A Dynamic Approach To Mitigate Performance And Latency Issues In Serverless Cloud Computing Environment

[a*]Bhatt Jay Mihirkumar, [b]Dr. Rajan Patel

[a*]*PG Scholar, Gandhinagar Institute of Technology Khatraj-Kalol Road, Moti Bhoyan, Ta.Kalol, Dist. Gandhinagar-382721, India*
[b]*Professor,Gandhinagar Institute of Technology Khatraj-Kalol Road, Moti Bhoyan, Ta.Kalol, Dist. Gandhinagar-382721, India*

Abstract

Serverless computing is next generation cloud computing execution model in which the allocation of machine resources is on demand. "Serverless" is in the sense that servers are still used to execute code. However, serverless application developers do not have to worry about capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers. While the Serverless allows engineers to focus on writing core business logic, it comes with the problem of performance and latency issues. Cold start, function placement, and resource allocation are the key aspects of that. Cold starts occur when Serverless functions are invoked before they have been loaded into the system. In this research, we present a dynamic approach to reduce latency and enhance QoS by mitigating the cold start problem and placing the function with proper configuration.

Keywords: Cloud Computing, Serverless, Clod Start, Latency.

1. Introduction

Cloud computing is the on-demand delivery of IT resources over the internet. Instead of buying, owning, and maintaining physical data centers, you can access technology services such as computing power, storage, and databases on an as-needed basis [1]. It is a general term for anything that involves delivering hosted services over the internet. These services are divided into three main categories or types of cloud computing: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [2].

Basically, cloud computing provides server resources on demand, which are located remotely and mostly managed by third-party organizations. The customer rents these resources and follows a pay-as-you-go model. However, in a private cloud, the resources are managed and owned by private individuals or organizations, where the hardware and software resources are on-premise.

Multi-cloud is another concept in cloud computing where organizations use both privately-owned on-premise and third-party public cloud provider services. It provides advantages of both concepts, like the security and privacy of private cloud computing and the flexibility of public cloud computing.

Talking about server resources, server resources are the hardware or software components that help achieve tasks in the computing world. Resources are entities with processes that provide output. It can be software that falls under software as a service or platform as a service, or it could be hardware resources that majorly fall under infrastructure as a service. Apart from that, it also includes network resources like API gateways, NAT gateways, elastic IP addresses, etc. Physical pieces of hardware technology to run the code that make up your server are often referred to as server resources [4].

Cloud computing has many advantages, including public cloud, where we do not have to manage and maintain the servers. It also provides rental services for resources, commonly known as the pay-as-you-go model, where individuals or organizations pay for the resources they use and the time they have been used. Moreover, it is flexible to use cloud computing services as they are provided according to the customer's needs and are configurable according to one's needs. Also, the cost to manage is less as the servers are located and managed by third-party service providers. Apart from that, it provides disaster recovery, high availability, and scalability.

2. Theory Background

2.1 Serverless Cloud Computing

Serverless cloud computing is next generation design architecture in cloud computing industry. It is the advance version of the

* Jay Mihirkumar Bhatt
E-mail address: 210120702008@git.org.in

pay as you go and less resource management concept. It is a cloud computing application development and execution model that enables developers to build and run application code without provisioning or managing servers or backend infrastructure [5]. It provides flexibility to developers and organizations to focus on business logic and code rather than investing time on designing deploying and maintaining sever, hardware, software and network resources.

The developer is unaware of the infrastructure. Instead, the developer has access to prepackaged components. The developer is allowed to host code here, though that code may be tightly coupled to the platform [6]. After developing the application business logic, the developer just has to provide it to the serverless system, which will automatically calculate the required hardware, software, and network resources. Then it will allocate those resources and deploy the application on them, assigning the network ports according to basic configuration done by the developer in script form.

1.1. Advantages

Although cloud computing simplifies the deployment of applications by removing the need for on-premises data centers, managing or configuring server resources at remote locations is still necessary. To overcome this issue, serverless architecture has become the go-to choice, providing the advantages of cloud computing and automation through a serverless framework [7].

1.1.1. Cost effective

The serverless framework is cost-effective because the business logic code is provided to the system, and the system decides how many resources to assign and for what period of time. After executing the business logic, the allocated resources are deallocated. Customers only pay for the resources required to successfully execute the code and for the time the resources were used.

1.1.2. Scalabilitys

The serverless framework is highly scalable. It can scale up when the number of requests increases by increasing resource allocation runtime, and it can scale down when the request count decreases by deallocating resources.

1.1.3. Easy to deploy

One of the main benefits of the serverless framework is its ease of management and deployment, compared with traditional private on-premises computing data centers and conventional public cloud computing. Developers only need to provide a piece of code for the business logic to the serverless framework system. Other deployment-related decisions are taken care of by the framework runtime, which includes resource requirement calculation according to the developer's configuration, resource allocation, deployment of code, etc.

1.2. Challenges And Issues

Despite the serverless framework being a next-generation technology and providing significant advantages over traditional and conventional on-premise and cloud computing technologies, it also faces several challenges [7].

1.2.1. QoS: Quality of Service

Quality of Service is the outcome of the total performance of a service, which includes cloud computing services, internet or computing networks, and specifically, the performance experienced by users. While calculating and processing the required resources to execute the business logic, the QoS can be affected.

1.2.2. Latency

Latency is the amount of time taken by data to go through one node to another on a particular network. Several factors can impact the latency in serverless framework, including cold start, resource allocation, and function placement.

1.2.3. Cost and Pricing Model

The automation of resource allocation and deployment also comes with a different pricing model structure, which is different from the pricing structure of conventional cloud computing and on-premises data centers. Sometimes it can be complicated due to the resource allocation of different business logics.

1.2.4. Resource Limits

In the serverless framework, users have to configure resource allocation strategies and limits before deploying the code. If the rate of requests suddenly increases in the future, the configured resources may not be sufficient. In such cases, the scaling process becomes slow and affects the latency directly or indirectly.

1.2.5. Cold Start

A cold start occurs when serverless functions remain idle for some time, and the next time these functions are invoked, a longer start time is required. This delay occurs due to the provision of a runtime container to execute the functions [11]. Launching a new container, setting up the runtime environments, and application-specific initialization collectively contribute to the cold start latency [9].

1.2.6. Function Placement and Resource Allocation

The geographical location of the function and the resources allocated to execute it both play a vital role in reducing latency and improving performance [8]. The latency and function placement are directly proportional to each other. The further away the function, the higher the latency. Efficient placement of the incoming workload on hosts is required to minimize the provider's capital expenses [9]. The function also requires sufficient resources to execute successfully [10].

2. Literature Survey

2.1. ENSURE: Efficient Scheduling and Autonomous Resource Management in Serverless Environments [4]

Here, the researchers present a scheduler that works at the function level and an automatic resource allocation and management algorithm designed to reduce customers' resource costs without impacting users' performance requirements. It follows the classification algorithm, which segregates incoming third-party requests at runtime. Additionally, it automatically scales capacity to prevent cold starts and schedules requests by merging the load on a number of invokers to reuse active nodes.

2.2. Cold Start in Serverless Computing: Current Trends and Mitigation Strategies [6]

This research provides collective details on the latest innovations in mitigating cold start delay. According to research, there are mainly two general approaches to dealing with cold start delay. The primary approach tries to reduce cold start delay by optimizing environments, and the second approach reduces the frequency of cold start occurrences by pinging. In optimizing environments, the general solution is by minimizing container preparation time as well as minimizing dependencies loading time. In the second approach, it works on the phenomenon of not letting functions get into the cold start form. It keeps pinging the function to keep it awake and ready to invoke at any time without high frequency.

2.3. FaaSRank: Learning to Schedule Functions in Serverless Platforms [7]

Normally, serverless function-as-a-service uses conventional scheduling algorithms for dividing function invocations, but it ignores FaaS characteristics like fast changes in resource usage and the lifecycle. This paper focuses on a function scheduler for serverless FaaS platforms based on information provided from servers and functions. It automatically adapts scheduling policies via experience using reinforcement learning and neural networks. The system is implemented in Apache OpenWhisk, which is an open-source FaaS platform. It evaluated performance against other schedulers, including OpenWhisk's basic scheduler on two 13-node OpenWhisk clusters. Real-world serverless workload traces provided by Microsoft Azure have been used. In tests, it sustained on average a lower number of invocations, 59.62% and 70.43%, as measured on two clusters.

2.4. SAND: Towards High-Performance Serverless Computing [8]

This paper focuses on providing minimum latency, enhanced resource efficiency, and high elasticity. To fulfill these properties, it works on two key functionalities. First is application-level sandboxing, and the second one is a hierarchical message bus. In application-level sandboxing, it isolates on two levels, isolation between different applications and isolation between functions of the same application. In the second approach, it creates shortcuts for functions that interact with each other. Moreover, in a hierarchical message bus, it uses a two-level hierarchy, a global message bus distributed across nodes, and a local message bus available on every node. After testing in a commonly-used image processing application, it achieves a 43% speedup compared to Apache OpenWhisk.

2.5. Defuse: A Dependency-Guided Function Scheduler to Mitigate Cold Starts on FaaS Platforms [9]

Currently available scheduling algorithms do not consider the ubiquitous dependencies between serverless functions. This research proposes the solution of cold start potential by using dependencies to mitigate cold starts. It identifies two types of dependencies between serverless functions, i.e., strong dependencies and weak ones. It uses frequent pattern mining and positive point-wise mutual information to mine such dependencies respectively from function invocation histories. By this, it constructs a function dependency graph. The connected dependent functions on the graph can be scheduled to reduce the occurrences of cold starts. The performance evaluation of its effectiveness is done by applying it to an industrial serverless dataset. The results show

that it can reduce 22% of memory usage while having a 35% decrease in function cold-start rates compared with the classical methods.

2.6. A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency [10]

Most of the current research focuses on addressing the issue of cold start in serverless computing by reducing the start-up or cook time of function images or minimizing the frequency of cold starts. Ongoing research has revealed that various factors such as the runtime environment, CPU and memory configurations, networking requirements, and invocation concurrency, affect the cold start of a function. This research proposes a Reinforcement Learning approach to analyze these factors, particularly the function's CPU utilization and invocation patterns, and reduce cold starts by pre-cooking the function thread in advance. The proposed approach communicates with the Kubeless serverless platform and is evaluated using the Apache JMeter to mirror the workload. The agent is compared against the default auto-scale feature of Kubeless and shows the ability to learn the invocation pattern, make informed decisions, and prepare the optimal number of function instances over time.

3. Problem Statement - Aims And Objectives

Based on the literature review, we can conclude that cold start is one of the most significant challenges that needs to be addressed in serverless computing. Cold start directly affects the request-response time, also known as the latency of the business logic, which is a key criterion for high performance. Cold start, function placement, and resource allocation and management are the major factors that contribute to increased latency and low performance.

The primary goal of this dissertation is to improve performance by reducing latency, which can be achieved by mitigating the cold start problem in serverless computing environments. There are several steps and methodologies that can play an important role in reducing latency by mitigating the cold start problem:

- Preloading the required dependencies and libraries needed to execute the code
- Provisioning the function separately, keeping it ready to be invoked rapidly
- Scheduling an algorithm to pre-cook the function business logic according to historical data of function invocations

4. Methodology - Proposed Work

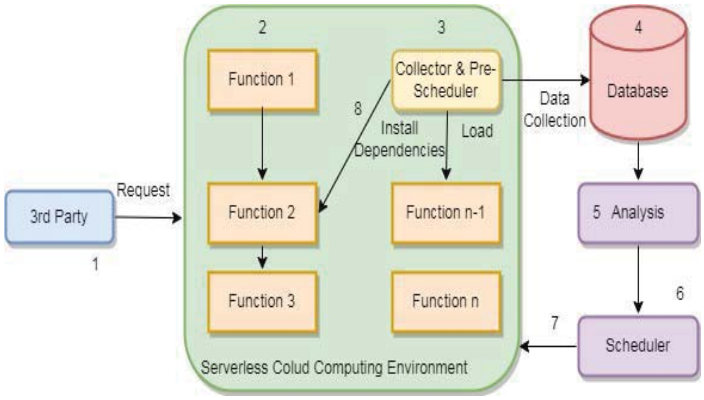

Fig. 1. Proposed Work.

Several approaches have been attempted to mitigate the cold start problem, which can reduce latency and improve performance. Keeping the function ready to be invoked at any time, or pre-cooking it before potential invocation using historical data, were the major aspects of these approaches. We propose a dynamic and hybrid approach that handles it in a novel way. It is a combination of pre-cooking the function and getting it ready to be invoked by pre-loading libraries and dependencies based on strong dependencies and historical data. Strongly dependent functions will be ready once previous functions in the chain are invoked, while weakly dependent and edge functions will be mainly dependent on the historical data of invocations.

The sequence of execution of different components to achieve this would be as follows:

- Initially, a 3rd party request will come in
- The edge function will be invoked and it will invoke the subsequent functions
- Simultaneously, we will collect trace data of different function invocations with their metadata, including start and end timestamps, resource allocation and requirements, etc.
- We will store the data in a database
- We will analyze the previous function invocations' data of sequence and time of function invocations
- We will analyze the strongly, weakly, and edge functions and then schedule the function before the next invocation.
- We will pre-cook the function by installing dependencies and libraries before the actual request triggers the function.

Existing research on Cold Start mainly focuses on mitigating the issue by addressing function dependencies, scheduling algorithms, and fusion functions. Some of the approaches include using strong and weak dependencies with a hybrid histogram scheduling and tree policies, edge-triggered and strongly dependent methods. Reinforcement learning-based approaches have also been used, with time series and Q-Learning policies or Hashing, Greedy, Static rank, and Mix-Norm schedulers being applied. By combining both of these approaches, we can effectively mitigate the latency issues caused by Cold Start.

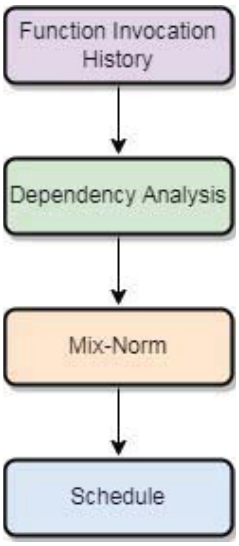

Fig. 2. Analysis And Scheduling.

5. Conclusion And Future Work

After reviewing the next generation technology of Serverless with its advantages and disadvantages, as well as the current issues and challenges in the field, we can conclude that by predicting the next function invocation using historical data and custom configurations, we can pre-cook the function by installing dependencies and loading it into memory, which can be helpful in mitigating the cold start problem in Serverless cloud computing environments. In the future, we plan to further reduce the operational, processing, and architectural overheads, as well as the cold start delay.

References

- Journal articles:

1. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, Philippe Suter, 2017. Serverless Computing: Current Trends and Open Problems.
2. Hassan B. Hassan, Saman A. Barakat, Qusay I. Sarhan, 2021. Survey on serverless computing.
3. Jinfeng Wen, Zhenpeng Chen, Yi Liu, Yiling Lou, Yun Ma, Gang Huang, Xin Jin, Xuanzhe Liu, 2021. An Empirical Study on Challenges of Application Development in Serverless Computing.
4. Amoghavarsha Suresh, Gagan Somashekar, Anandh Varadarajan, Veerendra Ramesh Kakarla, Hima Upadhyay, Anshul Gandhi, 2020. ENSURE: Efficient Scheduling and Autonomous Resource Management in Serverless Environments.
5. Aakanksha Saha, Sonika Jindal, 2018. EMARS: Efficient Management and Allocation of Resources in Serverless.

6. Parichehr Vahidinia, Bahar Farahani, Fereidoon Shams Aliee, 2020. Cold Start in Serverless Computing: Current Trends and Mitigation Strategies.
7. Hanfei Yu, Athirai A. Irissappane, Hao Wang, Wes J. Lloyd, 2021. FaaSRank: Learning to Schedule Functions in Serverless Platforms.
8. Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt, Nokia Bell Labs, 2018. SAND: Towards High-Performance Serverless Computing.
9. Jiacheng Shen, Tianyi Yang, Yuxin Su, Yangfan Zhou, Michael R. Lyu, 2021. Defuse: A Dependency-Guided Function Scheduler to Mitigate Cold Starts on FaaS Platforms.
10. Siddharth Agarwal, Maria A. Rodriguez, Rajkumar Buyya, 2021. A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency.
11. Akash Puliyadi Jegannathan, Rounak Saha, Sourav Kanti Addya, 2022. A Time Series Forecasting Approach to Minimize Cold Start Time in Cloud-Serverless Platform.

Acknowledgement

I owe my deep gratitude to my research guide Dr. Rajan Patel who guided me all along, till the completion of my thesis work by providing all the necessary information, guidance, supervision, moral support, and discussion about my thesis progress with devotion of their valuable time. She gave me a great deal of freedom to choose a research topic, focusing more on my research interests. Without her experience and insights, it would have been exceedingly difficult to do qualitative work.

I would also like to extend my sincere gratitude to Dr. H N Shah, Director of Gandhinagar Institute of Technology, Gandhinagar, for his very humble support, motivation, continuous encouragement for innovation with new ideas, and for providing wonderful positive environment for research work.

I am thankful to and fortunate enough to get constant encouragement, support, guidance and valuable comments from all Faculty members of Computer Engineering Department which helped me in successfully completing work. I heartily thank to all Library Staff Members for providing me great resources of library as well as excellent peaceful environment and Lab Assistants for allowed me to do my research work in labs of Computer Department. At last, I hereby take this opportunity to thank all my colleagues, family and friends and those who directly or indirectly helped me in preparing this thesis work.