure function trace [3], as illustrated in Figure 1, which shows a 3× reduction in cold-start overheads with our Greedy-Dual (GD) policy.

5.2. Caching-based Resource Provisioning

The fundamental challenge underlying resource provisioning for FaaS workloads is the performance vs. resource allocation tradeoff. Running a workload on large servers/VMs provides more resources for the keep-alive cache, which reduces the cold-starts and improves the application performance. We develop a *static* provisioning policy that determines the server memory size for a given function workload, that uses hit-ratio curves that are constructed using the notion of reuse-distances.

To handle highly dynamic function workloads, our system also dynamically scales VMs up or down (vertical auto-scaling). Our *dynamic* VM-sizing approach uses hit-ratio curves and a proportional controller, to minimize both the cold-start overhead and the VM size. Our provisioning policies are also evaluated using the Azure function trace, as illustrated in Figure 2, which shows the dynamic server memory adjustment in response to the function workload using our hit-ratio curve and proportional control based method.

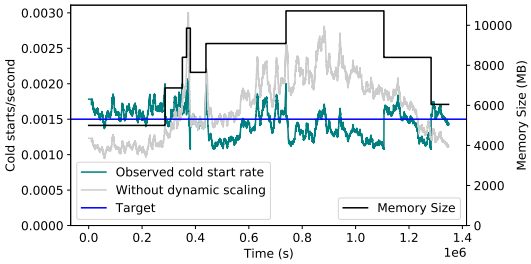

Figure 2: With dynamic cache size adjustment, the cold starts per second are kept within a band, which reduces the average server size by 30%.

Empirical findings. We implement our caching-based techniques in a popular FaaS platform, OpenWhisk, and empirically evaluate our techniques on real-world FaaS workloads.

1. We conduct extensive trace-driven analysis of the tradeoffs of keep-alive techniques under different workload characteristics based on the Azure FaaS traces [3] and popular FaaS applications [**?**]. Our experimental results indicate that caching-based keep-alive can reduce cold-start overheads by 3×, improve application-latency by 6×, and reduce system load to run 2× more functions.
2. Our resource provisioning policies use hit-ratio curves to determine the ideal server configuration (such as memory size) required to handle different function workloads. The proportional-control based dynamic vertical-scaling approach can adjust server resources to reduce the cold-start probability, and reduce the average server size by more than 30%.

6. Why ASPLOS

This paper combines the most cross-cutting area in computer systems, caching, with resource management for Functions as a Service clouds. We use caching techniques typically used in storage systems and CDNs. Our unique use of caching for keep-alive will interest a broad swathe of the caching experts in all different systems areas. We also combine serverless computing with virtual machine overcommittment.

7. Citation for Most Influential Paper Award

This paper presents a surprising equivalence between Functions as a Service and caching, and lays the groundwork for a new family of principled keep-alive techniques. The general, caching-based keep-alive and resource provisioning techniques developed in this paper strengthen the theoretical and practical foundations of serverless computing performance.

References

[1] Ludmila Cherkasova. Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy. 1998.

[2] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv:1902.03383 [cs]*, February 2019. arXiv: 1902.03383.

[3] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. *arXiv:2003.03423 [cs]*, June 2020. arXiv: 2003.03423.