2024 IEEE 10th International Conference on Edge Computing and Scalable Cloud (EdgeCom) | 979-8-3503-7713-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/EdgeCom62867.2024.00018

Pre-warming: Alleviating Cold Start Occurrences on Cloud-based Serverless Platforms

Thu Yein Htet*, Thanda Shwe†, Israel Mendonça‡, Masayoshi Aritsugi‡
*Graduate School of Science and Technology, Kumamoto University
†Faculty of Advanced Science and Technology, Kumamoto University JSPS International Research Fellow
‡Faculty of Advanced Science and Technology, Kumamoto University
Kumamoto 860-8555, Japan
{thuyein52@st., thandashwe@, israel@, aritsugi@}cs.kumamoto-u.ac.jp

***Abstract*—Serverless computing has gained increasing attention recently due to its benefits, such as streamlined deployment processes and enhanced efficiency. Despite its growing popularity, the well-known cold start problem in serverless platforms limits the wider adoption of serverless computing across applications that need real-time latency and response time. Understanding and mitigating the impact of cold-start latency is crucial for maximizing the efficiency and effectiveness of serverless computing platforms. Cold start delay remains unresolved, particularly in public cloud platforms as research conducted on private serverless platforms cannot be seamlessly applied to public cloud platforms. In this paper, we propose pre-warming strategy which warms the functions ahead of actual function invocation requests by predicting the arrival time of invocations. The results demonstrate a significant reduction in the cold start occurrences and an enhancement in overall performance. Based on our experiments, we found that compared to the baseline method, our proposed model reduced the cold start occurrences up by more than 80%.**

***Index Terms*—Serverless Computing, Cloud Computing, Cold Start, FaaS**

I. Introduction

Serverless computing emerged as an ideal choice for the deployment and execution of workloads such as web applications or machine learning inference by offering a modular and scalable approach while abstracting away the underlying infrastructure and allowing developers to focus solely on code execution [1]–[3]. Serverless platforms such as AWS Lambda [4], Azure Functions [5], or Google Cloud Functions [6], provide automatic scaling, and fees are incurred solely for the duration when the functions are actively executing. This adaptive resource allocation approach enables users to scale resources up or down as needed, ensuring that the system remains responsive to changing demands while maximizing cost-effectiveness.

Although serverless aims for the cost-effectiveness of applications with dynamic and unpredictable workload patterns, some applications require real-time performance requirements, especially end-to-end latency. The "cold start" phenomenon in serverless platforms significantly impacts latency-sensitive applications. Cold start occurs when a serverless function initializes a new instance, loading necessary libraries. For resource-intensive tasks, like loading large libraries and models, cold start latency becomes critical, hindering wider FaaS adoption [7], [8].

Cold starts, a persistent issue in serverless environments, especially for interactive applications requiring quick responses, have attracted significant research attention [9]–[13]. Addressing this problem has led to various studies focusing on system-level optimizations within private serverless platforms [14]–[17]. However, these approaches may not seamlessly translate to public cloud serverless platforms due to limited control and configuration [10], [11], [14]–[16].

Given these considerations, the on-the-top solutions which can be implemented on any serverless platform without altering the core architecture, can be a promising solution to mitigating the cold start occurrence across both private and public serverless platforms. A naive solution is to periodically invoke the functions (e.g. every 5-15 minutes) to keep them warm [18]. However, it is not cost-effective and can introduce significant overhead. Besides, cold start time is not constant, and determining the optimal time for invoking functions is challenging due to varying factors like cloud data center region, runtime configuration, and memory size, all of which can influence the cold start occurrence. Therefore, invoking the function proactively right before the actual invoke can be an adequate solution to prevent the cold start occurrence.

We propose a function pre-warming method for addressing cold starts in serverless platforms. This approach involves pre-warming function instances just before invocation, leveraging the XGBoost algorithm for cold start prediction. When a cold start is predicted, we proactively warm up the function to ensure swift execution without delay. Consequently, this proactive-warming significantly reduces cold-start delays and mitigates their impact on function invocations. Our contributions include:

- To address the challenge posed by cold starts, we develop a pre-warming strategy focused on warming up functions ahead of function invocation requests, thus mitigating the cold start occurrence and impact on system performance.

This work was supported in part by JSPS KAKENHI Grant Numbers JP24K02944 and JP22F22069.

979-8-3503-7713-2/24/$31.00 ©2024 IEEE
DOI 10.1109/EdgeCom62867.2024.00018

- We investigate the cold start problem in public cloud (AWS), considering both how often it occurs and the extent of its impact.
- We evaluate our proposed pre-warming method with current serverless implementation and naive solution across three function types, namely, image extract, classify image, and step functions.

The rest of the paper is organized as follows: Section II discusses related studies on various approaches that mitigate the cold start issues in serverless platforms. Section III describes an investigation on cold start occurrence in serverless platforms. Our proposed system design and detailed description of each module are presented in Section IV. Section V presents an experimental evaluation of our proposed pre-warming strategy, and we discuss the results in Section VI. Finally, Section VII concludes the paper and gives an outlook on future research.

II. Related Work

A. System-level Optimization

Roy et al. [14] introduced IceBreaker, a method for dynamically selecting the most cost-effective node type to warm up a function based on its time-varying invocation probability. IceBreaker comprises two main components: the function invocation predictor and the placement decision maker (PDM).

Li et al. [15], presented Pagurus, a solution that addresses cold startup problems by repurposing idle warm containers across functions. It employs an innovative container management scheme focusing on inter-function container sharing, diverging from traditional prewarm strategies. An adaptive container provisioning model (ACPM) was designed in [10] to reduce cold-start latency by dynamically provisioning containers in two stages.

Current researches on mitigating cold starts in serverless platforms primarily focuses on container management techniques such as caching and pre-warming. However, these approaches often conflict with the fundamental principle of serverless computing, which emphasizes charging users based on actual resource consumption. Our proposed solution offers a novel approach that can be seamlessly integrated into existing serverless platforms without necessitating modifications to the underlying infrastructure. By proactively invoking functions and predicting cold start occurrences, we effectively reduce both container initialization and execution preparation times. This methodology is applicable to a wide range of serverless platforms, whether public or private, and holds particular significance for real-time interactive applications where minimizing latency is paramount.

B. Optimization based on Application Level

Liu et al. [12] presented FaaSLight which reduces cold start latency as application-level optimization. FaaSLight detects application functionality-related code by creating a function-level call graph and distinguishes it from other optional code. Mahgoub et al. [13] presented ORION to optimize the serverless Directed Acyclic Graph (DAG), aiming to reduce the end-to-end latency of serverless DAG applications.

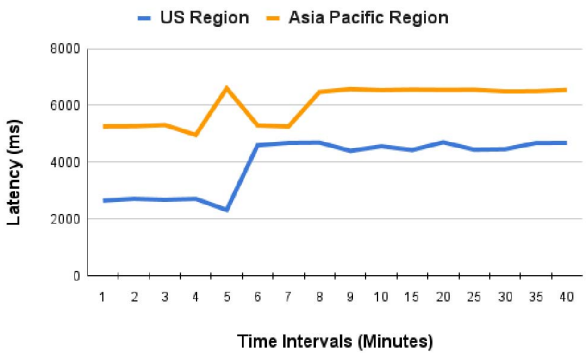

Fig. 1. Cold Start Occurrence Time

Lee et al. [17] presented a function fusion scheme, that aims to mitigate the cold start delay by decreasing the response time of the workflow by merging functions while taking into account branches and parallelism. To accomplish this, elements of the workflow are identified and three types of latencies are used to predict the response time of a workflow and determine whether fusion is beneficial.

Application-level optimization strategies rely on the specific structure and stages of the application, often requiring modifications to the application code or an examination of its structure. In contrast, our approach applies to any serverless function without the need to inspect the application's structure or code, making it more versatile and widely applicable.

III. Investigation on Cold Start and Motivation

In this work, experiments were conducted in the Amazon Lambda service to understand cold start occurrence times better. To ensure investigation quality, two different regions, Asia Pacific Region (Tokyo) and US Region (North Virginia), were selected for the Lambda function. The detailed configuration for the environment is shown in Section V Table II. In a series of experiments, the function was executed multiple times at uniform and varied time intervals. A simple Lambda function was deployed to read a photo file from AWS S3 and generate metadata. The experiments comprised 16 cycles, with time intervals ranging from 1 to 10 minutes in the first 10 cycles, followed by intervals of 15, 20, 25, 30, 35, and 40 minutes for the remaining cycles. The function was triggered 10 times within each cycle, and the average value was computed. The observed latencies for each cycle and cold start occur times are shown in Fig. 1.

Observation 1. The cold start is happening earlier than expected.

From the experiments, we noticed that cold starts in serverless platforms happen sooner than noted in previous research [19], [20] which is 40 minutes. As shown in Fig. 1, leading up to the 6-minute threshold, the average execution latency remains relatively lower compared to the latency

TABLE I
FUNCTION EXECUTION TIME

Function Type	Normal Execution Time (ms)	Cold-Start Execution Time (ms)	Percentage Increased
Image Extract (Single Lambda Function)	936.42	2800.93	299.1%
Classify Image (Containerized Function)	4830.16	13978.93	289.4%
Image Processing (Step Functions)	2700.00	9400.00	348.1%

observed beyond the 6-minute mark. This observation suggests an onset of cold-start occurrences at the 6-minute interval between the two function invocations. This finding indicates a possible change in cold-start behavior that needs more investigation. In addition, a noteworthy finding emerged regarding the relationship between inter-invocation time intervals and cold-start occurrences.

Observation 2. Cold-start is likely to begin when the interval between two invocations exceeds 6 minutes or more.

In our experiments, we observed that cold-start events begin to occur when the time delay between successive function invocations exceeds a threshold of 6 minutes in both tested regions. This threshold serves as a point of reference for understanding the situations associated with cold-start phenomena within serverless platforms. This finding highlights the need to mitigate cold-start latency and optimize serverless performance.

To examine the severity of cold start, we conducted experiments utilizing three distinct Lambda functions, namely, Image Extract (Single Lambda Function), Classify Image (Containerized Function), and Image Processing (Step Functions). Our methodology involves comparing the regular execution time with the execution time observed during cold start occurrences. The findings of these comparative analyses are presented in Table I.

Observation 3. The impact of cold starts may vary depending on the region where the function is deployed and the complexity of the function.

In Table I, the findings underscore the substantial impact of cold-start latency on the performance of serverless functions, particularly for tasks involving large-sized functions and function chains. Based on the findings of our experiments shown in Fig. 1 and Table I, the impact of cold start varied depending on several factors. These factors include the geographical region in which the function is deployed and the complexity of the function itself.

IV. SYSTEM DESIGN

In this section, we explain the design of our proposed prewarming approach to reduce the cold start occurrence, highlighting its flexibility for effortless integration into serverless platforms.

A. System Design Overview

Fig. 2 illustrates the system overview of the proposed model, showcasing its primary components: Threshold Determinator, Predictor, and Pre-warmer. The system follows a structured procedure. Firstly, the Threshold Determinator orchestrates several cycles to identify the exact time of cold start occurrence through an iterative process. It enhances to precisely pinpoint the cold start instances. Subsequently, the cold-start time is relayed to the Pre-Warmer for further process in the warm-up of the functions. Simultaneously, as the user invokes functions, the system captures and stores the inter-arrival times of incoming requests. Leveraging the historical inter-arrival time of function requests, the prediction model then forecasts the timing of the next request arrival. Following this, the predicted arrival time is compared with the cold-start time stored in the Pre-Warmer. This comparison determines the likelihood of a cold start event. If the predicted arrival time hints at a potential cold start, proactive measures are taken. Specifically, the function undergoes pre-warming in advance to ensure they are ready for execution before the actual invocation arrives, effectively reducing cold start delays.

B. Threshold Determinator

The primary function of the Threshold Determinator is to determine the occurrence of cold starts specific to the service and deployment region of the function. To achieve this, the Threshold Determinator employs a function that undergoes multiple cycles of execution, each with varying time intervals. During each cycle, the execution time of the function is recorded and compared against previous cycles at equivalent intervals to identify the presence of cold start occurrences. This iterative process begins with time intervals set from lower to higher values. If the execution time of a subsequent invocation is found to be lower by a predefined threshold compared to its predecessor, it is indicative that no cold start has been discovered. Conversely, if the execution time fails to exhibit a decrease, it signals the occurrence of a cold start. Upon identification of a cold start time, the Threshold Determinator verifies the corresponding time interval and transmits this information to the Pre-warming Stage for further action.

C. Predictor

The primary function of the Predictor is to forecast the timing of the next request arrival by analyzing patterns from previous request arrivals. Various time series prediction techniques, including XGBoost, Light GBM, ARIMA, LSTM, Regression, Random Forest, and GRU models, were experimented with to achieve accurate forecasts. The performance of these methods was evaluated using Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE) metrics, with the dataset split into 70% for training and 30% for testing. Among these models, XGBoost demonstrated superior performance, with a Root Mean Square Error of 0.01179 on test data. Evaluation using Azure Trace data reaffirmed XGBoost's effectiveness, achieving a Root Mean Square value of 0.02203 and the results are illustrated in

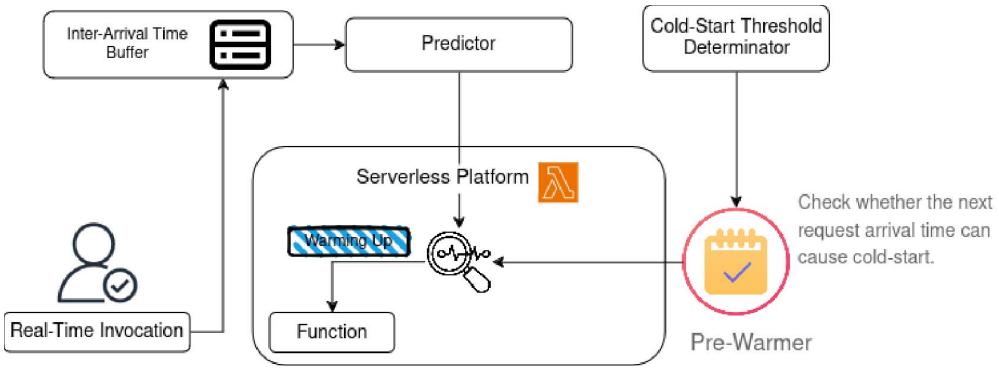

Fig. 2. Workflow of Cold Start Mitigating System

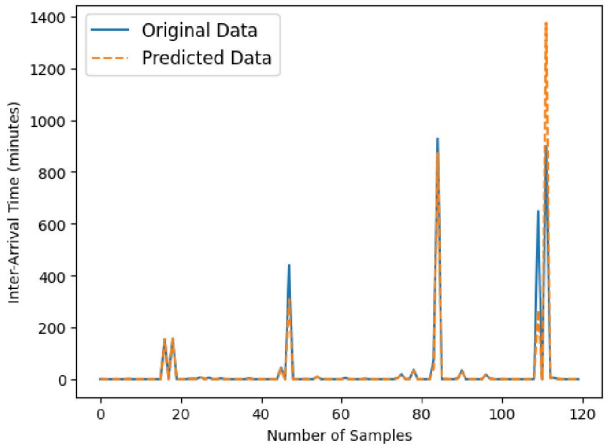

Fig. 3. Performance of XGBoost on Azure Trace

Fig. 3. These results underscore the robustness and efficacy of the XGBoost model in forecasting request arrival times across different datasets and scenarios, leading to its deployment in our Predictor module.

D. Pre-Warmer

The pre-warmer plays an important role in reducing cold-start occurrences within the system. Initially, it captures and stores the onset time of cold starts for subsequent analysis. Upon receiving prediction data from the Predictor, the pre-warmer scrutinizes the predicted arrival time against the stored cold-start onset time. If the predicted arrival time does not pose a risk of a cold start, the function is invoked normally with actual invocation. Conversely, if the predicted arrival time indicates a potential cold start, the pre-warmer initiates a preemptive function invocation slightly ahead of the predicted arrival time of function request, a process commonly referred to as "warming up". This proactive approach aims to mitigate the likelihood of cold-start occurrences, ensuring more efficient function execution without experiencing additional delay due to cold starts. Algorithm 1 describes our proposed pre-warming strategy. In lines 6, 7, and 9, of Algorithm 1 we used the term 'InvokeFunction' because our host platform is AWS Serverless and the functions reside within the AWS Lambda service, where the term 'invoke' is commonly used for function execution.

Algorithm 1: Cold Start Mitigation Algorithm

Input: `[prev_func_arrival_times]`

```
1  model ← TrainModel([prev_func_arrival_times])
2  cold_start_occurrence_time ← FindColdStartOccurrenceTime();
3  while true do
4      next_arrival_time ← PredictNextArrivalTime(model);
5      if next_arrival_time > cold_start_occurrence_time then
6          /* Pre-warming the Serverless Functions on Lambda */
           InvokeFunction(FunctionName);
7          /* Calling Serverless Functions on Lambda */
           InvokeFunction(FunctionName);
8      else
9          /* Calling Serverless Functions on Lambda */
           InvokeFunction(FunctionName);
```

V. EXPERIMENTAL EVALUATION

In this section, we evaluate pre-warming with the aim of answering the following research questions.

- **(RQ1)** How many instances of cold starts can be mitigated through the proposed pre-warming method?
- **(RQ2)** What is the cost-effectiveness of implementing the proposed pre-warming method?

TABLE II
DEPLOYED FUNCTIONS

Function Type	Function Description	Environment Configuration
Image Extract (Single Lambda Function)	Read a file that is a photo from the AWS S3 (Simple Storage Service) and produce metadata.	Memory - 128 MB Storage - 512 MB Timeout - 10 seconds.
Classify Image (Containerized Function)	Leverage a Docker container encapsulating TensorFlow with the Inceptionv4 model to process images stored in Amazon S3, subsequently deploying the containerized function to AWS Lambda.	Memory - 3500 MB Storage - 3000 MB Timeout - 10 seconds.
Image Processing (Step Functions)	A workflow with four functions chained together, Image Extract, Transform Metadata, Storedata and Thumbnail. Image Extract - Extract metadata, Transform Metadata - transforming the acquired metadata, Storedata - storing the transformed data into Dynamo DB, Thumbnail - resizing the image	All the functions use default configuration which is Memory - 128 MB Storage - 512 MB Timeout - 3 seconds.

- **(RQ3)** How does the utilization of a prediction model within the proposed pre-warming method reduce the adverse impact of cold starts on actual invocations?

A. Methodology

Baselines. As there are no direct existing solutions that are similar to our proposal, we compare our approach with (1) *Baseline*: Current AWS Lambda implementation, and (2) *Naive solution*: invoking the function as a warm-up trigger with a fixed time interval of five minutes.

Testbed. We implement pre-warming in Python and evaluate using invocation traces sampled from Azure Functions traces [21]. Our experiments are conducted in AWS Lambda in ap-northeast-1 region. The function invocation requests are submitted from t3.small EC2 instance (CPU: 1vCPU, Mem: 1 GB, OS: Amazon Linux) in the same region.

Evaluation metrics. Firstly, we examined the number of cold start occurrences across three methods for three different functions to understand the frequency of cold starts. Following that, we proceed with cost evaluation to examine the operational expenses. Then, we analyzed the latencies of workload execution to investigate the impact of our proposed method on the function execution workloads.

Deployed Functions. We deployed three function types which are described in Table II in ap-northeast-1 region to investigate the cold start impact across various function sizes and complexity.

Workloads. To mimic the workloads, we used the Azure functions trace dataset [21] which comprises function invocations for two weeks. A total of 1,980,951 function invocations were recorded with the hosting of 83,137 functions. In the trace, the most accessed application recorded 535,667 invocations, while the least accessed application was invoked only once. We opted to use the median value of 415 invocations that can represent the typical access frequency over the two-week duration. This frequency strikes a balance, being neither excessively frequent nor less frequent, allowing us to thoroughly assess and evaluate the impact of cold starts. Then, we segmented into two equal portions: one comprising 11 days of data and the other comprising 3 days of data. We extracted inter-arrival time between successive invocations containing 3 days' worth of data, consisting of 205 entries, to ensure efficient experimentation, and examine cold start behavior.

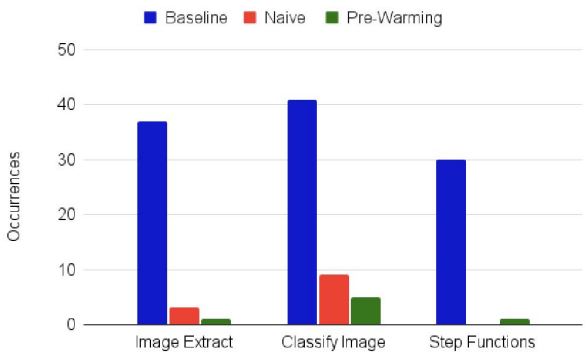

Fig. 4. Cold Start Occurrence Comparison

B. Results

*1) **Cold Start Occurrence (RQ1)**:* To answer RQ1, we conducted a comparative analysis to assess the frequency of cold-start occurrences between our pre-warming method and the baseline approaches across three function types. The results of this comparison are shown in Fig. 4.

The baseline method exhibits more frequent cold starts compared to our pre-warming approach, attributable to longer intervals between requests increasing the likelihood of cold starts. Our pre-warming approach notably reduces the number of cold starts by proactively preparing containers based on predicted values before actual invocations. This preemptive warming ensures that containers and libraries are ready when invocations occur. However, containerized functions experience elevated cold-start occurrences due to their higher memory and space allocations, resulting in earlier load-off times during periods of high request rates. Despite implementing naive solution methods, cold start issues persist, likely due to traffic conditions during peak hours.

*2) **Cost Comparison (RQ2)**:* As for cost comparison, we compare the total cost of the Baseline, proposed pre-warming method, and Naive Solution. Fig. 5 illustrates the comparison between them.

As depicted in the figure, the baseline exhibits the lowest cost, due to its execution of normal workloads only, without additional warm-up invocations. In contrast, our proposed method incurs a slightly higher cost compared to the baseline. This cost is attributed to the predictive nature of the method, which activates the function preemptively, based on predicted

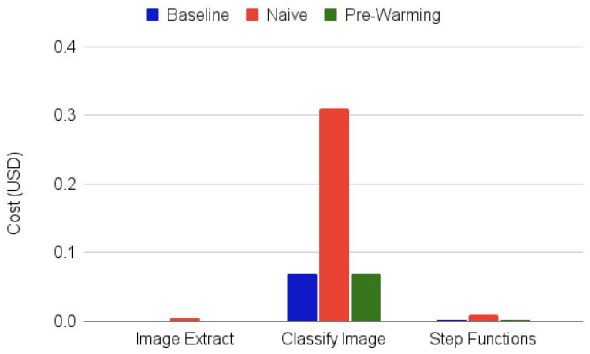

Fig. 5. Cost Comparison

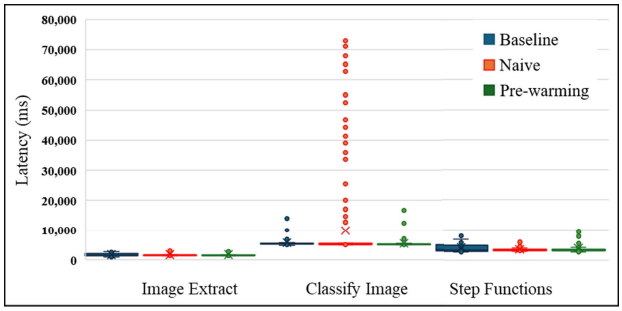

Fig. 6. Latency Comparison

arrival times that may induce cold starts. Although our proposal incurs a slightly higher cost, it effectively reduces cold-start occurrences and reduces the cold start impact on function invocations. The naive solution incurs the highest cost due to the continuous warming-up processes. As it operates under a periodical and frequent warm-up invocation strategy, warming up the function at every 5-minute interval contributes to the highest cost in Naive Solution.Instead of invoking functions continuously, our method only warms up the function immediately before actual invocation by predicting the arrival time of function requests, resulting in significant cost savings.

*3) **Impact of Cold Starts (RQ3)**:* Fig. 6 shows the latencies of function executions and comparisons between our proposed method, the baseline, and the naive approach. The latencies observed in function executions indicate that our pre-warming strategy outperforms traditional methods across all three function types. This difference comes from the increased latency associated with cold-start occurrences during normal execution. In the absence of pre-warming, functions draw overhead related to library preparation, container setup, and source code loading upon invocation, resulting in elevated latency. Although the naive solution minimizes cold starts for the majority of functions, instances of significantly higher latency occur sporadically, potentially attributable to preparation stage inefficiencies. Conversely, our pre-warming strategy strategically prepares containers in advance, mitigating cold-start occurrences and associated latencies. This proactive approach experiences lower execution times during actual invocations, enhancing efficiency and responsiveness. Notably, chain functions are most affected by cold starts due to the cumulative effect across each function in the chain. While our experiment involved a chain with four functions, the impact would be more pronounced in longer chains. Our pre-warming approach effectively addresses the cold start impact within chain functions, underscoring its versatility and efficacy in reducing cold-start effects.

VI. Discussion and Limitations

While our proposed approach effectively reduces cold start occurrences, several limitations should be added to consideration. Firstly, deploying our pre-warming module on separate instances introduces additional costs for deployment and maintenance. However, leveraging small instances available within the free tier may mitigate these expenses. For consideration of deployment costs with billable usage, among the offerings available on Amazon EC2, we suggest that the t3.small instance as particularly suitable for its balanced combination of memory size and virtual CPUs. Choosing the t3.small instance involves an approximate cost of $0.0272 per On-Demand Hour. Secondly, the accuracy of the prediction model used to forecast incoming function requests poses a challenge. Despite rigorous selection and testing of prediction models using Azure trace data and random Poisson distribution, uncertainties persist. Inaccurate predictions may lead to premature or delayed pre-warming invocations, undermining the efficacy of our strategy. The predictive model plays a crucial role in our approach to mitigate undesirable outcomes. Inaccurate predictions can result in two problematic scenarios: 1) pre-warm invocations occurring prematurely, leading to a notable delay between warming up and actual use, potentially resulting in subsequent occurrences of cold starts; and 2) pre-warm invocations occurring late, thereby exposing actual invocations to the risk of cold starts.

Moreover, our experiments were conducted solely on AWS, limiting the generalizability of our findings to other platforms. Although our approach may appear vendor-specific, we assumed that serverless computing principles remain consistent across providers. In our experiments and evaluations, we used AWS EC2 instance for hosting our proposed pre-warmer while AWS Lambda was employed for executing functions. Similar setups can be arranged on other major cloud providers, such as Google Cloud Platform and Microsoft Azure. Our pre-warmer is deployable on a compute instance, Google Cloud Compute Engine for GCP and Azure Virtual Machines for Microsoft Azure. For functions deployment, Google Cloud Functions or Azure Functions can be used as serverless platforms. However, further exploration across diverse platforms is warranted to validate this assumption. Additionally, our evaluation compared our proposal with existing implementations and naive solutions, as opposed to comprehensive comparisons across various optimization approaches. While system-level optimizations require modifications to underlying platforms,

our approach offers a vendor-agnostic solution deployable on public serverless platforms without extensive platform modifications or application code alterations.

VII. Conclusion

This paper proposed a pre-warming approach that can be easily deployed to reduce the cold start occurrence and impact of cold start on actual function invocations in serverless platforms. The approach involves forecasting the timing of incoming function requests using an XGBoost model. By initiating pre-warming invocations ahead of actual requests, the likelihood of cold starts was significantly reduced compared to the baseline, which are, for the Image Extract function, the percentage was nearly 93%. For the Classify Image function, it approached 89%. In the case of the Step Functions, it was close to 96%. Furthermore, our approach demonstrated cost-reduction benefits when compared to the naive solution.

Future research will explore solutions to further reduce cold start occurrences in larger containerized serverless functions. We also intend to optimize the handling of long function chains.

References

[1] T. P. Bac, M. N. Tran, and Y. Kim, "Serverless computing approach for deploying machine learning applications in edge layer," in *2022 International Conference on Information Networking (ICOIN)*, 2022, pp. 396–401. [Online]. Available: https://doi.org/10.1109/ICOIN53446.2022.9687209

[2] Z. Tu, M. Li, and J. Lin, "Pay-per-request deployment of neural network models using serverless architectures," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, Y. Liu, T. Paek, and M. Patwardhan, Eds. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 6–10. [Online]. Available: https://aclanthology.org/N18-5002

[3] A. Barrak, F. Petrillo, and F. Jaafar, "Serverless on machine learning: A systematic mapping study," *IEEE Access*, vol. 10, pp. 99 337–99 352, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3206366

[4] (2014) Aws lambda. [Online]. Available: https://aws.amazon.com/jp/lambda/

[5] (2016) Azure functions. [Online]. Available: https://azure.microsoft.com/ja-jp/products/functions

[6] (2017) Google cloud functions. [Online]. Available: https://cloud.google.com/functions

[7] S. Kounev, N. Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana, and A. A. Chien, "Serverless computing: What it is, and what it is not?" *Commun. ACM*, vol. 66, no. 9, pp. 80–92, Aug. 2023. [Online]. Available: https://doi.org/10.1145/3587249

[8] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[9] S. Agarwal, M. A. Rodriguez, and R. Buyya, "A reinforcement learning approach to reduce serverless function cold start frequency," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 797–803. [Online]. Available: https://doi.org/10.1109/CCGrid51090.2021.00097

[10] A. Kumari and B. Sahoo, "ACPM : adaptive container provisioning model to mitigate serverless cold-start," *Cluster Computing*, 05 2023, https://doi.org/10.1007/s10586-023-04016-8. [Online]. Available: https://doi.org/10.1007/s10586-023-04016-8

[11] B. Sethi, S. K. Addya, and S. K. Ghosh, "LCS: Alleviating total cold start latency in serverless applications with lru warm container approach," in *Proceedings of the 24th International Conference on Distributed Computing and Networking*, ser. ICDCN '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 197–206. [Online]. Available: https://doi.org/10.1145/3571306.3571404

[12] X. Liu, J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, and X. Jin, "FaaSLight: General application-level cold-start latency optimization for function-as-a-service in serverless computing," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3585007

[13] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, and S. Bagchi, "ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 303–320. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/mahgoub

[14] R. B. Roy, T. Patel, and D. Tiwari, "Icebreaker: Warming serverless functions better with heterogeneity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 753–767. [Online]. Available: https://doi.org/10.1145/3503222.3507750

[15] Z. Li, L. Guo, Q. Chen, J. Cheng, C. Xu, D. Zeng, Z. Song, T. Ma, Y. Yang, C. Li, and M. Guo, "Help rather than recycle: Alleviating cold startup in serverless computing through Inter-Function container sharing," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 69–84. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/li-zijun-help

[16] A. Fuerst and P. Sharma, "Faascache: Keeping serverless computing alive with greedy-dual caching," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 386–400. [Online]. Available: https://doi.org/10.1145/3445814.3446757

[17] S. Lee, D. Yoon, S. Yeo, and S. Oh, "Mitigating cold start problem in serverless computing with function fusion," *Sensors*, vol. 21, no. 24, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8416

[18] (2023) How to keep your lambda functions warm. [Online]. Available: https://www.pluralsight.com/resources/blog/cloud/how-to-keep-your-lambda-functions-warm

[19] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 159–169. [Online]. Available: https://doi.org/10.1109/IC2E.2018.00039

[20] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–15. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00073

[21] (2019) Azure public dataset. [Online]. Available: http://github.com/Azure/AzurePublicDataset