Mitigating Cold-Start Delay using Warm-Start Containers in Serverless Platform

[1]Anisha Kumari, [2]Bibhudatta Sahoo, and [3]Ranjan Kumar Behera
[1,2]*Department of Computer Science and Engineering*, *National Institute of Technology* Rourkela, Odisha, India
[3]*Department of Computer Science and Engineering*, *Birla Institute of Technology, Mesra, Ranchi, India*
anishamishracs@gmail.com, bibhudatta.sahoo@gmail.com, ranjan.behera@bitmesra.ac.in

2022 IEEE 19th India Council International Conference (INDICON) | 978-1-6654-7350-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/INDICON56171.2022.10040220

***Abstract*—Serverless computing is an emerging cloud technology which is preferred due to its quick response time and scale-to-zero features, where the code can run closer to the users to minimize the latency, and users are charged only for the actual amount of resource consumption. The unit of execution in serverless computing is the function that can independently run on a separate container. A container is required to configure and set up for each time a request is issued for function invocation. This requires some time before the actual execution start is known as a cold start. Cold-start delay and the frequency of cold-start highly affect the performance of serverless execution which needs to be addressed. Most of the researchers are trying to address the problem by keeping the container warm for a fixed period of time known as an idle container window, which may not be the right solution for mitigating cold-start delay. If the idle container window is too long it may lead to more resource consumption, which will contradict the scale to zero feature of serverless computing. On the other hand, if the window length is too short, it may not able to handle a large number of invocation requests. In this paper, an adaptive model is proposed to predict the length of idle container window and the requirement for a number of pre-warmed containers in advance. We have used a Deep neural network and LSTM model to capture the previous function invocation patterns, which can be used to pre-determine the length of the idle-container window. The effectiveness of the proposed model is extensively compared with other cold-start mitigation strategies followed in AWS Lambda, Microsoft Azure, Openfaas, and Openwhisk by using four real-world serverless applications.**

***Index Terms*—serverless computing, cloud computing, FaaS, warm start, cold start, Memory Consumption, Reinforcement Learning**

I. INTRODUCTION

Serverless computing is an emerging service model in cloud computing paradigms which was launched by Amazon Web Service (AWS) in 2014 as part of its cloud computing portfolio. Developers can build cloud-native functions without having to worry about configuring and managing resources, applications, and servers. A number of challenges and issues are associated with the performance of serverless execution environment [1]. One of the most significant issue is cold start latency, which resultant from delay in configuration of infrastructure setup for function execution. In serverless environment, the function executions are stateless in nature i.e, state of a function does not retain between the function invocation. Once a function has been completely executed, any resources that were provisioned for it will be released. When a new function invocation request is issued, new set of resources in a container are configured which takes lot of time before the actual execution starts. The duration of this operation is referred to as the cold start delay. In delay-sensitive applications, such as smart health on the Internet of Things (IoT), which requires real-time attention to the patient's present state or online data analysis, this delay is very important. In order to develop a cost-effective serverless environment, service providers came up with number of platforms; such as Google Cloud Functions (GCF) [2], Microsoft Azure [3], and Amazon Web Services (AWS) Lambda [4], OpenFaaS [5], Fission, Openwhisk and OpenLambda [6]. Each of these platforms have their own strategies to the solve the problem of the delayed start-up caused by cold start latency. Most of these platforms keep the container warm for fixed period of time even after the function complete its execution. It can able to reduce the cold-start delay but incur more cost as it consumes more resources even at the idle period. A sererverless application may consist of different types of functions such as I/O intensive, network intensive or CPU intensive [7]. They usually consume different types of resources, hence require different container configuration setup for varying time periods. Therefore fixing the idle container window is not at all a good solution for each of the request. In this paper, an effort has been made to dynamically determine the length of warm container which usually depends on the invocation patterns.

The major contribution of the paper can be outlined as below:

- An efficient adaptive model is presented which leverage the feature of deep neural network and LSTM to capture the past function invocation pattern to mitigate the cold-start delay and the frequency of cold-start.
- Deep neural network has been used to learn the invocation pattern to determined the length of idle container window i.e, the time period upto which the container should be kept warm after a function complete its execution.
- LSTM model is implemented to capture the dependencies between the function request patterns and the number

978-1-6654-7350-7/22/$31.00 ©2022 IEEE

of cold start delay which can be used to determined the required number of warm containers when multiple invocation requests are issued simultaneously.
- An extensive comparison has been made between the proposed adaptive model and the strategies used in other serverless platforms like AWS Lambda, Microsoft Azure, Openfaas and Openwhisk by using four real-world serverless applications.

The rest of the paper is organized as follows: In section II, some of the existing works related to solving the cold-start problem in serverless platform is presented. Section III presents the background details of the presented work which includes serverless architecture, Function-as-a-Service, Cold-start problem etc. The proposed adaptive model is discussed in section IV. In Section V, the experimental details are discussed which includes dataset description, experimental setup and result discussion. The conclusion and future work is presented in Section VI.

II. RELATED WORK

There have been a significant amount of researches made in reducing the cold startup time inorder to enhance the performance of serverless computing. In this section, we have presented some of the potential researches that have been focused on container launch and serverless cold-start. In a paper by Lin et al. [8], authors have investigated how to address the cold start issue in serverless architectures using the Knative Serving FaaS platform. An adaptive container pool scalability strategy (ACPS) was employed by Xu et al. [9] in which pre-launched containers are organized into groups based on the programming languages they support. In addition to this, it is anticipated that the number of cold starts for each category can be dynamically decide the number of container instances.

A technique called Prebaking is introduced by Silva et al. [10] that utilizes the snapshots that were taken before the function execution. In their work, they have presented a prototype for prebaking approach and tested it by comparing its startup time to that of the usual Unix process creation and startup routine. Later on, Solaiman et al. [11] proposed a container management architecture which they called as WLEC which includes both warm and template queues. Each warm container has its unique copy stored in the template queue. Many serverless service providers, such as Amazon's Lambda platform, provide a feature called Provisioned Concurrency [12] from which one can configure the maximum number of request a client can issue at a time. Lee et al., [13] proposed a method to reduce the cold-start delay of workflows which use function fusion when multiple workflows are executed in a parallel fashion. A reaction time model for workflows that accounts for latency is presented, and a method is quickly found that reduces latency and increases response time on a cold start.

Li et al. [14] proposed a model named as Pagurus to reduce the startup latency in serverless platform. Their model is based on sharing the some of the libraries among the warm container. Instead of starting a new container and setting up all the libraries, if some of the packages can be borrowed from other warm-container, the cold-start delay can be reduced significantly. Suo et al. [15], proposed a model named as HotC which based on reusing the runtime environment to reduce the cold-start delay. Unlike other approaches where container are usually kept warm for certain period of time or restarting periodically, HotC maintain a pool of warm containers which can be reused when a new request is issued. In their work, they have used the Marcov chain model [16] to process the runtime history of containers to predict the number of containers in the pool.

III. COLD START PROBLEM

Cold start is the process by which a FaaS platform initializes the environment for a function to be run when the runtime environment for a FaaS application does not already exist [17] [18]. Uploading the function code, initializing the runtime, and building a virtual network are all part of the initialization process. From a few seconds to many minutes, the startup process might take numerous forms. A cold start is particularly expensive for FaaS because more than half of all invocations are completed in less than one second, and the remaining 75 percent are completed in less than three seconds [19]. The drawn-out process of starting from cold might have a detrimental effect not only on the capacity of the system but also on the performance of the FaaS applications. As a result, several platforms make use of an optimization known as a warm-start in order to maintain the environment active after an invocation has been completed. Memory or the page cache is where the "warm state" of the environment is stored. This includes the runtime, any loaded libraries, and any files that have been visited [20]. The environment and the warm state are utilized across consecutive invocations to speed up the function invocation process. However, maintaining a steady supply of warm resources has a price. Too many idle contexts use memory, lowering the system's overall capacity and performance for executing functions. Moreover, experience from various service vendors, like the real-world traces from Azure, demonstrates that only a tiny number of services are used regularly. Below 50% of the functions are used per hour, but less than 10 percent are triggered every minute. AWS Lambda maintains functions warm for 15-60 minutes after an invocation has concluded, which is the minimum amount of time that cloud providers keep the environment alive to avoid wasting resources.

IV. PROPOSED APPROACHES FOR COLD START PROBLEM

Cold start latency is one of the major problem which effects the performance of execution of serverless application. A number of methods can be adopted to reduce the cold start latency in serverless environment. Some of the solutions to the cold start problem could be increasing the allocated memory for functions, using the scripting language instead of compiled language, keeping the shared data in memory or maintaining the pool of warm functions etc. Most of the service provides

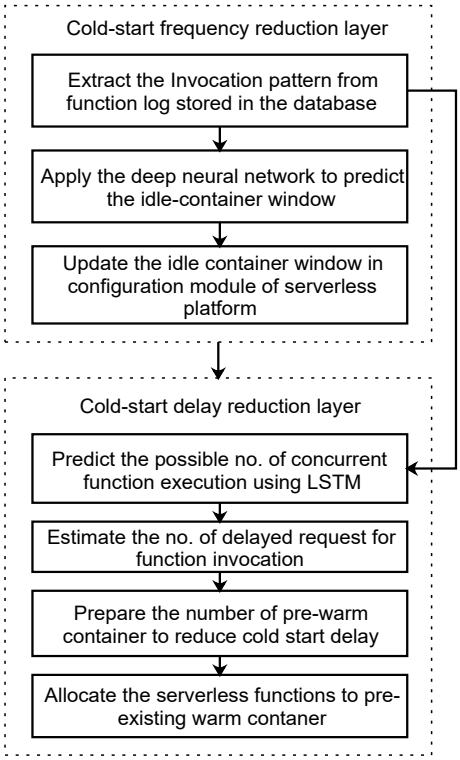


Fig. 1: Proposed approach for mitigation of cold-start delay

uses warm container ie., keeping the container active for some periods of time even after the function execution to reduce the cold start-latency. The period of time the container remain active is known as idle-container window, which usually fixed for most of the serverless platform. The fixed size idle container window is an effcient solutions as the invocation pattern for might be vary for different serverless application. When the idle container window is too long, number of cold start delay might be reduce but the resource consumption will be high, which makes the application too costly. On the other hand, when the container idle window is low and the number of request are very high, the frequency of cold start delay will be very high. Therefore, it is desirable to have an adaptive idle container window which can adjust the window-length based on the complexity of an application and the function invocations pattern. In this paper, an effort has been made to develop a deep neural network based model which can predict the length of idle cold-start window by observing the past invocation pattern. Basically, the proposed model to handle the cold start problem in execution of serverless application consist of two phases:

a) Phase I: Cold start frequency reduction layer: The main objective of this layer is to predict the length of idle-container window which can dynamically change based on invocation pattern of an application. This layer consisting of three steps. In the first step, the invocation pattern available in the function log in the database is extracted. The invocation pattern includes the series of time intervals between the function invocation which had already been recorded. It also includes the status of each function whether it had executed in cold start or warm start mode. These information can be treated as the state of the system which can be used to predict the length of the idle window container. An effective back propagation deep neural network is used to process the state information which provides mean and standard deviation of window length for past few record. There is a trade off associated with window idle length and the frequency of cold start. It can be observed that when the idle window length is large, the frequency of cold start may be reduce but at the same time resource consumption is also increases, which makes the application more expensive. Conversely, when the length of the window is less, the number of clod start will be automatically increases and the resource consumption will be high. It is necessary to maintain a proper balance between resource consumption and the frequency of cold start. In the proposed approach, the back propagation model of neural network is used to predict the window length which can make proper balance between rate of cold start delay and resource consumption. It can be treated as the error in the model which needs to be minimized using the neural network model. Mathematically the error for a time period can be represented as follows:

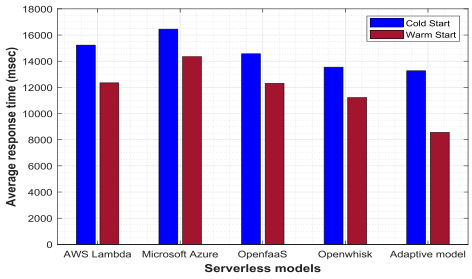
$$Error = \frac{f_c}{T_i} + R_c \tag{1}$$

where f_c is the frequency of cold start delay, T_i is the total number of invocation and R_c resource consumption over a time period. The proposed deep neural network is used to minimize the error rate which consequently provide expected mean and standard deviation of idle window length. This information is then updated in the system through the configuration module of serverless platform.

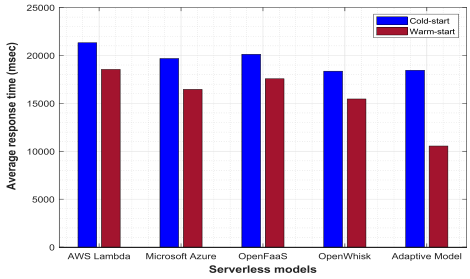
b) Phase II: Cold start delay reduction layer: Another approach to reduce the cold start delay is to prepare the pre-warm container before the invocation. When the request for a function arrives, the pre-warm container can be allocated instead of creating new container. However, there always a limitation exist on the number of pre-warm container, which consume lot of space even if functions are not executing. When the large number of requests arrive at the same time, some of the request must be delayed as it is not possible to allocate warm container to all the functions. One of the solution to this problem could be predetermining the maximum possible request that can be allocated based on the available resources. Based on this information, the same number of warm container can be prepared in advance to reduce the cold start delay. The main objective of this layer is to predict the number of concurrent invocations which can be executed simultaneously over a period of time. In the proposed approach, we have used the LSTM model, which can able to remember the past invocation pattern to predict the future concurrent invocations.

V. Numerical Evaluation

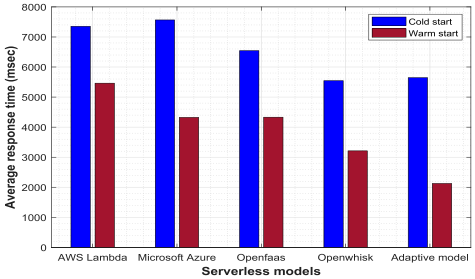
In this paper, an adaptive model is proposed to reduce the cold start delay and the frequency of cold start over



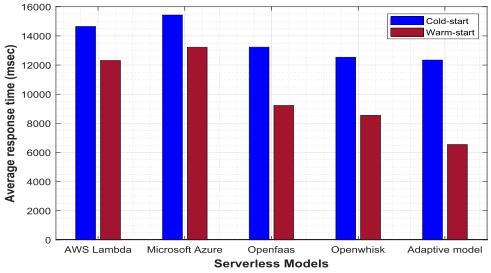
(a) Send-email-ses application



(b) Image-resizer-service application

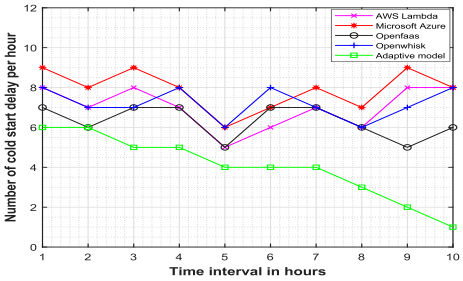


(c) Serverless-form-handler application

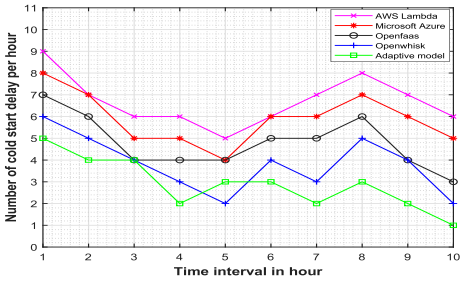


(d) Chromium application

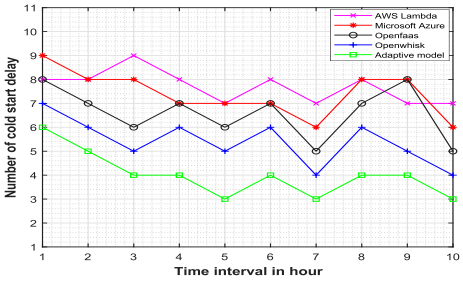
Fig. 2: Rate of change of response time (msec) due to cold-start mitigation strategies



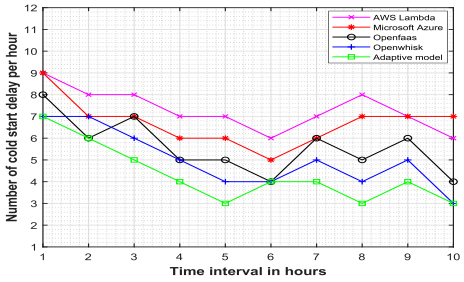
(a) Send-email-ses application



(b) Image-resizer-service application



(c) Serverless-form-handler application



(d) Chromium application

Fig. 3: Comparative analysis in term of number of cold start delay per hour

a period of time. The proposed approach follow two-phase model where used two machine learning models are adopted to predict the idle window container and the number of pre-warm container requirement. In this section, we have discussed the experimental setup, dataset used and the result obtained for different models.

TABLE I: Serverless applications used for experiment

Serverless Applications	Number of Functions	CPU Intensive	IO Intensive	Network Intensive	Description
Send-email-ses	8	4	2	2	An email application that sends email to specified addresses
image-resizer-service	12	5	4	3	Resizing the image recieving from S3 Bucket
Serverless-form-handler	10	4	4	2	Recieve a form submission data via API and store in DynamoDB
Chromium	16	4	4	8	Web-scrapping tool for parsing a webpage

TABLE II: Number of cold start delay for different serverless models

	Send-email-ses	image resizer	Serverless-form-handler	chromium
AWS Lambda	70	67	77	73
Microsoft Azure	79	59	74	67
Openfaas	63	48	66	56
Openwhisk	72	38	54	50
Adaptive model	**40**	**29**	**40**	**43**

A. Experimental setup

In order to validate the performance of the adaptive model, we have considered several other serverless platforms like AWS Lambda, Microsoft Azure, Openfaas and Openwhisk serverless platform which have their own setting to handle the cold-start problem. In our work, Openwhisk is considered as the execution environment, where the proposed machine learning module is integrated to reduce the cold start delay. Other serverless platforms have their default parameter setting to mitigate the cold start delay.

In the first phase, we have adopted deep neural network which is used to minimize the error as presented in equation 1. The structure of neural network in the first phase consisting of four hidden layer, two output layer and one input layer. Each of the hidden layer consisting of 32 number of neurons and a Relu activation function. Softplus activation function is used at the output layer to make it normalized. In the second phase, we have used LSTM model as it has the ability to remember the previous patterns of function invocation, which can be used to predict the possible number of future concurrent invocation requests. The structure of LSTM model consisting of five hidden layers each having 32 number of neurons. RElu activation function is used at each layer.

B. Dataset Preparation

The experiment has been performed by considering four real-world serverless applications, which are consisting of different combination of CPU, I/O and network intensive functions. Each of these function require different configuration setup for container, which is completely taken care by the service providers. The heterogeneous requirement for container setup highly effect the cold start delay and the number of cold start occurrence. The detail of the serverless applications are listed in Table I. All these applications are collected from the AWS serverless application repository [21] which is freely available at the servers of AWS platform. The serverless applications are first deployed on a platform and then number of requests are issued for execution the functions at every one hour time interval. The cold start and the warm start execution are monitored to perform comparative analysis.

C. Performance evaluation parameters

The following parameters are considered to evaluate the performance of the proposed adaptive model.

- Response time: It is defined as the overall response time in execution of serverless applications which are monitored using the log service provided by the platform.
- Cold-start delay: It is defined as the total delay occurs during the serverless execution due to configuration and container setup.
- Frequency of cold start delay: The second objective of the adaptive model is to reduce the frequency of cold start delay.

D. Results and Discussion

The proposed adaptive model has the ability to predict the idle container window by considering the previous invocation patterns. The performance of the proposed adaptive model is validated by deploying and executing four real-world serverless applications on different platforms. The average response time for an application is measured for both cold-start and warm-start mode. In most of the serverless platform, the idle container window is fixed by default. We have measured the improvement of response time for application execution before and after the provision of warm container. It has been observed that the cold start delay is drastically reduced if the length of the idle container changes based on the function invocation requirements.

The comparative analysis of rate of improvement of average response time for different cold-start mitigation strategies is presented in Fig. 2. The comparative analysis of various models for send-email-ses, Image-resizer, Serverless-form-handler and Chromium application datasets are shown in Fig. 2a, Fig. 2b, Fig. 2c and Fig. 2d respectively. From Fig. 2a, it can be observed that the improvement of average response time for adaptive model is around 35% which is a very significant improvement as compared to strategies used in other models.

The comparative analysis for image resizer-service application is shown in Fig. 2b. From Fig. 2b, it can be observed that the improvement in average response time for Adaptive model is around 42%, where as in AWS Lambda, Microsoft Azure, Openfaas and Openwhisk platform the average improvement of response time is 13%, 16%, 12% and 15% respectively in Image-resizer application. In case of Image-resizer serverless application, the model used in Microsoft Azure platform has better improvement in average response time as compared to mitigation strategies used in AWS Lambda, Openfaas and Openwhisk platforms.

The average response time of serverless-form-handler application in both cold-start and warm-start mode is compared in various models as presented in Fig. 2c. It can be observed from Fig. 2c that the response time is more significantly improved

as compared to other application when warm-start containers are used. The rate of improvement for Adaptive model, AWS-Lambda, Microsoft Azure, Openfaas and Openwhisk is found to be 25%, 42%, 33%, 42% and 62% respectively. The adaptive model in serverless-form-handler application has highest improvement in response time as compared to methods adopted in other platforms. Fig. 2d shows the comparative analysis of average response time for Chromium serverless application dataset. The rate of improvement due to warm-container for Adaptive model in Chromium application is found to be 47%, whereas it is 15%, 14%, 30% and 31% for AWS Lambda, Microsoft Azure, Openfaas and Openwhisk platform respectively.

The other objective of the proposed adaptive model is to reduce the number of cold-start by pre-determining the required number of warm-container. The comparative analysis in term of number of cold start delay is also performed to validate the proposed work. Fig. 3, present the comparative analysis of number of clod start occurrence per hour for different methods adopted in serverless platform. In this experiment, we have first deployed the four real-time applications in the server, then we have issued 100 number requests. It can be noted that we have issued request for different combination of functions (CPU intensive, IO intensive, network intensive) over the time interval. We have issued the request for 10 number of invocation per hour for series of ten consecutive intervals. The number of cold start delay over the series of 10 interval is recorded for different platforms.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an adaptive solution which can dynamically change the policy related to number of pre-warm container and time duration for which containers should be kept warm. The proposed adaptive model passes through two phase where two different machine learning algorithms are adopted. In the first phase, deep neural network is used to learn the previous function invocation patterns to predict the idle container window. In the second phase, LSTM model is used to predict the possible number of future request based on which the number of pre-warm container can be prepared in well advanced to reduce the number of cold start delay. From the experiment, it is concluded that cold-start delay and frequency of cold-start can be reduced by using an adaptive model which can change the cold-start handling policy according to the request pattern. It not only improve the number of cold-start but also improve the response time drastically.

In future, this work can be extended to the real-time applications, where huge amount of data are continuously streaming on to the system. Cold-start mitigation strategy can improve the performance drastically especially when data bounds to be process in fraction of time in the serverless platform.

REFERENCES

[1] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo 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)*, pages 205–218, 2020.

[2] Jeongchul Kim and Kyungyong Lee. Functionbench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504. IEEE, 2019.

[3] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure. *New York, NY, USA:: Apress*, pages 3–26, 2015.

[4] Peter Sbarski and Sam Kroonenburg. *Serverless architectures on AWS: with examples using Aws Lambda*. Simon and Schuster, 2017.

[5] Anisha Kumari, Bibhudatta Sahoo, Ranjan Kumar Behera, Sanjay Misra, and Mayank Mohan Sharma. Evaluation of integrated frameworks for optimizing qos in serverless computing. In *International Conference on Computational Science and Its Applications*, pages 277–288. Springer, 2021.

[6] Ranjan Kumar Behera, Sushree Das, Monalisa Jena, Santanu Kumar Rath, and Bibhudatta Sahoo. A comparative study of distributed tools for analyzing streaming data. In *2017 International Conference on Information Technology (ICIT)*, pages 79–84. IEEE, 2017.

[7] Anisha Kumari, Ranjan Kumar Behera, Bibhudatta Sahoo, and Sanjay Misra. Role of serverless computing in healthcare systems: Case studies. In *International Conference on Computational Science and Its Applications*, pages 123–134. Springer, 2022.

[8] Ping-Min Lin and Alex Glikson. Mitigating cold starts in serverless platforms: A pool-based approach. *arXiv preprint arXiv:1903.12221*, 2019.

[9] Zhengjun Xu, Haitao Zhang, Xin Geng, Qiong Wu, and Huadong Ma. Adaptive function launching acceleration in serverless computing platforms. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 9–16. IEEE, 2019.

[10] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. Prebaking functions to warm the serverless cold start. In *Proceedings of the 21st International Middleware Conference*, pages 1–13, 2020.

[11] Khondokar Solaiman and Muhammad Abdullah Adnan. Wlec: A not so cold architecture to mitigate cold start problem in serverless computing. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*, pages 144–153. IEEE, 2020.

[12] Jungae Park, Hyunjune Kim, and Kyungyong Lee. Evaluating concurrent executions of multiple function-as-a-service runtimes with microvm. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 532–536. IEEE, 2020.

[13] Seungjun Lee, Daegun Yoon, Sangho Yeo, and Sangyoon Oh. Mitigating cold start problem in serverless computing with function fusion. *Sensors*, 21(24):8416, 2021.

[14] Zijun Li, Quan Chen, and Minyi Guo. Pagurus: Eliminating cold startup in serverless computing with inter-action container sharing. *arXiv preprint arXiv:2108.11240*, 2021.

[15] Kun Suo, Junggab Son, Dazhao Cheng, Wei Chen, and Sabur Baidya. Tackling cold start of serverless applications by efficient and adaptive container runtime reusing. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 433–443. IEEE, 2021.

[16] Anisha Kumari and Bibhudatta Sahoo. Serverless architecture for healthcare management systems. In *Handbook of Research on Mathematical Modeling for Smart Healthcare Systems*, pages 203–227. IGI Global, 2022.

[17] Daniel Kelly, Frank Glavin, and Enda Barrett. Serverless computing: Behind the scenes of major platforms. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 304–312. IEEE, 2020.

[18] Anisha Kumari, Ranjan Kumar Behera, Bibudatta Sahoo, and Satya Prakash Sahoo. Prediction of link evolution using community detection in social network. *Computing*, 104(5):1077–1098, 2022.

[19] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[20] Shichao Chen and Mengchu Zhou. Evolving container to unikernel for edge computing and applications in process industry. *Processes*, 9(2):351, 2021.

[21] Amazon AWS. Aws serverless application repository, 2020.