

Optimization Research and Solutions for the Cold Start Problem in Serverless Computing

Haoyuan Gao*

Software Engineering, Northeastern University, Shenyang, 110000, China

*gaohyuan@mail.sdufe.edu.cn

Abstract. This paper delves into the issue of cold starts in serverless computing, a phenomenon where a function, after not being invoked for a period, requires initialization operations when called again, leading to additional latency. This latency can have a significant impact on applications that require rapid responses. In this essay, we discuss various strategies to address the cold start problem, including optimizations at the hardware level, operating system level, application level, and within serverless computing itself. Hardware-level optimizations primarily involve increasing the number of CPU cores or the size of memory to enhance function execution speed, as well as optimizing network connections and storage devices to reduce data transmission and access latency. Operating system-level optimizations include the use of pre-warming strategies and optimized scheduling policies. Application-level optimizations involve the use of caching, prefetching, function fusion, and other techniques. Serverless computing optimizations involve the use of container-based deployments and the optimization of network connections and storage devices. It also discusses specific research and experiments, such as the REAP mechanism, function fusion technology, and container-based deployment. These studies and experiments demonstrate that while these optimization strategies can effectively reduce cold start latency, they also have their limitations and challenges, necessitating the selection of the most suitable solution based on specific application scenarios and needs.

Keywords: Serverless Computing, Cold Startup, Container Deployment, Optimization Strategies.

1. Introduction

The cold start problem is a key issue in serverless computing, especially on Function-as-a-Service platforms [1]. On FaaS platforms, users incur charges contingent on function execution, thereby naturally encouraging platform providers to prevent the running of idle resources. Yet, this tactic can culminate in the cold start predicament, that is, function execution may be postponed due to the absence of resources. This cold start period could span from hundreds of milliseconds to multiple seconds, a considerable challenge for certain applications.

Several preparations need to be made before the execution of service requests, including starting containers, initializing specific runtimes, such as Python, and loading the relevant function code [2]. In the context of serverless computing, the duration of most function executions typically aligns closely with cold start times, having a similar order of magnitude. The execution of a function requires loading the corresponding resources into memory, and when elastic scaling is needed, the extensive preparatory work in the cold start mode can severely delay function execution [3].

Considering the hardware perspective, modifications such as enhancing the CPU cores count or memory size, and fine-tuning network connections along with storage devices could mitigate data transmission and access latency [3]. As for the operating system layer, techniques such as pre-warming strategies can be employed to adjust the scheduling and resource management methodologies, effectively reducing cold start latency. At the level of system programming, the method of loading and executing functions could be optimized. An example of this is importing only the critical code and obtaining the remaining code on-demand, which can curtail the time it takes for functions to start up [4]. At the application level, optimizing the design and implementation of the application, such as using more efficient algorithms and data structures, can reduce the execution time of functions [5].

2. Related Research

Serverless computing represents a cloud computing model that takes the headache of server management and capacity planning off the developers' shoulders. This model allows them to simply focus on crafting and deploying code, without fretting over the underlying infrastructure. Within this serverless framework, it's up to the cloud service providers to handle infrastructure allocation, scaling, and management automatically. This leaves developers free to pour their energy into building and innovating their applications.

At the heart of serverless computing lies Function-as-a-Service (FaaS). This service model allows developers to fragment their applications into independent, scalable units, or functions, that respond to events. A particular function is only called to action when its corresponding event triggers. This model promotes efficient resource utilization since resources are allocated and billed only during a function's execution.

Serverless computing, while advantageous, poses some challenges, notably the cold start problem. If a function goes uncalled for a while, it must be restarted upon the next call, resulting in latency. Cold start latency can impact user experience and potentially lead to higher billing, because serverless computing charges are typically tied to function execution time. Hence, finding an effective solution to the cold start problem is a crucial research avenue in the field of serverless computing.

Siddharth Agarwal and his colleagues proposed a reinforcement learning agent that determines function call patterns by analyzing metrics like function CPU utilization. It aims to mitigate cold starts in serverless functions by pre-warming function instances. While this method can dynamically learn function call patterns and thereby determine the optimal number of function instances to reduce cold starts, it may lead to overfitting, require large datasets for training, and its applicability might be limited to specific serverless environments like Kubeless due to platform dependencies.

Zijun Li and his team introduced a novel container management scheme, Pagurus. Pagurus swaps idle function containers with a new container already equipped with software packages needed by other functions, which bypasses the requirement to initiate a new container from scratch. However, Pagurus necessitates interoperation on each node, potentially increasing management complexity. Furthermore, its efficiency hinges on the cold start frequency of functions and package similarity. If these attributes vary significantly in practical applications, Pagurus' effectiveness could be compromised.

To sum up, several solutions aim to minimize cold start latency by optimizing hardware configurations. For instance, augmenting the number of CPU cores or memory size can boost function execution speed. Furthermore, optimizing network connections and storage devices at the hardware level can decrease data transmission and access latency. Software optimization provides another approach to mitigate cold start latency. For example, frequently used functions or modules could be preloaded, or the function start-up and execution process could be optimized to minimize cold start latency. Certain studies advocate reducing cold start latency through prediction and scheduling strategies. The REAP method, for instance, aims to decrease cold starts by predicting future requests and pre-starting containers. It also optimizes container usage through scheduling strategies. Moreover, the FaaSLight strategy works by distinguishing and isolating non-essential code from the main application. It achieves this by creating a function-level call graph, which allows it to precisely identify and retain only the code integral to the application's function. This technique effectively reduces the code size during the loading phase, facilitating faster initialization and execution times. Despite this size reduction, the final FaaS application maintains its integrity and operational efficiency.

3. Methods

3.1. Hardware-Level Optimization Methods

The researchers conducted experiments to investigate the impact of simultaneous function calls within the system. They configured a function, `json_dumps`, to operate at over 70% capacity, positioning it in an equilibrium state. Beyond the 8th second, another function, `deltablue`, was triggered at a rate of four times per second over a span of four seconds. The findings revealed that the initiation of `deltablue` caused `json_dumps` to enter a hyperactive state, taking some time to revert to equilibrium once `deltablue` ceased its activity. During this period, a 35% decrease in Instructions Per Cycle (IPC) was observed in comparison to the equilibrium state. This decline was associated with a considerable surge in page faults per thousand instructions, context switches, and Last Level Cache (LLC) misses - all overheads attributed to the initiation of a new container for `deltablue`.

When the function's memory limit was decreased, there was an approximately proportional increase in latency and execution time. However, lowering the memory resulted in a minor increase in capacity, while the surge in latency and execution time was substantially more pronounced.

3.2. Optimizing Network Connections and Storage Devices

Specifically, this method is divided into two stages: source domain and target domain. In the source domain stage, the system extracts contextual information from D2D interactions, including user content viewing history, social relationships, etc. This information is used to predict what content users might request.

Then, in the target domain stage, the system optimizes the caching of strategic content in small base stations based on the prediction results of the source domain stage, considering factors such as the storage capacity of the small base station, the expected popularity of the content, traffic load, and backup capacity.

The advantage of this method is that it can overcome data sparsity and cold start problems, thereby improving the Quality of Experience (QoE) and backup offloading efficiency. The data sparsity problem refers to the fact that in large-scale wireless networks, due to the large number of users and the variety of content, the demand data for content from users is very sparse, which poses challenges for predicting user demand and optimizing cache strategies. The cold start problem refers to the difficulty in accurately predicting user demand when new users or new content join the system due to a lack of historical data.

Through this cache optimization method based on transfer learning, the system can effectively use the contextual information in D2D interactions, predict user demand, and optimize cache strategies, thereby solving data sparsity and cold start problems, and improving user QoE and backup offloading efficiency.

3.3. Operating System-Level Optimization Methods

REAP (Request-Aware Power Management) is a solution. It works mainly in two parts: request-aware preheating strategy and dynamically adjusted idle timeout strategy.

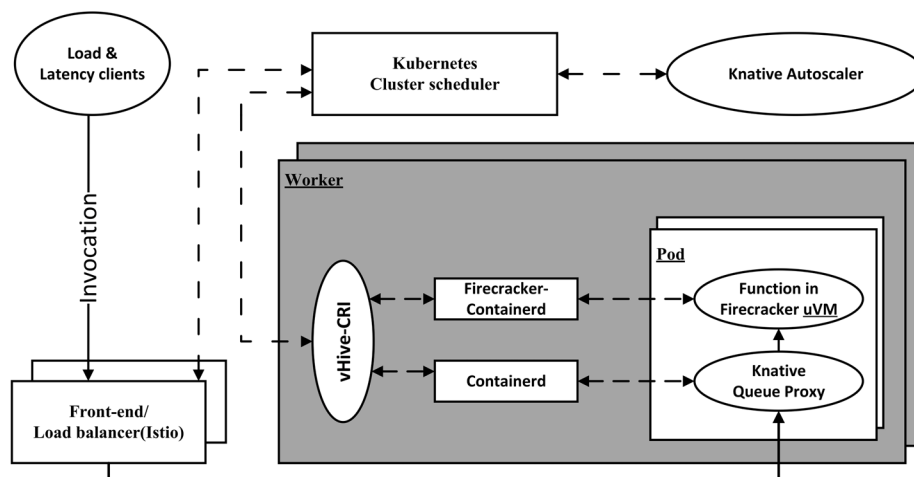


Fig. 1. vHive provides an architectural overview that distinguishes the data plane and the control plane, represented by solid and dashed arrows respectively. (Photo/Picture credit: Original)

Request-aware preheating strategy: REAP preheats function instances based on the prediction of future request patterns by analyzing historical request data, thereby reducing the occurrence of cold starts. This strategy can dynamically adjust the number of preheats according to the actual request pattern, thus reducing cold starts while avoiding resource waste due to excessive preheating as shown in Fig. 1.

REAP employs a dynamically adjusted idle timeout strategy, which balances resource utilization and the frequency of cold starts. Specifically, when resource utilization in the system is high, REAP decreases the idle timeout to release inactive function instances, enhancing resource efficiency. Conversely, when the system's resource utilization is low, REAP prolongs the idle timeout, retaining more active function instances and subsequently decreasing cold starts.

4. Solutions to the Cold Start Problem

4.1. Overview of Solutions

At the hardware level, cold start latency can be reduced by optimizing hardware configurations. Hardware-level optimization can include improving network connections and storage devices to reduce the latency of data transmission and access.

At the operating system level, it can be reduced by optimizing the behavior of the operating system. For example, the REAP (Record, Evict, and Preload) mechanism is an optimization strategy at the operating system level. REAP records the memory pages of a function's stable working set and proactively prefetches them from the disk to the memory. This method can significantly reduce cold start latency.

At the system program level, it can be reduced by optimizing the behavior of functions. For instance, the "Function Fusion" method reduces cold start latency by fusing consecutive functions. Another example is FaaSLight, which builds a function-level call graph to identify code related to application functionality and separates other code, thereby reducing the loading time of the code.

At the application level, AWS Lambda offers ZIP deployment and container-based deployment, each having a unique impact on cold start latency. By selecting an appropriate deployment method, one can effectively mitigate cold start latency.

4.2. Typical Solutions to the Cold Start Problem

4.2.1. Architecture Implications and Strategies for Cold Start Problem in Function as a Service Computing [3]

The impact of short-lived functions on architecture: In a FaaS environment, the lifespan of functions is short enough to affect micro-architectural features related to temporal locality. For

example, current branch predictors have a higher prediction error rate for short functions. This has stimulated research to modify branch predictors. Research can be conducted on fast-training branch predictors or branch predictors that retain the state of each application as shown in Fig. 2.

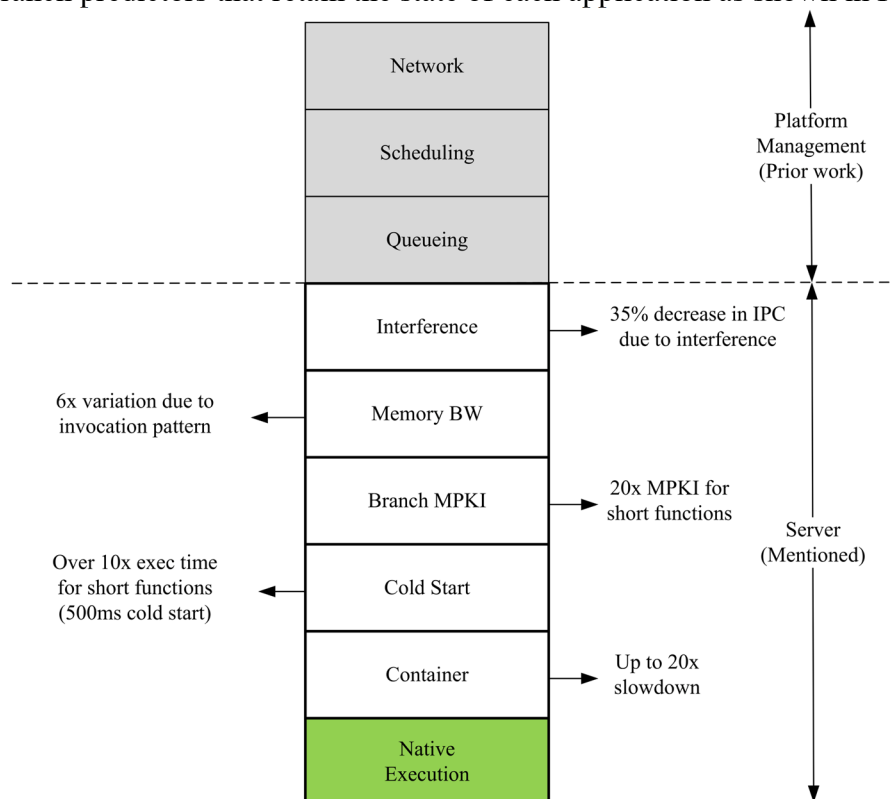


Fig. 2. previous works [6-8] which primarily zeroed in on platform works or point to point situations [3].

The research also examined the influence of the Last Level Cache (LLC). By restricting the LLC size available to CPU cores, the authors analyzed its effect. The study revealed that once the LLC size drops below 2MB, all functions' execution speed substantially diminishes. This discovery suggests that, for these functions, the LLC isn't a performance bottleneck unless it's under 2MB.

The influence of memory constraints on performance: The researchers examined the effects of varying memory constraints on function performance by manipulating the memory limit of functions. They discovered a proportional increase in latency and execution time when the memory limit decreased. However, a further reduction resulted in a marginal increase in capacity, while latency and execution time escalated significantly. This could imply that FaaS providers might be excessively optimizing capacity, potentially sacrificing latency and execution time.

Interference effects: In FaaS, interference effects are severe. For example, when a function is called at over 70% of its capacity, calling another function 20 times within 4 seconds will cause a sustained effect.

4.2.2. REAP Method to Reduce Cold Start Latency of Serverless Functions [9]

REAP is a streamlined software mechanism that logs the memory pages of a function's consistent working set and actively prefetches them from the disk into the memory. In comparison to the baseline snapshot, REAP diminishes the cold start latency by an average of 3.7 times.

The paper describes how REAP operates, with key features including: Parallel processing of page faults: This design employs a tracking file that lists the page offsets of the stable working set. Goroutines fetch the related pages from the client memory file in parallel, thereby facilitating parallel page fault handling.

Utilization of the working set file: The design incorporates a working set file. This technique allows for page retrieval from the working set file at a speed over 3 times faster than parallel page size reading.

The final version of REAP, as presented in section 5.2.3 of the original document, fetches working set pages from the working set file, bypassing the operating system page cache. Compared to a design point, which doesn't bypass the operating system page cache, the prefetching of working set pages from the disk is effectively doubled.

While REAP can substantially enhance the processing speed of actual functions, its efficiency diminishes when many pages are absent from the working set, as seen with code segment calls. In these cases, the function processing latency is reduced on average by a factor of 4.6. However, this improvement is not particularly significant for integrated functions, the exception being video processing.

4.2.3. Function Fusion Method to Reduce Cold Start Latency [10]

In the pursuit of mitigating the cold start issue, the authors propose a strategy termed "Function Fusion". This method aims to decrease cold start latency by combining subsequent functions, all while averting a surge in workflow response time.

Here are some key steps in dealing with the cold start problem: Function fusion decision: If there exist two consecutive functions, fa and fb, where fa precedes fb, the cold start latency of fb can be eliminated by merging these two functions. Post fusion, these two functions transform into a new function, fnew, whose execution time can be described by a given formula (1).

$$T_{fnew} = T_{fa} + T_{fb} \quad (1)$$

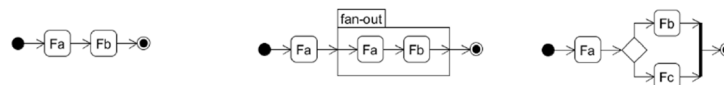


Fig. 3. A very common example is the fused multiply-add operation, where the product of two numbers is added to a third number in a single operation. (Photo/Picture credit: Original)

Determining whether to implement Fan-Out Fusion: If a function, denoted as 'f', is succeeded by a Fan-Out, then fusing the Fan-Out could prolong its execution time. If the increment in execution time owing to parallelism loss surpasses the cold start latency that was eliminated, then the fusion might worsen the issue.

Writing fused functions: During the fusion of functions, an intermediary function is required to coordinate the source functions and facilitate the transfer of input and output data amongst them. The intermediary function is responsible for loading the source functions for invocation. In situations where two consecutive functions bear distinct information unions, the intermediary solution will adapt the information structures between functions as illustrated in Fig. 3.

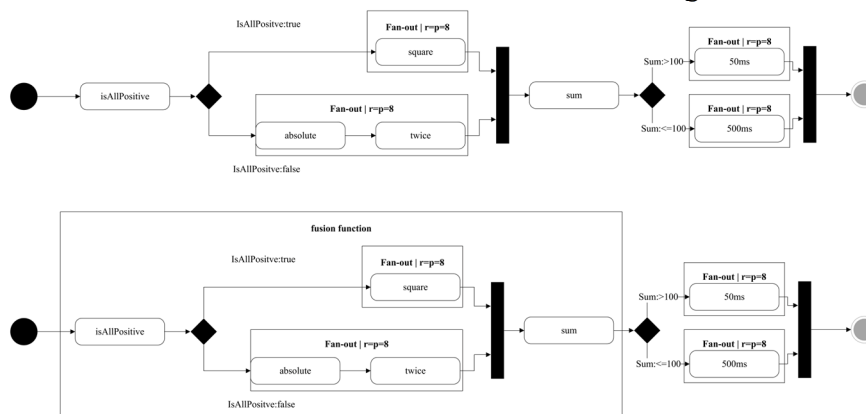


Fig. 4. The first one is an original workflow that is not fused, second one is the workflow fused by the method. (Photo/Picture credit: Original)

Implementation of the fusion automation procedure: The research team has materialized the fusion automation procedure based on this design as depicted in Fig. 4.

4.2.4. AWS Lambda Application Deployment Strategy to Reduce Cold Start Latency [11]

Furthermore, the authors explored the impact of language runtime, memory configuration, and function gathered size on AWS Lambda's initialization time, suggesting various strategies to combat the cold start problem. Their main findings and recommendations are as follows:

ZIP Deployment vs. Container Deployment: AWS Lambda provides two deployment methodologies: traditional ZIP deployment and the more recent container-based deployment. In ZIP deployment, developers must compress the source code into a ZIP file. Upon Lambda's first execution, it imports and unzips this function package from the S3 Bucket, subsequently loading the function in the container for execution. This process can be somewhat lengthy, particularly with large package sizes. Conversely, container-based deployment omits the unzipping and container construction phase, potentially reducing initialization time as showcased in Fig. 5.

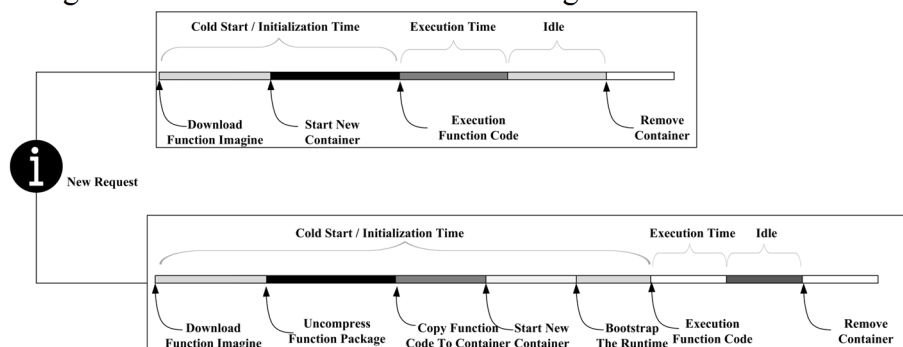


Fig. 5. AWS Lambda lifecycle. (Photo/Picture credit: Original)

5. Conclusion

This paper discusses various strategies for solving the cold start problem, including hardware optimization, operating system optimization, application program optimization, and serverless computing optimization. In terms of hardware optimization, this paper can improve the execution speed of functions and reduce the delay of data transmission and access by increasing the number of CPU cores or memory size, or optimizing network connections and storage devices. In terms of operating system optimization, this paper can use preheating strategies and optimized scheduling strategies. In terms of application program optimization, this paper can use technologies such as caching, prefetching, and function fusion. In terms of serverless computing optimization, this paper can use container-based deployment, optimized network connections, storage devices, etc.

This paper also discusses some specific research and experiments, such as the REAP mechanism, function fusion technology, container-based deployment, etc. These studies and experiments show that although these optimization strategies can effectively reduce cold start latency, they also have their limitations and challenges, and the most suitable solution needs to be chosen according to specific application scenarios and requirements.

Future research may continue to explore more effective solutions. Existing preheating strategies are mainly based on fixed time intervals or request patterns, but these strategies may not effectively cope with dynamically changing workloads. Future research may explore the use of advanced prediction models, such as machine learning, to predict function call patterns and preheat accordingly. Function fusion is an effective strategy to reduce cold start latency, but how to choose the appropriate functions for fusion, and how to handle errors and exceptions of fused functions, are still problems to be solved. Future research may explore more effective function fusion technologies. Container-based deployment is a new deployment method that can reduce initialization time. However, how to effectively manage and schedule containers, and how to handle the security issues of containers, are issues that need further research. Future research may further optimize container-based deployment.

Hardware and network optimization are important means to solve the cold start problem. Future research may explore new hardware architectures or network protocols to further reduce cold start latency. To better understand and solve the cold start problem, this paper needs a more comprehensive performance evaluation and benchmark testing. Future research may develop new performance evaluation methods and benchmark testing tools.

In summary, the cold start problem is a multi-faceted problem that requires consideration of multiple factors. This paper looks forward to future research providing more solutions to meet the performance and availability requirements of serverless computing.

References

- [1] M. Steinbach, A. Jindal, M. Chadha, M. Gerndt and S. Benedict, "TppFaaS: Modeling Serverless Functions Invocations via Temporal Point Processes," in *IEEE Access*, vol. 10, pp. 9059-9084, 2022.
- [2] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking Functions to Warm the Serverless Cold Start. In *Proceedings of the 21st International Middleware Conference (Middleware '20)*. Association for Computing Machinery, New York, NY, USA, 1–13.
- [3] Mohammad Shahradd, Jonathan Balkind, and David Wentzlaff. 2019. Architectural Implications of Function-as-a-Service Computing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 1063–1075.
- [4] S. Agarwal, M. A. Rodriguez, and R. Buyya, "A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency," *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Melbourne, Australia, 2021, pp. 797-803.
- [5] Niu W, Ma X, Lin S, et al. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning[C]//*Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020: 907-922
- [6] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 405–410, June 2017.
- [7] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (ATC 18)*, (Boston, MA), pp. 133–146, USENIX Association, 2018.
- [8] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, vol. 00, pp. 442–450, Jul 2018.
- [9] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. Benchmarking, analysis, and optimization of serverless function snapshots. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 559–572.
- [10] Lee S, Yoon D, Yeo S, Oh S. Mitigating Cold Start Problem in Serverless Computing with Function Fusion. *Sensors*. 2021; 21(24):8416.
- [11] J. Dantas, H. Khazaei and M. Litoiu, "Application Deployment Strategies for Reducing the Cold Start Delay of AWS Lambda," *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, Barcelona, Spain, 2022, pp. 1-10