University of Bamberg

Distributed Systems Group

Master Thesis

in the degree programme International Software Systems Science
at the Faculty of Information Systems and Applied Computer Sciences,
University of Bamberg

Topic:

Model Based Pre - Warming to Mitigate the Cold Start of Serverless Function

Author:
Tasfia Sharmin

Reviewer:
Prof. Dr. Guido Wirtz

Date of submission:
04.03.2023

Contents

List of Figures

List of Tables

Listings

Abbreviations

SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
AWS	Amazon Web Services
JVM	Java virtual machine

1 Introduction

In today's world of cloud computing serverless computing has become quite a phenomenon as it has taken the responsibility of the underlying infrastructure where the applications will run on and the developers can solely focus on the coding aspect. The term serverless has been coined up to portray that the developers do not have to worry about anything related to servers.

As serverless has taken away all the managerial aspects from the developers but unfortunately, it comes with specific challenges. Among them, 'Cold Start' is a potential problem. Cold Start refers to the idea when a serverless function is invoked for the first time or has remained inactive for a long period and needs some time to be prepared for execution ([MSD$^+$19], [SA20]) The size of the deployed package and the employed programming languages are a couple of the aspects that have an impact on this issue [MEHW18]

In this thesis paper, I will present an overview of the cold start problem in serverless function in particular. Further, I will analyze possible problems when running applications on serverless computing platforms by programming my own application [CIMS19].

2 Theoretical Background

2.1 Cloud Computing

Cloud computing has transcended its initial emergence decades ago to become a foundational pillar of modern technological advancement. It has brought revolution by ensuring flexibility, scalability, and cost-effectiveness. Through the ingenious power of virtualization, cloud computing has pioneered a model where both software and hardware resources are delivered on demand, precisely mirroring the user's requirements [Cho15]. The cloud's inherent global reach enables seamless access to applications and files from any device, anywhere in the world, fostering enhanced collaboration and productivity[MG11].

Cloud computing offers diverse service models to cater to a range of technical capabilities and project demands. **Software as a Service (SaaS)**: SaaS offers readily available, pre-configured software applications accessible through a web browser [OAAAGW18, HBS21]. **Platform as a Service (PaaS)**: PaaS offers a development environment with programming tools and resources to the users where they can leverage this platform to build custom applications without managing the underlying infrastructure [OAAAGW18, HBS21]. **Infrastructure as a Service (IaaS)**: IaaS grants users fine-grained control over computing resources such as servers, storage, and network components to build applications [OAAAGW18, HBS21].

2.2 Serverless Computing

The relentless evolution of cloud computing ushers in a new era with the captivating emergence of serverless computing. This novel paradigm fundamentally reshapes the existing model by shifting the focus from infrastructure management to code execution. This platform abstracts away the underlying server complexities, empowering developers to concentrate on crafting the business logic of their applications [BCC^{+}17, SA20].

serverless computing provides a cost-effective, scalable, and elastic platform. It offers the scaling of resources that adapts to the fluctuating demands without any manual intervention. This results in cost savings as the developers only pay for the resources the application requires and eliminates the need for upfront investments in large-scale infrastructure [CIMS19].

Additionally, this presents a valuable business model for cloud providers, optimizing the utilization of resources typically underutilized in long-running workloads. Through serverless-based resource allocation, cloud providers can enhance the efficiency of their spare capacity, making more effective use of previously underused resources [TCCBR21].

2.2.1 Capabilities

Auto-scaling: Serverless applications adapt to workload fluctuations, scaling up or down automatically. This eradicates the necessity of overseeing individual server instances, preventing unnecessary costs during idle periods.

Flexible Scheduling: Applications gain freedom from specific server dependencies. Serverless controllers dynamically schedule them across the cluster, ensuring an even distribution for optimal performance.

Event-driven Execution: Functions respond to specific events, such as API calls or data changes, removing the need for continuously running servers and minimizing resource consumption.

Transparent Development: Developers concentrate on coding without managing infrastructure. Cloud providers take care of server operations, runtime environments, and resource distribution.

Pay-as-you-go: Costs are incurred only for the actual resources utilized by functions, promoting cost efficiency and eliminating the requirement for upfront investments in servers.

2.3 Serverless Function

Serverless computing revolves around the concept of serverless functions, although the term itself can be somewhat misleading as servers are still integral to the process. Despite this, "serverless" signifies that servers are abstracted, allowing developers to concentrate on business logic without delving into the underlying infrastructure[WW21].

Serverless execution environment where you can upload small pieces of code (functions) that is responsible for a single task that don't need to run continuously. Instead, they wait for specific events (e.g., API requests, database changes) to trigger their execution. This eliminates the need for managing servers and allows for highly scalable and cost-effective deployments.

Amazon Web Services (AWS) Lambda pioneered serverless computing, establishing key dimensions like cost, programming model, deployment, resource limits, security, and monitoring. It supports languages such as Node.js, Java, Python, and C# [1]. While early versions had limited composability, recent improvements have addressed this issue. AWS Lambda seamlessly integrates with various AWS services, facilitating the use of Lambda functions as event handlers and for composing services [BCC$^+$17].

Google Cloud Functions was released in 2016 and offers serverless functions in Node.js, Python, Go, Ruby, or Java [2] in response to HTTP calls or events from specific Google Cloud services. Although functionality is currently evolving, future versions are expected to expand its capabilities.s [BCC$^+$17].

[1] `https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html`
[2] `https://cloud.google.com/functions/docs/runtime-support`

Microsoft Azure Functions supports HTTP webhooks and integrates with Azure services to execute user-provided functions. The platform accommodates multiple languages, including C#,Java, Javascript, Python, Typescript and any executable [3]. The runtime code is open-source under the MIT License, and for streamlined development, the Azure Functions CLI provides a local environment for creating, testing, and debugging [BCC+17].

IBM OpenWhisk focuses on event-driven serverless programming, allowing the chaining of serverless functions to create composite functions. It supports languages like Go, Javascript, Java, Swift, Python, and arbitrary binaries within a Docker container [4]. OpenWhisk, available on GitHub under the Apache open-source license, incorporates additional components for crucial functionalities like security, logging, and monitoring [BCC+17].

2.3.1 Challenges

Cold Start: When a function hasn't been active for a while, it experiences a "cold start," taking time to initialize and impacting performance.

Vendor Lock-in: Embracing a specific platform might tie you to their ecosystem. Switching providers can be complex and costly, especially for intricate applications.

Limited Customization: Unlike traditional deployments, you often have less control over the execution environment and runtime configurations. This might restrict optimization opportunities.

Debugging Challenges: Debugging serverless functions can be trickier than traditional applications due to distributed execution and ephemeral nature. Specialized tools and approaches are often required.

Monitoring Complexity: Monitoring performance and resource usage are needed in this domain as the executed function are short lived so additional tools are required that might be complex.

Security Concerns: Multi-tenancy in serverless platforms necessitates robust security measures. Understanding provider practices and implementing proper security checks is essential.

Limited Resource Visibility: Accessing detailed resource usage data for individual functions might be limited compared to traditional deployments. This can hinder cost analysis and optimization.

Development Learning Curve: Shifting to a serverless mindset and mastering serverless development patterns requires learning new concepts and tools.

[3]`https://learn.microsoft.com/en-us/azure/azure-functions/functions-versions`
[4]`https://openwhisk.apache.org/documentation.html`

2.4 Cold Start

Serverless computing has revolutionized application development, offering remarkable scalability, agility, and cost savings. However, one persistent hurdle threatens to impede its seamless performance: cold starts. For a user uploading a fresh serverless function, the first request can be met with a chilling reality. Unlike traditional deployments with readily available resources, serverless platforms don't allocate resources to idle functions. So, upon the first request, the platform scrambles, allocating resources and deploying a new instance to handle the task, leading to a delay known as a cold start. This initial latency can significantly impact response times and user experience[LYYO21].

Furthermore, even functions experiencing frequent requests aren't entirely immune. When traffic surges, serverless platforms auto-scale by deploying new function instances. From the perspective of these fresh copies, the first request encounters a cold start, potentially impacting performance during sudden spikes [LYYO21].

Therefore, while cold starts represent the necessary flip side of serverless advantages, their potential impact demands careful consideration. By acknowledging this challenge and implementing mitigation strategies, developers can ensure their serverless applications remain warm and responsive, delivering a consistently positive user experience.

2.5 Contributing Factors of Cold Start

Programming Language Impact: Compiled languages like Java and C# exhibit higher cold start overhead due to the additional setup required for their respective runtime environments (e.g., **Java virtual machine (JVM)**). This overhead originates from the necessity to initialize and load the runtime prior to function execution. Interpreted languages such as JavaScript, on the other hand, generally bypass this extra step, potentially leading to faster startup times[MEHW18].

Deployment Package Size: Larger deployment packages directly translate to increased cold start overhead. The process of copying, loading, unpacking, and subsequently executing the function image is proportional to its size, resulting in a time-consuming bottleneck for bulky packages. Smaller packages necessitate fewer resources and processing time before actual function execution, streamlining the startup process [MEHW18].

Memory/CPU Resource Influence: Cold start overhead diminishes with increasing memory and CPU allocations. This can be attributed to the enhanced ability of the container to handle the initial loading and setup tasks more efficiently, particularly when CPU utilization remains high at lower memory settings. Conversely, at higher memory settings where the CPU becomes underutilized, the potential benefits of resource scaling might be negligible. It's important to note that this specifically focuses on the scenario where low memory settings coincide with significant CPU usage [MEHW18].

ohers will be added later

2.6 Current Research on Solving Cold Start Problem

3 Comparison of Cloud Service Providers

4 Conclusion

This is the last chapter of this thesis.

References

[BCC+17] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. *Serverless Computing: Current Trends and Open Problems*, pages 1–20. Springer Singapore, Singapore, 2017.

[Cho15] David C. Chou. Cloud computing: A value creation model. *Computer Standards & Interfaces*, 38:72–77, 2015.

[CIMS19] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Commun. ACM*, 62(12):44–54, nov 2019.

[HBS21] Hassan Hassan, Saman Barakat, and Qusay Sarhan. Survey on serverless computing. *Journal of Cloud Computing*, 10, 07 2021.

[LYYO21] Seungjun Lee, Daegun Yoon, Sangho Yeo, and Sangyoon Oh. Mitigating cold start problem in serverless computing with function fusion. *Sensors*, 21:8416, 12 2021.

[MEHW18] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. 10 2018.

[MG11] P. M. Mell and T. Grance. The nist definition of cloud computing, 2011.

[MSD+19] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. Agile cold starts for scalable serverless. In *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'19, page 21, USA, 2019. USENIX Association.

[OAAAGW18] Isaac Odun-Ayo, M Ananya, Frank Agono, and Rowland Goddy-Worlu. Cloud computing architecture: A critical analysis. In *2018 18th international conference on computational science and applications (ICCSA)*, pages 1–7. IEEE, 2018.

[SA20] Khondokar Solaiman and Muhammad Abdullah Adnan. Wlec: A not so cold architecture to mitigate cold start problem in serverless computing. 04 2020.

[TCCBR21] Rafael Tolosana-Calasanz, Gabriel Castañé, José Bañares, and Omer Rana. *Modelling Serverless Function Behaviours*, pages 109–122. 12 2021.

[WW21] Stefan Winzinger and Guido Wirtz. Data flow testing of serverless functions. 04 2021.

In accordance with § 9 Para. 12 APO, I hereby declare that I wrote the preceding master's thesis independently and did not use any sources or aids other than those indicated. Furthermore, I declare that the digital version corresponds without exception in content and wording to the printed copy of the master's thesis and that I am aware that this digital version may be subjected to a software-aided, anonymised plagiarism inspection.

Bamberg, 04.03.2024 Tasfia Sharmin