International Journal of Advanced Smart Convergence Vol.13 No.3 109-116 (2024)
http://dx.doi.org/10.7236/IJASC.2024.13.3.109

IJASC 24-3-12

A Study on Cold Start and Resource Improvement Using Time Warming Allocation Engine in Serverless Computing

Gun-Woo Kim*, Seok-Jae Moon**, Byung-Joon Park***

**Master, Department of Computer Science, Kwangwoon University, Korea*
***Professor, Department of Artificial Intelligence Institute of Information Technology, KwangWoon University, Korea*
****Professor, Department of Computer Science, Kwangwoon University, Korea.*
E-mail: {kgwo0528, msj8086, bjpark}@kw.ac.kr

Abstract

With the advent of serverless computing, cloud customers no longer needed to maintain and manage server environments directly. Instead, cloud service providers took on that role, managing and maintaining the server environment according to customer requests, a concept known as Function as a Service (FaaS). This service demonstrated improvements in operational costs and resource utilization over traditional cloud computing, offering various advantages such as enhanced scalability. However, a delay occurred in processing and returning results to user requests, a phenomenon referred to as the cold start problem. This paper proposed the Time Warming Allocation Engine (TWAE) to improve resource management and mitigate the cold start problem in Function as a Service. The proposed engine comprised a collection module, a learning module, a classification module, and an allocation module. Additionally, it utilized a list called Pre-Warming. Through this approach, it suggested directions for improving cold start issues and resource utilization according to different time periods.

Keywords: *Serverless Computing, Cloud Computing, Reinforcement Learning, Resource Provisioning, Integrated Environmental Management*

1. Introduction

The advent of cloud computing marked a significant leap towards viewing computing as a utility. Application architectures rapidly evolved from monolithic structures to service-oriented architectures, and then to microservices architectures. This evolution recognized the potential of executing small pieces of code as functions, leading to today's Function as a Service (FaaS) [1]. Serverless computing is a model where cloud customers write code without needing to maintain and manage server environments directly; instead, cloud service providers handle server provisioning and management tasks [2]. Notable serverless services include Amazon Lambda, Google Cloud Functions, and Microsoft Azure Serverless. FaaS, a form of serverless

Manuscript Received: July. 13. 2024 / Revised: July. 21. 2024 / Accepted: July. 26. 2024
Corresponding Author: msj8086@kw.ac.kr
Tel: +82 -10-7753-8086, Fax: +82-050-4366-4238
Author's affiliation Professor, Department of Artificial Intelligence Institute of Information Technology, KwangWoon University, Korea

Copyright© 2024 by The Institute of Internet, Broadcasting and Communication. This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/4.0)

computing, allows cloud customers to use specific functions without maintaining servers, allocating resources only when needed, and charging based on actual usage [3]. This model offers advantages in operational costs, resource consumption, ease of application and product development, and scalability. Despite these benefits, several issues still need addressing. One of the most critical challenges is the cold start problem and resource allocation issues. The cold start refers to the delay that occurs when a function is invoked for the first time or after a long period of inactivity, caused by the creation and initialization of a new container for function execution [4]. This delay affects latency-sensitive applications, such as smart health in the Internet of Things (IoT), which requires real-time responses to current patient conditions or online data analysis [4]. Various approaches have been proposed to address this issue, including the use of always-on containers for each function [5], research on hot and cold container queues for different functions [6], and studies on serverless utilization in edge computing [7].

This paper proposes the Time Warming Allocation Engine (TWAE) to address the cold start problem and improve resource utilization in serverless environments. TWAE employs Policy Ensemble Reinforcement Learning. The learning data is based on historical time-based statistical data. Models generated through time-based reinforcement learning create pre-warming queues for different times. At this stage, functions are identified and selected for the pre-warming queue based on service-appropriate thresholds. This approach improves the existing cold start issues while enabling real-time resource utilization improvement. The structure of this paper is as follows. Section 2 reviews related research, and Section 3 describes the structure and flow of the proposed Time Warming Allocation Engine. Section 4 discusses the performance analysis of the proposed engine, and finally, Section 5 concludes the paper.

2. Related Work

In the serverless computing environment, various models and architectures have been comprehensively studied to address the cold start problem, resource allocation, and network offloading issues. Approaches to resolving the cold start problem include utilizing AI-based models for workload prediction, adopting function fusion techniques to combine sequential functions, implementing advanced container preparation strategies such as the WLEC architecture using S2LRU++, proposing system-level methods like the SEUSS system which deploys functions using unikernel snapshots, and frameworks like vHive which store function images on disk and pre-load the page working set for re-execution. Additionally, pre-warming functions to keep containers warm and execution-ready has also been proposed [8]. To tackle the offloading issue, several approaches have been suggested. These include grouping workflow functions within the same instance to reduce initialization costs and eliminate intermediate data transfer through data locality, and approaches like FnSched, which adjusts CPU sharing to inspect available resources and accommodate incoming application calls [9].

3. Proposed System

3.1 Time Warming Allocation Engine (TWAE)

This section discusses the structure of the Time Warming Allocation Engine in a serverless computing environment. Figure 1 illustrates this structure. The Time Warming Allocation Engine comprises the Collection Module, Learning Module, Classification Module, and Schedule Module.

- **Collection Module.** In this module, information regarding the type of service requested by the cloud customer, the required resource usage, and the time needed for future learning is periodically preprocessed.

Cloud customers use APIs defined by the cloud service provider to request functions on the server. After scheduling, a container with an appropriate environment for the function is allocated to a suitable server. Once the operation is completed, the results are delivered to the cloud customer, and the resources used, time taken, and request information are collected in storage. The Collection Module is invoked before the model learning process, performing preprocessing and verifying the validity and completeness of the collected information.

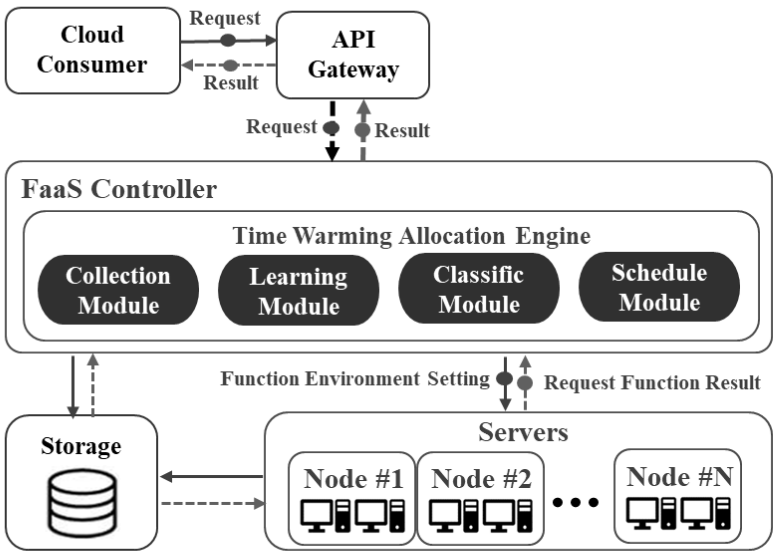


Figure 1. Overall structure of the proposed system

- **Collection Module.** In this module, information regarding the type of service requested by the cloud customer, the required resource usage, and the time needed for future learning is periodically preprocessed. Cloud customers use APIs defined by the cloud service provider to request functions on the server. After scheduling, a container with an appropriate environment for the function is allocated to a suitable server. Once the operation is completed, the results are delivered to the cloud customer, and the resources used, time taken, and request information are collected in storage. The Collection Module is invoked before the model learning process, performing preprocessing and verifying the validity and completeness of the collected information.

- **Learning Module.** This module uses the time-based statistical data collected by the Collection Module to train a model that can determine whether a function is in a Hot, Warm, or Cold state. It flexibly processes various data, including resource consumption information for each function and dependent services. To achieve this, Policy Ensemble Reinforcement Learning [10] is employed to assess resource usage and the state of the functions. The reinforcement learning model is pre-trained to estimate and classify when (time), in what environment, and how much resources and time each function will consume. This pre-training occurs before invoking the Classification Module. This approach allows for real-time improvement in resource utilization by allocating resources tailored to each function and dynamically handling new functions. The Learning Module is invoked periodically (e.g., every hour) for training. To achieve effective results, various clustering and neural-network methods can be employed.

- **Classific Module.** This module uses the pre-trained reinforcement learning model to estimate the state (Hot, Warm, Cold) and resource consumption of each function. The Classification Module includes two subroutines: Reservation-Classific and Request-Classific. Functions frequently used in the current time frame are classified as Warm, while those not frequently used are classified as Cold. The Hot state refers to functions called more frequently than those in the Warm state. Reservation-Classific: This subroutine is invoked every hour (or a different interval based on the situation) to determine the state of each function. Request-Classific: This subroutine is called by the Gateway API whenever there is a function request from a user. It estimates resource consumption using the pre-trained model and then calls the Schedule Module's Request.

- **Schedule Module.**. This module schedules the appropriate environment for the functions requested by cloud customers based on their pre-classified states. The Schedule Module consists of the subroutines Pre-Warming, Pre-Hotting, Colding, and Request, each of which operates according to the classified states. Request: Invoked by the Classification Module, it sets up the container environment based on the estimated resource consumption. The periodically operating subroutines are Pre-Warming, Pre-Hotting, and Colding, functioning in the order listed:
  - **Pre-Warming:** For functions in the Warm state, it preemptively calls dependent services to create containers.
  - **Pre-Hotting:** For functions in the Hot state, it creates not only the dependent services but also a container ready for the function service.
  - **Colding:** For functions in the Cold state, it terminates the containers of previously Warm or Hot functions' dependent services.

By implementing these routines, the Schedule Module aims to reduce cold start latency, operational costs, and improve Quality of Experience (QoE).

3.2 Time Warming Allocation Engine Sequence Diagram

Figure 2 illustrates the data flow of the proposed Time Warming Allocation Engine (TWAE) in the form of a sequence diagram. It represents the overall flow of the TWAE, which is repeated each time a new request is made by the user.

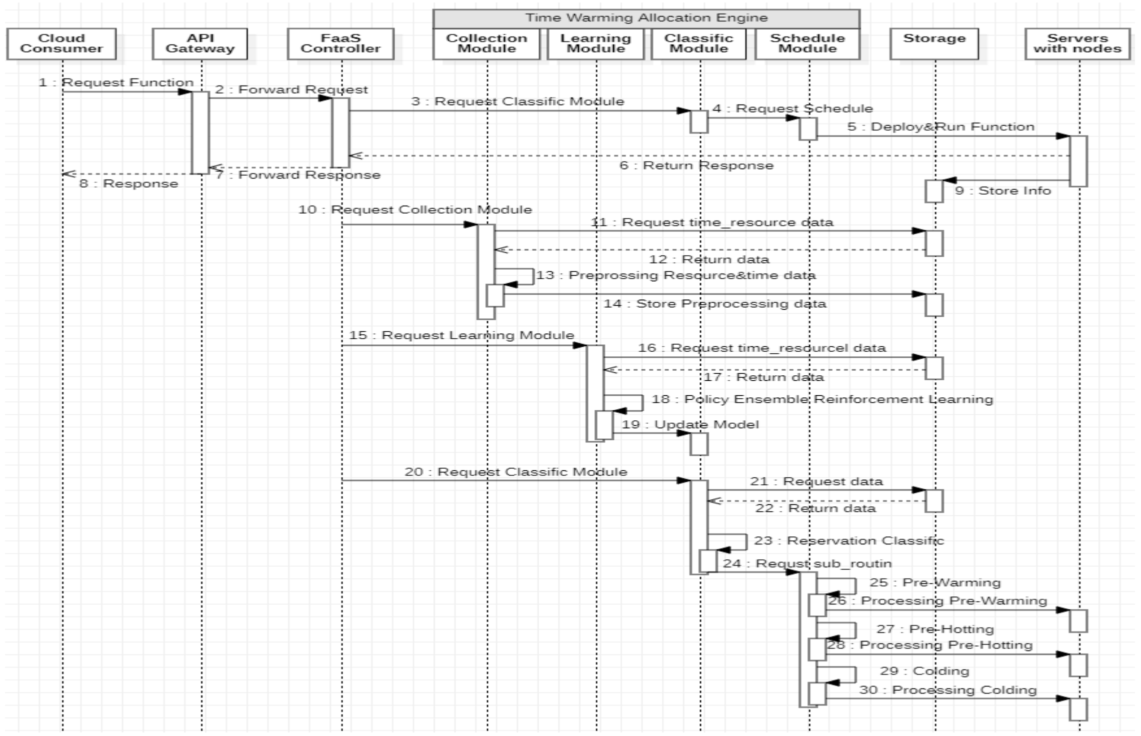


Figure 3. Time Warming Allocation Engine Sequence Diagram

1 ~ 2. Request Function: When a cloud customer requests a function through the provided API, the request is passed through the API Gateway to the FaaS Controller.

3. Request Classification Module: The FaaS Controller requests the Classification Module to determine the appropriate environment for the requested function. The Classification Module selects suitable resources for the function and requests the Schedule Module.

4 ~ 5. Request Schedule: Based on the values received from the Classification Module, the Schedule Module calls for the deployment and execution of the function in a container.

6 ~ 9. Return Response & Store Info: Upon completion of the container's task, the response for the requested function is returned to the cloud customer, and the generated data is stored in storage. Additionally, the container that has completed its operation is terminated.

10 ~ 14. Request Collection Module: Periodically, the Collection Module checks the validity and completeness of the data in storage, performing preprocessing. Clustering may be added during the preprocessing process.

15 ~ 19. Request Learning Module: Periodically, the Learning Module uses the preprocessed time-based resource statistics data in storage to train a model using Policy Ensemble Reinforcement Learning. The trained model updates the Classification Module's model for future classifications.

20 ~ 30. Request Classification Module: Every hour, the Classification Module is called for Pre-Warming, Pre-Hotting, and Colding. This process pre-creates dependent services or terminates previously created ones.

3.3 Time Warming Allocation Engine Algorithm

Algorithm 1 lists the execution steps of the Time Warming Allocation Engine according to the sequence diagram. This method was implemented in Python for experimentation with Kubernetes and Knative.

Sequence Algorithm 1 TWAE: FaaS Controller

```
1:  INPUT : Cloud Consumer Request Info
2:  BEGIN:
3:  # BEGIN BY Cloud Consumer Request
4:  Sub_routine_flag, Consumer_request_detail = FWAE.Formatting(Consumer_Request_Info);
5:  # Consumer_Request_Info was the request information from the cloud consumer.
6:  # Split the required data through data formatting.
7:  # The cloud consumer referenced the API defined by the cloud provider to request the desired
    function. The request data from the cloud consumer was represented as.
8:  FWAE.Classific_Module.Request(Sub_routine_flag[0], Consumer_request_detail);
9:  # The arguments of the Request were Sub_routine_flag and Consumer_request_info.
10: # Sub_routine_flags: 1 referred to Request-Classific, and 2 referred to Reservation-Classific.
11: FWAE.Collection_Module.Request();
12: # When the requested function task was completely finished, the data was preprocessed using
    the Collection Module and then stored in the Storage.
13: # Collection_Module can be used periodically to update or whenever a user's request is
    processed.
14: IF Every Hour on the Hour:
15:     FWAE.Classific_Module.Request(Sub_routine_flag[1], NULL);
```

```
16: # The time could vary depending on the configuration.
17: # In the called Classific Module with the flag value set to 2, each subroutine operated in
    sequence: Pre-Warming, Pre-Hotting, and Colding.
18:     FWAE.Learning_Module.Request();
19: # When the Learning Module was called, it trained the model using Policy Ensemble
    Reinforcement Learning.
20: # If training was completed, it updated the existing model in the Classific_Module through the
    internal function Update_Model(Model).
21: END
```

4. EXPERIMENTS AND RESULTS

In this section, the environment for analyzing the performance of the proposed Time Warming Allocation Engine was established using Kubernetes and Knative. The simulation structure consists of three host servers, each with the same configuration as shown in Table 1. The scheduling methods were implemented using Python and Ruby. For the experiment, data from Azure Function 2019 [11] was preprocessed to create time-based simulation data. The schema structure of the simulation data includes Timestamp, Service Type, Execution Time Percentage, Memory Usage Percentage, among other attributes. Additionally, this study referenced widely used open-source FaaS benchmarking [12] to incorporate different runtime behaviors and resource requirements in a microservice manner.

Table 1. Experiment Specification

Item	Details
Processor	Intel(R) Core(TM) i9-12900 @ 3.20GHz - 16
Memory	64 GB
Storage	2 TB HDD
Network	10 Gbit/s network card
OS	Ubuntu 20.04

In this experiment, the evaluation was conducted based on the recently proposed heuristic-based approach ENSURE [13], to assess whether the system can adapt to the varying request frequencies of cloud customers over different time periods, maintain Quality of Experience (QoE), and optimize resource utilization.

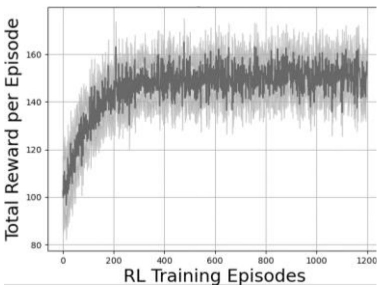


Figure 4. RL Training

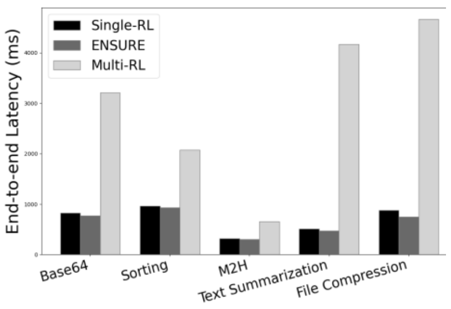
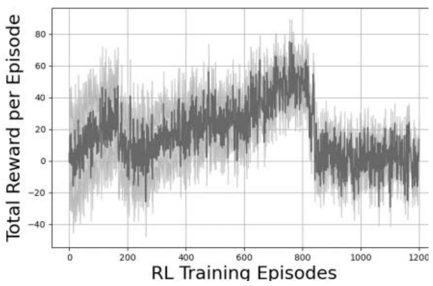


Figure 5. RL Agent 99% End to end Latency



Figrue 6. RL Training in Multi Tenant

To verify the convergence of the model trained with Policy Ensemble Reinforcement Learning, the previously described data was used for training. The evolution of rewards per episode was analyzed, as shown in Figure 4. Convergence was observed after approximately 200 episodes.

For performance evaluation, the trained RL model was compared to the threshold-based autoscaler ENSURE. ENSURE's approach used parameters and thresholds as specified in its respective paper. The comparison results indicated that the RL model trained using the proposed method maintained approximately 20% higher CPU utilization than ENSURE. This is illustrated in Figure 5. The reason for this result is that the traditional ENSURE method allocated more resources to containers compared to the RL model-based approach proposed in this paper. This demonstrates that the Policy Ensemble Reinforcement Learning model improved over traditional heuristic-based approaches in complex environments.

Lastly, performance evaluation was conducted by applying the system to a multi-tenant environment. Multi-tenancy refers to an environment where multiple users share the same infrastructure while independently operating their data and applications. Serverless platforms inherently have a multi-tenant environment. It was observed in Figure 5 that the multi-tenant environment had relatively higher latency. This issue arises because various RL agents are making resource allocation decisions, as shown in Figure 6. The decisions of different RL agents led to higher volatility and lack of stable convergence. This indicates the need for support systems for training convergence and performance mitigation when using RL agents in multi-tenant environments.

5. CONCLUSION

This paper proposed the Time Warming Allocation Engine (TWAE) to address the cold start problem in multi-tenant environments using Ensemble Policy Reinforcement Learning. The proposed Ensemble Reinforcement Learning utilized Boltzmann methods and voting mechanisms, trained with historical function-specific statistical data. The generated model classified functions into Hot, Warm, and Cold states and determined the appropriate container resource allocation for each function. The environment was set up using Kubernetes and Knative, with reference to the Microsoft Azure Dataset. This demonstrated that the proposed TWAE is suitable for solving the cold start problem in serverless computing and showed improvements in resource allocation predictions for each function.

However, guaranteeing convergence in multi-tenant applications proved challenging. Relying solely on historical function usage statistics was insufficient to fully consider user requirements. Future work is needed to develop systems that ensure training convergence in multi-tenant environments, address network offload issues, and incorporate hardware design solutions. These efforts aim to more efficiently solve cold start and security problems.

Acknowledgement

This work is financially supported by Korea Ministry of Environment(MOE) Graduate School specialized in Integrated Pollution Prevention and Control Project.

References

[1] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar, "Serverless Edge Computing: Vision and Challenges", In Proceedings of the 2021 Australasian Computer Science Week Multiconference (ACSW '21), 2021, DOI: doi.org/10.1145/3437378.3444367

[2] E. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” arXiv, Feb. 2019, DOI: https://arxiv.org/abs/1902.03383

[3] G. C. Fox, Vatche Ishakian, V. Muthusamy, and A. Slominski, “Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research,” arXiv, Aug. 2017, DOI: https://doi.org/10.13140/rg.2.2.15007.87206.

[4] 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), pp. 1-7, Aug. 2020, DOI: https://10.1109/COINS49042.2020.9191377.

[5] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, et al., "{SOCK}: Rapid Task Provisioning with Serverless- Optimized Containers", In 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), 2018., DOI: https://www.usenix.org/conference/atc18/presentation/oakes

[6] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 2017, pp. 405-410, doi: 10.1109/ICDCSW.2017.36..

[7] A. Das, S. Imai, S. Patterson and M. P. Wittie, "Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement," 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 41-50, May. 2020, DOI: https://10.1109/CCGrid49817.2020.00-89

[8] Golec, Muhammed, et al. "Cold start latency in serverless computing: A systematic review, taxonomy, and future directions." *arXiv preprint arXiv:2310.08437* (2023).

[9] LI, Zijun, et al. The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys (CSUR)*, 2022, 54.10s: 1-34.

[10] M. A. Wiering and H. van Hasselt, "Ensemble Algorithms in Reinforcement Learning," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 38, no. 4, pp. 930-936, Aug. 2008, doi: https://10.1109/TSMCB.2008.920231.

[11] Azure Public Dataset. https://github.com/Azure/AzurePublicDataset/tree/master

[12] Amoghavarsha Suresh, Gagan Somashekar, Anandh Varadarajan, Veerendra Ramesh Kakarla, Hima Upadhyay, and Anshul Gandhi. 2020. ENSURE: Efficient Scheduling and Autonomous Resource Management in Serverless Environments. In International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS 2020). 1–10.