A Review of Resource Reusing Paradigm in Serverless Computing System

1st Donghyeon Kim
Department of Computer Science and Engineering
Chung-Ang University
Seoul, South Korea
dhkim@cslab.cau.ac.kr

2nd Mingyu Jo
Department of Computer Science and Engineering
Chung-Ang University
Seoul, South Korea
mgjo@cslab.cau.ac.kr

3rd Dogyun Kim
Department of Computer Science and Engineering
Chung-Ang University
Seoul, South Korea
ehrbs1101@cau.ac.kr

4th Sangoh Park
Department of Computer Science and Engineering
Chung-Ang University
Seoul, South Korea
sopark@cau.ac.kr

***Abstract*—Serverless computing has emerged as a revolutionary paradigm in cloud computing, allowing developers to focus on code development without managing the underlying infrastructure. However, the cold start problem remains a significant challenge and can negate many of the benefits of serverless computing. This paper comprehensively reviews three different reuse techniques, namely container reuse, data caching, and function reuse, which are resource reuse paradigms in serverless computing systems, to mitigate the cold start problem and optimize performance. This review paper emphasizes the importance of resource reuse strategies in improving the efficiency and responsiveness of serverless applications, enabling wider adoption of serverless computing in more areas.**

I. Introduction

Cloud computing has revolutionized the way applications are developed and deployed, and in recent years, serverless computing has emerged as a new paradigm [1] [2] [3] [4] [5]. The evolution of cloud technology can be categorized into two generations. The first generation of cloud technology, which emerged in 2010, relieved the burden of system management and maintenance through server consolidation and centralized data centers. The second generation of cloud technology was focused on further reducing the burden of developing cloud-native applications on programmers and solution architects.

Serverless computing provides developers with high-level software abstractions, such as Functions-as-a-Service (FaaS), deployed transparently, allowing users to remain unaware of the underlying server infrastructure. Modern software engineering methodologies, such as DevOps and continuous integration/continuous delivery (CI/CD) pipelines, have rapidly adopted this model to facilitate the rapid development of cloud-native applications. These methodologies encourage the partitioning of applications into multiple functions that are invoked periodically or in response to events.

However, serverless computing faces a significant challenge: the cold start problem [6] [7]. When a function is activated after a period of inactivity or activated for the first time, it requires significant time to configure and initialize the proper execution environment. This delay significantly increases function execution time and reduces system efficiency. As the number of requests increases and more functions need to be started, the cold start delay also increases, negating many of the benefits of serverless computing. Due to the nature of serverless computing, which executes at the function level, the overall workflow is typically divided into lightweight functions that can be completed within a short time frame. These functions are then registered and utilized accordingly. However, because of this cold start issue, preparing the execution environment takes up most of the total execution time rather than the actual function execution. The delay caused by cold starts degrades the user experience and negatively impacts the system's overall performance.

Various approaches have been proposed to mitigate the cold start problem in serverless computing, including predictive provisioning, utilizing ephemeral storage, intelligent scheduling, traffic prediction, and resource reuse [8] [9]. These techniques aim to minimize cold start latency and reduce the time required for function initialization. Various reuse techniques are being studied in serverless computing [12] [13]. Container reuse recycles previously initialized containers to serve new requests. Task caching stores and reuses the results of previously executed tasks. Data caching keeps frequently accessed data in memory for quick access. Function caching reuses previously compiled function code to save compilation time. Each technique reuses resources at different levels to optimize performance.

This review paper aims to provide a comprehensive overview of the resource reuse paradigm in serverless computing. We aim to provide a comparative analysis of different reuse techniques, evaluate the strengths and weaknesses of each approach, and identify the limitations of current research. In doing so, we hope to provide researchers and practitioners in serverless computing with a unified view of resource reuse and

suggest future research directions. This paper is organized as follows. First, we provide background on serverless computing and the cold start problem. Next, various resource reuse techniques are described in detail and compared. Then, we discuss the limitations and challenges of the current research, and finally, we present future research directions and conclusions.

II. Background

A. Serverless Computing

Serverless computing is a new computing paradigm based on cloud computing that allows developers to develop and run applications without having to manage server infrastructure directly. This paradigm gained popularity following the introduction of AWS Lambda in 2014 and is now an essential part of cloud-native applications.

Serverless computing combines Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS). FaaS allows developers to write and deploy code as individual functions, while BaaS provides backend services such as databases, authentication, push notifications, and more.

Serverless computing has the following key characteristics:

- Event-driven: Functions registered in a serverless system are executed in response to specific triggers, such as HTTP requests, database changes, or scheduled events.
- Automatic scaling: Unlike the traditional Serverfull paradigm, users do not need a load balancer because the serverless platform automatically scales resources up or down based on the number of requests. This ensures reliable performance even during traffic spikes.
- On-Demand billing: Users are charged based on the time of functions run and the functions of the resource use. In the traditional Serverfull paradigm, users pay even when the server is not working, whereas serverless is economical because users do not pay for idle time.
- Stateless: Functions running on serverless systems do not store state by default and must use external storage if necessary. This ensures independence and scalability of the function, but the external storage can introduce overhead.

These features of serverless computing provide several important benefits. It frees developers from managing servers and running infrastructure, allowing them to focus on developing core business logic, significantly improving productivity. In addition, automatic scaling capabilities can effectively respond to traffic fluctuations, making the system more reliable and scalable. The pay-as-you-go billing model provides cost efficiency, especially for volatile workloads, and the deployment of individual functions allows for rapid development iterations and updates. These characteristics allow organizations to respond to market changes more quickly and flexibly.

However, serverless computing does have some drawbacks. One of the biggest is the cold start phenomenon, the delay that occurs when a function is called for the first time or after a long period of inactivity. This can negatively impact the user experience. Complex applications can be composed of many small functions, complicating management and debugging, and function execution timeouts can make them unsuitable for long-running tasks. In addition, the stateless nature can introduce additional complexity to application development that requires state management.

Cold start is one of the most significant technical challenges in serverless computing environments. It is a delay that occurs when a function is called for the first time or after a long period of inactivity. This issue stems from the inherent nature of serverless architectures and directly impacts user experience and application performance.

A multi-step initialization process often causes cold starts:

1) Create a container or execution environment: Create a new isolated environment for running a function. This process uses operating system-level virtualization or container technology and can be time-consuming.
2) Initialize the runtime: Load and initialize the programming language runtime (e.g., Node.js, Python, Java) required to run the function. This process takes particularly longer for heavier runtimes such as Java.
3) Loading function code and dependencies: Loads the actual function code and any necessary libraries, frameworks, and other dependencies into memory. Depending on the size of the function and the complexity of its dependencies, the loading time can vary.
4) Execution of function initialization code: This happens before the actual business logic is executed and performs tasks such as establishing database connections, loading environment variables, and initializing external service clients. Executing this initialization code can also cause delays.

Function execution time depends on a variety of factors. The choice of programming language and runtime has an impact. For example, interpreter languages like Node.js or Python typically start faster than compiled languages like Java or .NET. The complexity and size of the function are also essential factors: the larger and more complex the codebase is and the more external dependencies the function has, the longer cold start time will be. The cloud provider and region can also make a difference in cold start performance; each serverless platform has its own characteristics. Network setup is also essential, especially for functions running inside a VPC, which can require additional time to set up network interfaces and further exacerbate cold start delays.

The problems caused by cold starts significantly impact the overall performance and user experience of serverless applications. The most prominent issue is poor user experience, especially in interactive applications, where long response times can significantly reduce user satisfaction. The lack of consistency in performance can also make it difficult to predict the entire system's performance, as the same function call can have significantly different response times at different times. Cost efficiency can be compromised, as keeping functions warm to reduce cold starts can reduce cost efficiency, a key benefit of serverless. In addition, long cold start times can cause function execution to time out or cause problems

with integration with other connected services, which can affect the stability and reliability of the overall system. These issues can challenge serverless technology for latency-sensitive applications or large enterprise systems.

Cloud providers offer a variety of technical solutions, including provisioning to support concurrency, function snapshots, and runtimes optimized for serverless architectures. Developers are also working to reduce the impact of cold starts by optimizing function design, minimizing dependencies, and warm-up strategies. Addressing or minimizing this issue remains a crucial challenge for the wider adoption and success of serverless technologies and is the subject of ongoing research and innovation in academia and industry.

III. Reusing Paradigms in Serverless Computing

In serverless computing, resource reuse is emerging as a key strategy for optimizing performance and improving cost efficiency by reducing execution delays caused by cold starts. Resource reuse refers to utilizing computing resources once initialized or created across multiple requests or executions. Resource reuse is significant in serverless environments where the execution of functions is transient and stateless in nature.

There are three main approaches to resource reuse. First, container image reuse recycles the image that creates the container, the environment in which functions run. This saves time on container initialization. Second, data reuse is a way to cache data used in previous runs and utilize it in subsequent calls. This can reduce costly operations such as large database queries or calls to external storage. Third, function instance reuse is a strategy for maintaining and recycling function instances that have already been initialized. This minimizes the execution of the function's initialization code, which speeds up response times.

A. Container Image Reusing

Container image reuse has emerged as a key strategy for optimizing performance and solving cold start issues in serverless computing environments. This approach aims to dramatically reduce initialization time by efficiently managing and reusing the container images needed to run functions. Container image tiering allows multiple functions to share a common base layer, which reduces storage and network usage and speeds up image loading times. Caching frequently used images allows them to be loaded quickly on subsequent function calls, which is especially effective for multiple functions that use the same runtime or library.

Several strategies can be utilized to implement container image reuse effectively. Efficiency can be enhanced by creating optimized base images with common dependencies, which can be reused across multiple functions. It is also essential to effectively leverage Docker to reduce build time and use container registries for efficient image storage and deployment. Using multi-stage builds to minimize the final image size and removing unnecessary packages and files to reduce image size are also effective ways to reduce image size.

Efficient container image reuse requires an automated management system. This includes managing image versions, deploying updates, and monitoring usage. An automated system enables consistent image management and rapid updates, even in large serverless environments. Furthermore, developing intelligent systems that use usage pattern analysis to predict and pre-cache frequently used images is a promising area of future research.

Container image reuse strategies have the potential to improve the performance and efficiency of serverless computing significantly. However, their effective implementation requires careful planning and continuous optimization. Important topics for future research include more lightweight container technologies, image optimization techniques specific to serverless environments, and efficient image management strategies in multi-cloud environments.

B. Data Reusing

Data reuse is an important strategy for optimizing performance and streamlining resource usage in serverless computing. By effectively managing and recycling the data that a function uses at runtime, this approach minimizes repetitive data loading and processing and reduces overall execution time. The core idea of data reuse is to retain data that is loaded or generated once in memory or local storage for reuse in subsequent function calls. This is especially effective for computationally expensive data, such as large database queries and files from external storage.

There are many ways to implement data reuse in a serverless environment. One of the most common is in-memory caching, where data is stored in memory and reused. This provides fast access times, but the limitation is that data is lost when the system shuts down and has less capacity than storage. Another approach is to utilize local ephemeral storage. Many serverless platforms provide functions with a limited amount of local storage, which can be utilized to store and reuse data. This method can handle larger data than in-memory caching, but access is relatively slow.

There are a few things to keep in mind when implementing a data reuse strategy. First, it is important to maintain data consistency. If the data being reused does not reflect the latest state, it can lead to incorrect results. Therefore, an appropriate cache invalidation strategy and data refresh mechanism are necessary. Second, careful management of storage usage is essential. Excessive data caching can exceed the memory or storage limits of a function or cause performance degradation. Third, from a security perspective, it is imperative to exercise caution when reusing sensitive data. Data encryption and access control should be used to ensure the safety of the data.

C. Function Reusing

Function reuse is one of the key strategies for optimizing performance and solving cold start issues in serverless computing. This approach means recycling a function instance already initialized and ready to run across multiple requests. The main objectives of function reuse are to reduce a function's

initialization time, speed up its overall execution time, and optimize resource usage.

In serverless environments, function reuse is primarily implemented through the concept of warm instances. The instance created when a function is first called is kept in memory for a certain amount of time, and if additional requests come in during this time, the instance that was kept in memory (the warm instance) is reused to process them. This allows the function logic to be executed immediately without repeating the runtime environment setup, code loading, initialization, and so on, significantly reducing response time.

However, there are a few things to consider when implementing function reuse. First, we need to be careful with state management. If the reused function instance improperly maintains the state from the previous execution, it can cause unexpected behavior. Second, we need to manage memory usage efficiently. Maintaining too many function instances can increase the resource usage of the entire system. Third, we need to maintain proper isolation between functions from a security perspective. When recycling a previously used instance, it should be able to run in isolation from the previous execution.

Various strategies can be considered to reuse functions effectively. For example, we can analyze a function's usage patterns to set the appropriate instance retention time or use predictive scaling to prepare function instances in advance.

IV. Conclusion

This review paper comprehensively examines serverless computing systems' resource reuse paradigm. Serverless computing is an innovative cloud computing model that relieves developers of the burden of infrastructure management, but performance degradation due to cold start issues remains a major challenge. Various resource reuse techniques have been proposed to solve this problem, and in this paper, we have divided them into three categories: container image reuse, data reuse, and function reuse.

Container image reuse is a method that can significantly reduce initialization time by efficiently managing and recycling images that comprise the function execution environment. Data reuse is a strategy to minimize repetitive data loading by caching and recycling data used during function execution. Function reuse is an approach that directly addresses the cold start problem by maintaining and recycling already-initialized function instances.

Each of these reuse techniques has advantages and disadvantages and can be used in combination for maximum effectiveness. Effective resource reuse techniques require comprehensive consideration of the application's characteristics, workload patterns, and performance requirements. It is also important to minimize the security risks of reuse and optimize resource usage efficiency.

In conclusion, the resource reuse paradigm is an important approach that can significantly improve the performance and efficiency of serverless computing. Mitigating cold start issues and maximizing the benefits of serverless computing have the potential to accelerate the adoption of serverless technology in more applications and workloads. With continued research and innovation, serverless computing will evolve into a more mature and powerful cloud computing paradigm.

Acknowledgment

This paper was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea Government(MSIT)(RS-2024-00345869) and Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE) (P0020632, HRD Program for Industrial Innovation)

References

[1] Y. Li, Y. Lin, Y. Wang, K. Ye and C. Xu, "Serverless Computing: State-of-the-Art, Challenges and Opportunities," in IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1522-1539, 1 March-April 2023.

[2] P. Vahidinia, B. Farahani and F. S. Aliee, "Cold Start in Serverless Computing: Current Trends and Mitigation Strategies," 2020 International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 2020, pp. 1-7.

[3] H. Shafiei, A. Khonsari, P. Mousavi. "Serverless Computing: A Survey of Opportunities, Challenges, and Applications," in ACM Comput. Surv., vol. 54, no. 11s, 2022.

[4] T. Kalaiselvi, G. Saravanan, T. Haritha, S. Babu, M. Sakthivel, and Sampath Boopathi, "A Study on the Landscape of Serverless Computing," Advances in systems analysis, software engineering, and high performance computing book series, pp. 260–282, Apr. 2024.

[5] V. Goar and Nagendra Singh Yadav, "Exploring the World of Serverless Computing," Advances in systems analysis, software engineering, and high performance computing book series, pp. 51–73, Apr. 2024.

[6] P. Vahidinia, B. Farahani and F. S. Aliee, "Cold Start in Serverless Computing: Current Trends and Mitigation Strategies," 2020 International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 2020, pp. 1-7.

[7] Anup Mohan, undefined., et al, "Agile Cold Starts for Scalable Serverless," in 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), 2019.

[8] Mustafa Daraghmeh, A. Agarwal, and Yaser Jararweh, "Optimizing serverless computing: A comparative analysis of multi-output regression models for predictive function invocations," Simulation modelling practice and theory, vol. 134, pp. 102925–102925, Jul. 2024.

[9] P. Silva, D. Fireman, T. Pereira, "Prebaking Functions to Warm the Serverless Cold Start," in Proceedings of the 21st International Middleware Conference, 2020, pp. 1–13.

[10] G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017, 2017.

[11] N. Akhtar, A. Raza, V. Ishakian and I. Matta, "COSE: Configuring Serverless Functions using Statistical Learning," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, Toronto, ON, Canada, 2020, pp. 129-138.

[12] H. D. Nguyen, Z. Yang, and A. A. Chien, "Motivating High Performance Serverless Workloads," Proceedings of the 1st Workshop on High Performance Serverless Computing, Jun. 2020.

[13] I. Müller, R. Marroquín, & G. Alonso, "Lambada: interactive data analytics on cold data using serverless cloud infrastructure", Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020.