National College of Ireland

Optimizing Cold Start Latency in Serverless Architectures through Edge-based Resource Allocation

MSc Research Project
Cloud Computing

Oluwaseyi Ezekiel Ademola
Student ID: 23240334

School of Computing
National College of Ireland

Supervisor: Ahmed Hamza Ibrahim

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Oluwaseyi Ezekiel Ademola
Student ID:	23240334
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Ahmed Hamza Ibrahim
Submission Due Date:	12/12/2024
Project Title:	Optimizing Cold Start Latency in Serverless Architectures through Edge-based Resource Allocation
Word Count:	7739
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Oluwaseyi Ezekiel Ademola
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstract

Serverless computing architecture enables the development and deployment of applications without the management or knowledge of the underlying infrastructure allowing developers to focus on building business logic while delegating resource management operations to the cloud service providers (CSPs). This simplification means that resources are allocated on demand and charges are on a pay-as-you-use basis making it an affordable option for most users. CSPs achieved this execution model by leveraging containerization whereby application codes are deployed in an isolated, stateless environment that provides the necessary resources for execution. Upon completion, these resources are released, requiring the fresh provisioning of resources for subsequent executions. The initialization of a new container, however, incurs a delay referred to as cold start latency, hence limiting the applicability of this model in latency-sensitive applications. This research addressed the problem using a location-aware algorithm that harnessed user mobility patterns and geofence-based prewarming to schedule containers and execute functions closer to the user. The setup was executed through simulation using real-world serverless functions and mobility datasets provided by Microsoft, and the result recorded a 70% reduction in cold start latency and a 76% decrease in resource consumption of function execution during peak and non-peak periods.

1 Introduction

1.1 Research Background

Serverless computing is a Function-as-a-Service (FaaS) model that simplifies application development and execution by abstracting the underlying complexity of infrastructure management, allowing developers to focus solely on building business logic typically composed of isolated, stateless, and event-driven functions. These functions may be triggered by various events, like HTTP requests, database modifications, or messages from IoT devices (Sethunath and Peng; 2022), while the cloud service provider oversees all aspects of the execution environment, including resource allocation, scalability, and monitoring. Upon completion of the function execution, resources are released, guaranteeing users are billed only for the actual compute time that was used, rather than incurring charges based on pre-allocated resources, making it a budget-friendly solution for many businesses. By offloading the management of infrastructure to the cloud provider, serverless computing reduces the operational burden on developers leading to faster development cycles, and provides scalability which allows applications to handle varying workloads ensuring that resources are allocated efficiently during peak times and scaled down during periods of low activity making it suitable for requirements where traffic patterns are unpredictable (MAMPAGE et al.; 2022).

As a result of these advantages, serverless computing architectures have been implemented in applications like machine and big data analytics to manage the processing of image and video analysis as well as the automatic scaling of resources for large-volume data. In web services for delivering backend and API services due to its simplification of big application components into small pieces of functions that are loosely coupled enabling developers to manage and scale applications as they grow in an agile manner (Zhou et al.; 2024). Moreover, the Internet of Things (IoT) is another domain that integrates serverless architectures by leveraging the lightweight nature of serverless functions for efficient

processing of data streams generated by IoT devices, enabling real-time decision-making with minimal latency (Zhao et al.; 2024).

A major component of serverless computing is containerization, though it often operates behind the scenes it is the core technology that provides a lightweight, secured, and isolated environment required to execute functions on demand. In serverless computing, each function is encapsulated within an isolated compute or container that provides the required resources to execute the function such as dependencies, libraries, and runtime environments doing so reducing resource consumption, cost overhead, and scalability of the application. This containerized approach provides consistent execution across various environments, whether in public clouds, private clouds, or edge computing environments (MAMPAGE et al.; 2022). Additionally, function executions are stateless, meaning that upon completion, any containers or resources dedicated to a function will be released, necessitating the provisioning of resources afresh for new or further executions. The initialization process involved in provisioning new resources consists of creating and deploying new instances, downloading and installing the necessary runtime and code, followed by the initialization of the runtime, and subsequently, the function is executed (Zhou et al.; 2024). The duration required for the completion of this process is termed the cold start latency problem.

1.2 Research Motivation

This problem presents a huge obstacle to the efficient implementation of serverless computing in applications that require real-time responses like real-time data processing, healthcare, IoT, and applications that rely on instant user interaction (Htet et al.; 2024). This not only impacts its applicability but also contributes to intensifying user discontent and potentially deterring end-users from serverless solutions highlighting the need for strategies to mitigate these issues and enhance the responsiveness of serverless architectures. Existing approaches to mitigating this problem such as prewarming techniques, and container caching, are limited and introduce additional concerns. For instance, prewarming techniques typically rely on maintaining idle containers in an active state to prevent function re-initialization delays. This technique while somewhat successful results in high resource consumption since containers must be maintained in a warm or active condition even during periods of inactivity. Moreover, precisely predicting invocation patterns to enhance pre-warming continues to pose a significant difficulty, particularly for applications characterized by intermittent or unexpected workloads. Similarly, the container reuse technique is limited to functions with similar configurations reducing its effectiveness in mitigating cold start across varied workloads making it an obvious challenge in multi-tenant serverless setup, as applications often need diverse runtime configurations (Verma et al.; 2024). A key strategy is the integration of Edge computing within serverless architectures. Edge-based resource allocation for serverless applications optimizes the advantages of edge computing by dynamically provisioning computational resources according to user proximity and workload patterns. For example, containers may be prewarmed on edge nodes inside geofenced regions where user activities are anticipated. This minimizes the resource consumption, raw network bandwidth, and latency, and overall reduces the duration needed to initialize serverless functions, guaranteeing improved response times for end-users.

1.3 Research Question

- **Can edge-based resource allocation mitigate cold start latency using a location-aware algorithm?**
- **In what ways does the proposed approach stand out from a baseline serverless platform configuration?**

The root cause of cold start latency can be traced to the on-demand nature of serverless functions, that intend to reduce cost by releasing resources when functions are not active. While this approach maximizes cost efficiency it sacrifices real-time responsiveness due to the time required for re-provisioning. A study by (Zhou et al.; 2024) revealed that cold start latency can be significant with delay reported between 1.3 to 166 times the function execution time, depending on several factors such as the function's runtime environment, size, dependencies as well as the cloud service provider's infrastructure. The time needed to fetch the function code alone can represent a significant portion of the total cold start delay, potentially comprising 47% to 89% of the whole delay. They indicated that initialization time is contingent upon the chosen programming language for function creation; for instance, functions developed in interpreted languages such as Python generally exhibit shorter initialization times compared to those created in compiled languages like Java, which can account for up to 45% of the total cold startup duration.

1.4 Research Objectives

To address the questions and challenges above, this research introduces an edge-based resource allocation technique, where resources are provisioned on-demand, utilizing a custom location-aware algorithm to deliver improved responsiveness for applications and businesses that require real-time processing. The algorithm first maps a geofence area to predict user mobility patterns which are then used to manage the provisioning of resources to execute serverless functions at the edge node closest to the user. It also leverages prewarming functionalities to reduce the occurrence of cold starts during critical periods.

This solution is limited to evaluating the proposed method in contrast with a baseline configuration. While also leveraging real-world datasets to yield realistic results, it remains important to acknowledge the limitations of simulation, especially the difficulty of precisely reflecting the complexities of an actual deployment. Therefore, the research findings will be interpreted within the framework of these restrictions and the limitation will be an area of focus in the future works of this study. The main contributions of this research work are laid out below:

- Propose a location-aware algorithm that triggers prewarming and allocates resources based on user proximity to optimize cold start latency.
- Conduct extensive simulation experiments with real-world datasets to simulate practical serverless behavior and user mobility patterns using official Microsoft Azure functions and Mobility datasets respectively.
- Evaluation of performance metrics in baseline serverless platform against the edge-based solution to verify the effectiveness of the proposed algorithm.

1.5 Research Structure

The rest of this research is organized as follows: Section 2 discusses the existing approaches like function caching, machine learning, and prewarming techniques to reduce cold start latency in serverless computing, analyzing their effectiveness and limitations. Also, specific works that used Edge computing to execute serverless functions closer to the user in line with this research technique were examined. In Section 3 the proposed solution is laid out including the preprocessing of real-world world mobility and serverless functions datasets, as well as tools and technologies used. Section 4 introduces the system architecture, experimental setup, and the location-aware algorithm solution for the allocation of resources at the edge. Section 5 discusses the implementation of the proposed solution followed by the analysis of the trace-driven simulations to assess the performance of the proposed algorithm empirically in Section 6. Finally Section 7 addresses the future work and conclusion.

2 Related Work

This literature focuses on cold start latency in serverless architecture and its integration with Edge computing platforms. Serverless architecture is a modern-day approach that allows the developer to run services without requiring the management of the underlying infrastructure. In recent years the collaboration between serverless architecture and Edge computing has shown promising potential as well as multiple setbacks (Hu et al.; 2023). Cold start latency can be described as one of the prominent issues that can limit the instantaneous initialization of a new execution environment for serverless functions. The initialization process involves multiple steps including the function codes and dependencies. Setting up containers can also be a time-consuming and demanding task in terms of resource allocation. These steps can result in delays, which in turn can undermine performance (Gackstatter et al.; 2022). This section explores the existing solutions to analyse the cold start latency issues and different mitigation approaches proposed by past research work. It will further analyse their methodologies and identify the limitations of their solutions. The research gap and niche also focus on these limitations and propose a solution that can ensure a feasible collaboration between serverless architecture and edge computing platforms.

2.1 Cold Start Latency

Cold latency has both direct and indirect impacts on the serverless architectures. It affects different key performance indicators such as throughput, response time, and availability. Businesses that use the serverless platforms are already paying a high cost. Cold start handling can cause additional resource consumption leading to cost inefficiencies (Hu et al.; 2023). A systematic literature analysis evaluated past research works on cold start latency and its impact on cloud-based architecture. The authors evaluated 23 research papers focusing on multiple aspects such as the impact on QoS parameters and factors influencing the latency parameters. The results from this study were classified into multiple solutions based on caching, AI/ML techniques, and optimization. The review described edge computing as a suitable mitigation technique as the functions are deployed closer to the end user proving its effectiveness in overcoming latency. These strategies can be used to predict workload and initialize containers before future requests

(Golec et al.; 2024).

Function code loading is another reason behind the cold start-up time. During the memory setup, all function codes and their relevant dependency must be loaded into the memory. Some functions have many dependencies and loading them can be a time-consuming task. (Lee et al.; 2021) in their research article proposed a function fusion approach to deal with cold start latency. In their proposed strategy, combining multiple functions into a single function can help in eliminating cold starts. However, one issue with this strategy is that parallel functions will become significantly slow due to the sequential execution. Security overhead is another reason behind cold start latency. Functions like token validation, identity authentication, and other encryption procedures contribute to delayed operations. These operations are important for regulatory compliance, but they still play their role towards cold start latency (Palade et al.; 2019). It is worth noting that even though cold start latency happens for a short period, still it has multiple direct and indirect consequences. These aspects impact the user experience, application performance, and the overall adoption of serverless computing (Liu et al.; 2023).

2.2 Existing Mitigation Strategies

Several researchers have proposed unique solutions that can be employed to deal with the impact of cold start latency. Long Short-Term Memory model (LTSM) is a prediction model that can be used to mitigate cold startup time. The technique was utilized in collaboration with the function scaling and was discovered that the model could deliver up to 95% prediction accuracy when trained on real-world datasets. The author described it as a useful approach that keeps the container warm or alive for a short span of time even after serving an invocation. The research work used different container characteristics such as invocation frequency and resource footprints to guide caching decisions. LTSM as a predictive algorithm anticipates the resource allocation demands which in turn helps in countering cold starts (Chen et al.; 2023).

Another effective strategy that can be used to mitigate cold starts is the traffic shifting technique. The author of this research work utilized a publicly available dataset containing 1.98 million invocations. Experimental logs generated further datasets enabling the researchers to predict invocations. According to the results, if an infrastructure can direct a small amount of traffic towards a function, it will keep it warm to handle any subsequent requests (Bannon; 2022). Similarly, a research work further examined an approach based on predicting traffic volumes, it was focused on Quality of Service (QoS) parameters. The authors also discussed a similar concept in their work known as scaling latency. They noted that due to cold starts, users often provision GPUs for peak volume traffic. Since GPU hosting is significantly more expensive compared to CPU-Based microservices, it is ideal to start pods and make predictions for the function's latency. This method avoids the need for warm pods (Golec et al.; 2024).

Further mitigation strategy was to identify different trends in serverless computing and their interactions with cold starts. The result showcased that while adopting serverless computing and other modern paradigms, developers don't have to manage their servers. The research paper also discussed several mitigation strategies that can be used to deal

with cold starts. The paper proposed optimized runtimes and hybrid architectures as a mitigation strategy. The author concluded that I/O-intensive functions have a longer cold start-up time compared to CPU-intensive functions. Customized runtime with adjusted bootstrapping can enhance the overall infrastructure performance (Eismann et al.; 2020). As far as the comparative analysis of these mitigation strategies is concerned it can be seen that even though pre-warming can eliminate cold starts, it can prove to be a cost inefficient solution. On the other hand, reactive measures are a more cost-efficient mitigation technique. However, under some conditions, these measures might not be able to completely mitigate the high demand scenarios (Zhao et al.; 2023).

2.3 Serverless Edge Computing

Edge computing is a networking philosophy that creates a link between data sources and computing. This process reduces the latency and bandwidth requirements. (Liu et al.; 2023) in their work described Edge computing as a method that runs fewer processes in the cloud. This method moves these processes to local places such as servers, IoT devices, or computers. Edge computing can be described as a method that processes data closer to its origin. This in turn allows the functions to process data at greater volume and speeds. The advantage of Edge computing is that it does not require a separate network. It can easily be located on individual router devices (Nastic et al.; 2017). In this section, this research work will explore past literature that analyzed the collaboration between serverless architecture and Edge computing.

A platform design approach evaluated the efficiency and feasibility of cloud and edge networks. The research work was implemented in .NET and Microsoft Azure and explored the reliability and fault tolerance of both cloud and edge networks. The author did find evidence that the integration of serverless computing and edge computing can achieve better performance and efficiency. However, it was seen that the serverless platforms do not care about fault tolerance which is extremely critical in IoT applications. Since edge-based and IoT applications operate in real-time, handling fault tolerance is critical, and ignoring it can be disastrous (Palade et al.; 2019).

An empirical study analyzed the characteristics, motivation, and implementation of serverless applications. The research work describes the collaboration between Edge computing and serverless environments as a double-edged sword. The author stated that using serverless architecture in an Edge computing environment can help reduce energy consumption. This in turn will be effective for cost efficiency. However, serverless architecture adoption and its associated delays can lead to performance degradation and cannot be used for latency-sensitive IoT applications. The author also gave the example of a driverless car and claimed that we cannot prioritize cost efficiency over safety (Eismann et al.; 2020). Another key consideration is the energy and resource efficiency of Edge computing and cloud-based architecture. (McGrath and Brenner; 2017) implemented a prototype design and implementation approach. A prototype was developed in .NET and implemented in Microsoft Azure. The research work highlighted several benefits of serverless architecture and its deployment with Edge computing. This collaboration allows developers to focus more on code instead of infrastructure. Another notable benefit was seamless integration with other cloud-based platforms. As far as the limitations are concerned, the author noted in their work that cloud-based offerings are often conscious

of memory and CPU use. On the other hand, for Edge-based services, energy and delays are major concerns. In order to use serverless architecture at the edge, it is necessary to use energy aware functions.

Cost efficiency has been described as a significant constraint by multiple research works. A cost modeling study evaluated the feasibility of serverless computing and its adoption in edge computing. In their research work, the authors stated that the adoption of serverless architecture will not be suitable for continuous workload. If continuous function invocations are required, then the cost efficiency of serverless architecture will diminish. This will be even more evident if the functions are long-running. Moreover, the obtained latency will also increase during continuous workloads due to stateless executions (Lin and Khazaei; 2020). (Deng et al.; 2020) in their work implemented a theoretical and analytical approach. The paper focused on predictive IoT solutions and edge AI. The authors have utilized methods like model splitting and knowledge distillation. The findings of their research work showed that utilizing cloud-based services with long-running AI tasks will be inefficient in terms of cost. Network security is another concern while establishing collaboration between serverless platforms and edge-based networks. Features like multi-latency and distributed microservices can make their interactions vulnerable. It is important to ensure security guarantees before enabling serverless architecture at the edge (Ahmadi; 2024).

2.4 Comparison Table

Table 1 contains a breakdown of methodologies from related work compared against this study.

Table 1: Comparison table

References	Prewarming Technique	Real Data Simulations	Location-Aware Policy
(Chen et al.; 2023)		✓	
(Bannon; 2022)	✓	✓	
(Golec et al.; 2024)		✓	
(Zhao et al.; 2023)	✓	✓	
(Nastic et al.; 2017)		✓	
(Palade et al.; 2019)		✓	
(Eismann et al.; 2020)	✓		
(Lin and Khazaei; 2020)		✓	
(Hu et al.; 2023)	✓	✓	
(Lee et al.; 2021)		✓	
This study	✓	✓	✓

2.5 Research Gap and Niche

The literature analysis has highlighted how serverless computing has faced several challenges in edge environments. The cold start seemed to be one of the most prominent challenges and this research work aims to propose a location-aware algorithm that will

use user mobility patterns to address the cold start latency issue. Based on the literature analysis it can be seen that resource allocation is quite complicated due to dynamic workloads (Chen et al.; 2023). There are several mitigation methods proposed by existing research work but most of them had significant limitations. The container pre-warming technique is widely adopted in serverless and Edge computing environments but the primary concern with this technique is that it would result in increased cost and inefficient resource utilization. Intelligent scheduling was another solution proposed by several authors but it was seen that it displays poor mobility awareness. This issue becomes even more pronounced in dynamic environments. Similarly, edge cloud collaboration would result in increased network overhead (Golec et al.; 2024; Chen et al.; 2023). Based on these findings, this research work has noted that there is a lack of context or location-aware resource allocation among the existing approaches. The existing mitigation techniques don't account for user mobility, resulting in inefficient resource allocation. Pre-warming which has been described as a reliable mitigation technique, has its associated setbacks. Over-pre-warming results in resource wastage, while under-pre-warming can cause latency issues (Liu et al.; 2023). This research work proposes a location-aware resource allocation algorithm. The algorithm aims to improve the efficiency of resource utilization which mitigation the latency issues. The proposed algorithm can identify user mobility patterns and allocate resources at the edge accordingly. Optimal resource placement will be achieved by identifying the most suitable node to execute a function. The algorithm will also enable the function to decide when to pre-warm a container without unnecessary resource allocation.

3 Methodology

3.1 Data Collection

The initial step in implementing the proposed solution of this research work is rooted in ethical data collection from various sources and figure 1 describes the process flow of the approach taken in the methodology of this research work. The Microsoft Azure Function trace and T-Drive data were identified as suitable input workloads for the simulation of the system carried out, not only for their relevance to the research but also for their inherent ethical attributes as they do not contain sensitive personal information. This careful decision emphasizes the importance of ethical data management and provides an ethical framework for this research. The datasets are provided by Microsoft and sourced from Github and Kaggle, they are made publicly available under the Creative Commons Attribution 4.0 License (Microsoft-Research; 2019) and Microsoft Research License Agreement (Microsoft-Research; 2011) respectively.

The Azure Functions trace, collected in July 2019, is a subset of the dataset analyzed in the study by (Shahrad et al.; 2020) and consists of three main related datasets: Function invocation counts and triggers, Function execution duration, and Application memory allocation distributions which were integral in reflecting the real-world workload patterns of serverless functions in the simulation. They cumulatively amount to approximately 1.49 million rows of data. In the same way, the T-Drive data is a subset of the dataset used in the study carried out by (Yuan et al.; 2010, 2011). It contains GPS traces of 10,357 taxis that were collected between the period of February 2 to February 8 2008 in Beijing totalling around 15 million geo-points. Each geo-point contains fields like taxi

id, date time, longitude, and latitude which were crucial in simulating real-world user mobility patterns for the proposed algorithm.

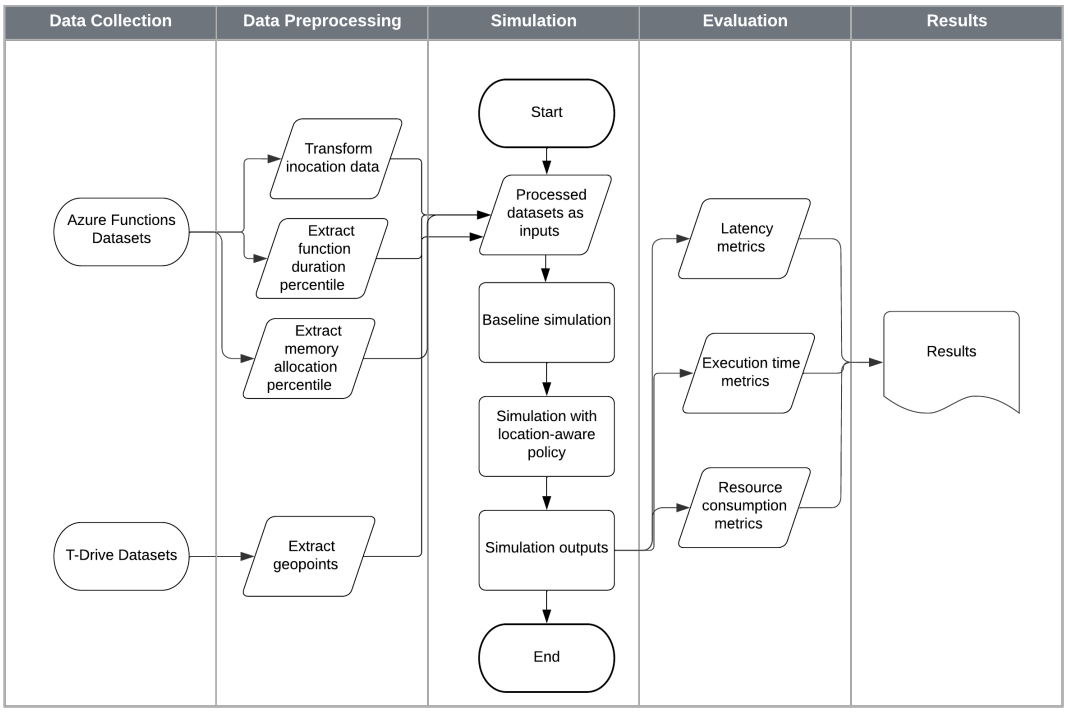

Figure 1: Methodology Process Flow Diagram

3.2 Data Preprocessing and Transformation

After carefully gathering the data required for the research work, the next important phase is to process the data into a structured format in line with this research analysis and simulation goals. The preprocessing approach taken in this research work was informed by insights drawn from a previous study by (Shahrad et al.; 2020) where they highlighted how periodic patterns, function execution patterns, and memory utilization impacted resource management policies. Likewise, this research expanded on these principles to design a reliable data pipeline for preparing the Azure Function and T-Drive datasets in a suitable way for modeling and simulation in CloudSim and iFogSim as well as the proposed location-aware algorithm aimed at optimizing cold starts at the edge. The data processing was conducted using Python libraries like Pandas and Numpy.

The Azure Functions trace consists of three categories of log files which will be further discussed in detail. The grouping is based on a set of related characteristics of the serverless workload which are listed below:

- Function invocation counts and triggers
- Function execution time distributions
- Application memory allocation distributions

Early exploration of the serverless workload granularity revealed that the functions in the Azure Functions trace are fundamentally grouped into applications, with all data pertaining to an application and its functions encapsulated within that application. Applications are characterized as the unit of resource allocation indicating decisions about functions warm-up are done at the application level, and memory allocation is assessed by the application, rather than by function. The trace schema defined unique fields for each log file category, but common key fields were used to correlate related functions and applications, as detailed below:

- 'HashOwner'- unique id of the application owner
- 'HashApp'- unique id for application name
- 'HashFunction'- unique id for the function name within the app

3.2.1 Function Invocation Distribution

The Function invocation trace contains 14 log files, one for each 24 hour period, with the schema key fields including the Triggers and the Invocation data columns in addition to the previously described common fields present in all log files. The Triggers define the event type responsible for executing the function based on different invocation sources (orchestration, http, queue, event, etc). The Invocation columns provide minute by minute breakdown of function invocation that occurred in a day. Each of the invocation data columns is numbered from 1 to 1440 representing a minute within a 24-hour period (1440 minutes total). To simplify the modeling of the workload pattern for the simulation of the proposed system, the invocation distribution was aggregated on an hourly basis by summing up every 60-minute interval (columns 1–60, 61–120, and so on) to get 24 hourly invocation distributions.

$$H(h) = \sum_{t=60h+1}^{60(h+1)} I(t)$$

Where:

- $H(h)$: Total invocations in hour h.
- $I(t)$: Number of invocations at minute t.
- h: Hour index, ranging from 0 to 23.
- t: Minute index, ranging from 1 to 1440.

The analysis further revealed significant variations in the invocation distribution across the hours of the day which led to the identification of peak hours for the functions, conducted by calculating the hours with the highest invocation count. The top 3 peak hours were selected and combined in a new column for the research simulation. This is important for understanding when to pre-warm containers, as frequent invocation patterns can indicate a need for pre-warming to reduce latency.

$$\text{Peak\_Hours} = \text{argmax}_{h_1,h_2,h_3}\{H(h) \mid h \in [0, 23]\}$$

Where:

- Peak_Hours: List of the three hour indices (h_1, h_2, h_3) with the highest values of $H(h)$.
- $H(h)$: Hourly invocation count.

3.2.2 Function Duration Distribution

Similar to the invocation distribution, the Function duration trace contains serverless function execution time over a 24-hour period for 14 days. Each of the data represents the performance of a specific function within an application for a given day offering both aggregated and percentile-based execution metrics. The schema provided the following distinct metrics all within 24 hours:

- 'Average' - The average execution time in milliseconds across invocations.
- 'Count' - The total number of function invocations.
- 'Minimum' & 'Maximum' - The shortest and longest execution time recorded within 24 hours.
- 'Percentile Average' - Granular breakdown of the execution time from 0th, 1st, 25th, 50th up to the 100th percentile.

The data were modified by selecting a subset to model execution time, simplifying the simulation setup while preserving data authenticity. The 'Average' column was selected and renamed to 'Average Execution Time'; the 'Minimum' and 'Maximum' columns were renamed to 'Minimum Execution Time' and 'Maximum Execution Time,' respectively. The 'Percentile Average 50' was renamed to 'Median Execution Time,' and the 'Percentile Average 99' was renamed to 'Peak Execution Time.' The columns were renamed to be more descriptive for ease of use in the simulation.

3.2.3 Memory Allocation Distribution

The Memory allocation distribution provides insight into the memory usage metrics of each application over a period of 24 hours, represented as averages and percentiles. The key columns in the schema are SampleCount which contains the total number of memory samples recorded for an application and the AverageAllocatedMb which provides the average memory consumption trends of each application. They are broken down into the 1st percentile up to the 100th percentile. The memory allocation data for each of the functions were analyzed and a subset of the data was selected to model the resource allocation patterns required for the research simulation. Specific columns such as AverageAllocatedMb_pct50, AverageAllocatedMb_pct95 and AverageAllocatedMb_pct99 were extracted and modified to Median memory usage, High memory usage, and Peak memory usage respectively for easier use.

3.3 Simulation Environment

The simulation environment of this research was based on the iFogSim simulation toolkit. It is an extension of the widely adopted CloudSim simulator commonly used in the modeling and simulation of cloud computing environments. It allows for the definition of infrastructure, service placement, and resource allocation policies for edge computing

without prior knowledge of CloudSim simplifying its usability. Its extensive use in simulating latency, mobility, and resource management scenarios made it the preferred tool for this study. The components of the simulator follow a layered architecture characterized by namespaces, classes, and entities which are discussed below.

3.3.1 Sensors

Sensors are defined as an essential part of the iFogSim architecture's physical layer that is responsible for handling user or physical interaction with the outside world. These are the devices that generate data commonly referred to as tuples (stream of data) in iFogSim. Data generation is event-driven so when configuring sensors in this framework the interval between these iterations must be specified. Specifically, the GPS sensor was used together with the T-Drive dataset to generate mobility events required for establishing when to prewarm containers and the best location closest to the user to execute the serverless functions.

3.3.2 Actuators

An actuator is a device that takes streams of data generated by sensors and processes them into an output result. It is an important physical component that completes the simulation loop by interpreting sensor events and generating feedback.

3.3.3 FogDevice

The FogDevice is a physical component of the iFogSim simulator that is responsible for hosting application modules. They represent the physical servers or any device that connects the data generation layer over a network in the cloud computing paradigm. Fog devices possess distinct hardware characteristics, including MIPS (Million Instructions Per Second), RAM (random access memory of the fog node), bandwidth, and various computing resources, to mimic actual fog nodes. Fog devices can act as regular datacenter or edge servers as in the case of this research to execute serverless functions and prewarm containers.

3.3.4 Monitoring Service

This layer manages resource availability, utilization, and power consumption. It feeds this information to the subsequent layer, the resource management module, and can be accessible to other devices when required (Yousuf Khan et al.; 2022)

3.3.5 Resource Management

The resource management layer is the fundamental component of the iFogSim layered architecture, it is responsible for managing the system resources, maintaining the quality of service needed by the application layer, and governing the scheduling policies (edgeward or cloud) of applications.

3.3.6 Application

The Application layer is a logical component of the simulator that models the serverless function's execution. This module comprises the AppModule, AppEdge, and the

AppLoop. Each application often consists of several interconnected AppModules, as this module signifies the processing components of the application by providing an isolated environment, like a container or virtual machine, necessary for executing serverless functions or any processing operations (Buyya and Srirama; 2019). It is defined by properties like RAM, CPU cores, storage, and other resources required for processing operations. The AppEdge defines the data flow among different AppModule entities, while the AppLoop is tasked with setting process control loops for additional functionality (Yousuf Khan et al.; 2022).

3.4 Validation Criteria

To obtain definitive conclusions and metrics related to the research findings, it is important to compare the implementation with a baseline configuration as outlined in the research questions. The evaluation of the key metrics with and without the location-aware policy solution will validate the cold start-up time optimization goals in line with the research objectives highlighted in section 1.4 clarifying the evaluation framework of this research. Due to the cost implication and complexity of provisioning a live edge server the iFogSim simulator was used to conduct modeling and simulation of executing serverless functions at the edge network using the Microsoft Azure Functions dataset to mimic a real-world serverless workload. The location-aware scheduling also leveraged the T-Drive dataset for realistic mobility patterns. The evaluation focuses on three key performance metrics:

- **Execution time**: The aggregate time taken for function execution.
- **Cold start latency**: The overhead incurred due to delay in the initialization process of function execution.
- **Resource consumption**: The memory utilization by the serverless functions.

The validation of the experiment carried out was achieved through a common Edge computing environment and the first experiment was carried out against a baseline with default configuration simulating all functions across peak and non-peak hours. The second experiment included location-aware policy components like the custom prewarm module and mobility pattern module to schedule the execution of serverless functions to achieve the research goals and findings.

4 Design Specification

This section discussed the architectural framework utilized in this research work. An overview of the experimental setup is presented, utilizing the tools and technologies highlighted in Table 2. Figure 2 illustrates the system architecture, outlining different layers of the architecture from the physical layer to the edge layer along with the configuration in figure 4.3 and location-aware algorithm 4.5 applied.

4.1 Experimental Setup

The setup necessary for the implementation of the research is seen in the above table. The simulation framework (iFogSim) is a Java-based opensource toolkit that comprises of

different classes (logical components) to model and simulate Fog and Edge environments. The GPS sensor at the data generation layer generates the required mobility events. The physical layer models the infrastructure at the edge network and the cloud. These devices or nodes are configured with a 64-bit system architecture running the Linux operating system. The virtualized environment was created using the Xen hypervisor abstracting the underlying hardware resources required for the containerized execution of the serverless functions.

4.2 Tools & Technologies

Table 2 below outlines the fundamental tools and technologies used to conduct the research experiment.

Table 2: Tools and Technologies

Type	**Tool/Technology**	**Description**
Simulation framework	iFogSim	Simulation framework for fog & edge computing
Programming Language	Java	For implementing simulation environment and location-aware algorithm and resources
Data Preparation	Python, Pandas	Used for preprocessing and visualization datasets
Version Control	Git & Github	Managing source code

4.3 Configuration

The project architecture relies on some fundamental configurations for different layers of the iFogSim architecture. The Cloud infrastructure was provisioned with high MIPS (10000), RAM (40 GB), storage (1 TB), and bandwidth (1 Gbps) while the Edge node was provisioned with lower MIPS (2000), RAM (4 GB), storage (200 GB), and bandwidth (200 Mbps) to reflect real-world representation.

4.4 Architecture Diagram

Figure 2 below illustrates the key components and overall perspective of this study system's architecture.

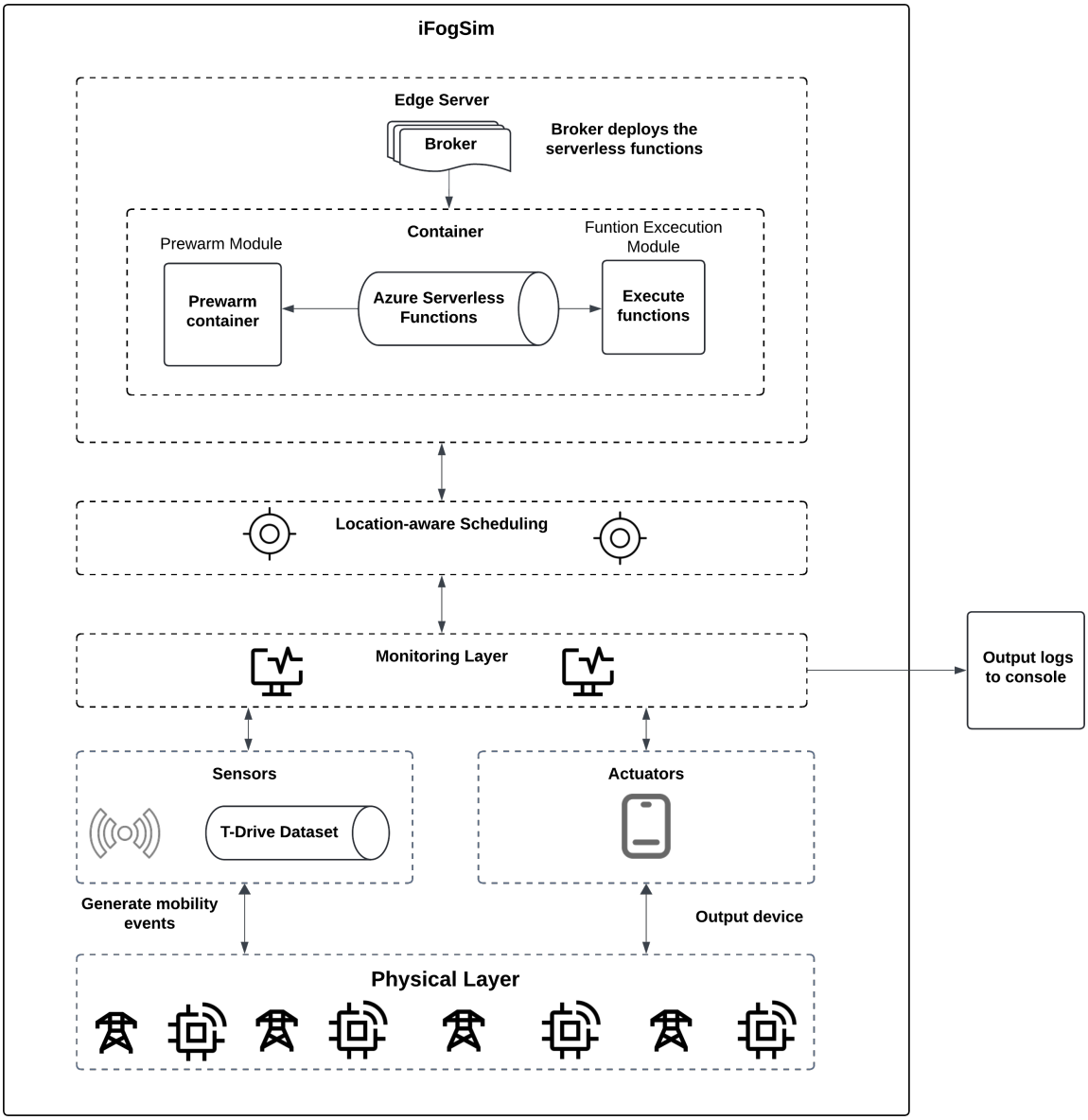

Figure 2: Architecture Diagram

4.5 Algorithm

The algorithm developed and implemented in this study examines some elements of the research carried out by (Sethunath and Peng; 2022), and extends it to align with the research objectives. According to the goals, the location-aware algorithm takes the required input of the T-Drive mobility datasets and Azure functions datasets to model real-world workloads. The prewarm container module was programmed for the reusability of resources in peak hours and the default geofence is defined to indicate best-fit location for function execution based on generated mobility events. When function execution requests are sent it checks if a warm container is available and executes functions instantaneously otherwise functions are rerouted. Idle containers are then released to maintain resource consumption of the edge environment.

Algorithm 1 Location-Aware Algorithm

Require: M: User mobility dataset, F: Serverless functions dataset, G: Geofences, C: Pre-warm containers, E: Simulation environment, PreWarmedContainers: Map tracking pre-warmed containers

Ensure: Log performance metrics: cold-start latency, execution time, resource utilization

Initialize M, F, G, and simulation environment E with fog devices and cloud servers

Initialize PreWarmedContainers and execution counter $execCounter$ to 0
repeat
  for each sensor s in sensors **do**
    $g \leftarrow$ Check if the current location of s is within any geofence in G
    if $g \neq$ null and g is not pre-warmed **then**
      PreWarmContainer(s, g)
    else if $g =$ null **then**
      CleanUpContainers()
    end if
  end for
  for each request r in requests **do**
    if r is not executed **then**
      Calculate and log cold-start or warm-start latency for r
      Execute r on the appropriate edge server
      Increment $execCounter$
    end if
  end for
until all requests are executed or simulation ends
function PreWarmContainer($sensorId$, $geofenceName$)**:**
if $sensorId$ not in PreWarmedContainers **then**
  Create and start a pre-warmed container for $sensorId$ in $geofenceName$
  Add the container to PreWarmedContainers
end if
end function
function CleanUpContainers():
remove unused containers from PreWarmedContainers
end Function

5 Implementation

The implementation of this research work was based on iFogSim. This is an open-source Java-based toolkit that leverages the Sense-Process-Actuate framework and a distributed dataflow model to simulate application scenarios within a fog and edge computing environment (Buyya and Srirama; 2019) with each layer of the environment customizable as they are presented as entities and classes. At the resource management layer, the custom location-aware scheduling policy was orchestrated following the algorithm discussed in Section 4.5 scheduling function execution closest to the user by integrating the mobility module, and prewarm module. These custom modules and classes collectively with other iFogSim modules facilitated the deployment of the research implementation laying a solid foundation for benchmarking the findings against the baseline configuration. Also, the development of the custom scheduling policy was programmed on the IntelliJ IDEA CE IDE, and the simulation was run on the macOS Sequoia operating system with the Apple M1 Pro chip. Further details are discussed in the following sections.

5.1 iFogSim

The first phase of the configuration of the iFogSim simulator started with the integration of the preprocessed datasets using the OpenCSV library. The library was used in the ingestion of the required columns of the datasets to ensure that geo-data, invocation data, memory usage percentiles, execution times, and relevant data were accurately represented as inputs of the simulation. These inputs were then mapped to the iFogSim entities such as Sensors and FogDevices using Java classes. Also, to model the hierarchical topology of the edge environment, different instances of the FogDevice class were created to represent the cloud and edge layers or nodes. The configuration of these nodes was illustrated in Section 4.3 and modeled such that cloud nodes possessed higher computational capacities and edge nodes were optimized instances for latency-sensitive applications reflecting the real-world serverless-edge environment. While the baseline configuration used the default scheduling policy (Time-shared/Space-shared), the location-aware policy handled the resource management layer for the main experiment by instantiating Fog devices based on the input data, this was achieved by linking the geofence data to fog devices such that each function execution is mapped to individual edge nodes based on the the user's location. The Function Profile class was then dedicated to store execution metric summaries which are then output to the console.

5.2 Custom Modules

Extending the iFogSim simulator was necessary for the implementation of the location-aware policy and this was achieved through three main interconnected modules: The Mobility pattern module, the Prewarm module, and the Serverless Application module outlined below.

5.2.1 Mobility Pattern Module

This module is the core component of the location-aware scheduling algorithm. The longitude, latitude, and user ID columns of the T-Drive datasets are first parsed to an entity modeling the user movement pattern as a sequence of geolocations creating the basis for discovering edge nodes that are closest to the user. Geofences are defined around

each of the edge nodes creating the area in which users can be served by the node. The area the geofence covers is controlled by the geo-points (latitude and longitude) that have been predefined. The Haversine formula was used to calculate the distance between the predefined geofence and the user's current location and if the computation is less than the geofence radius the user is considered to be within the node coverage area. Events are subsequently generated based on entry and exit within this area to dynamically allocate resources to run the function and either warm or release the container of the node.

5.2.2 Prewarm Module

The prewarm module was developed to proactively manage resource allocation of the containers on the edge nodes. It receives a geofence entry event from the Mobility Pattern Module and checks if a container is available for function execution in the prewarm pool for the user in the geofence otherwise it initializes a new container based on the memory allocation pattern of the Azure functions datasets and applies a delay of 2 seconds logged as cold startup time. It also monitors the pool for idle containers ensuring that when a geofence exit event is received or the idle threshold is exceeded the containers are released to free up resources on the edge node.

5.2.3 Serverless Execution Module

This module integrates the output of both the mobility and prewarm modules to execute serverless functions. When a function execution request is sent it leverages the mobility module to detect the user's location and best execution node to fulfill the request. If a container is available in the prewarm pool it executes the functions immediately otherwise, it initializes a new container for execution. It also integrates the Functions Profile that maintains all the execution metrics logs from cold start latency to resource consumption metrics used in benchmarking the performance of the research algorithm.

6 Evaluation

The research evaluation was focused on different workload patterns and the scheduling of serverless functions, both with and without the location-aware algorithm, defining two distinct execution environments which are the baseline and the solution the study explored. The initial simulation iteration was conducted on the serverless functions workload during off-peak hours, whereas the subsequent experiment took place during peak hours. The serverless functions were executed hourly according to the workload, with results derived from the cumulative execution summaries of all executed functions. The research findings are categorized according to the validation criteria set out in Section 3.4 and are analyzed as follows.

6.1 Experiment 1

The simulation of the vanilla configuration shows a total cold start latency average of all function execution during peak hours at 395,104,500 ms and off-peak hours results were significantly higher at 2,765,731,500 ms as illustrated in Figure 3. It was examined that the wide margin in the cold start latency could be attributed to the unpredictable invocation of functions leading to a high frequency of cold starts.

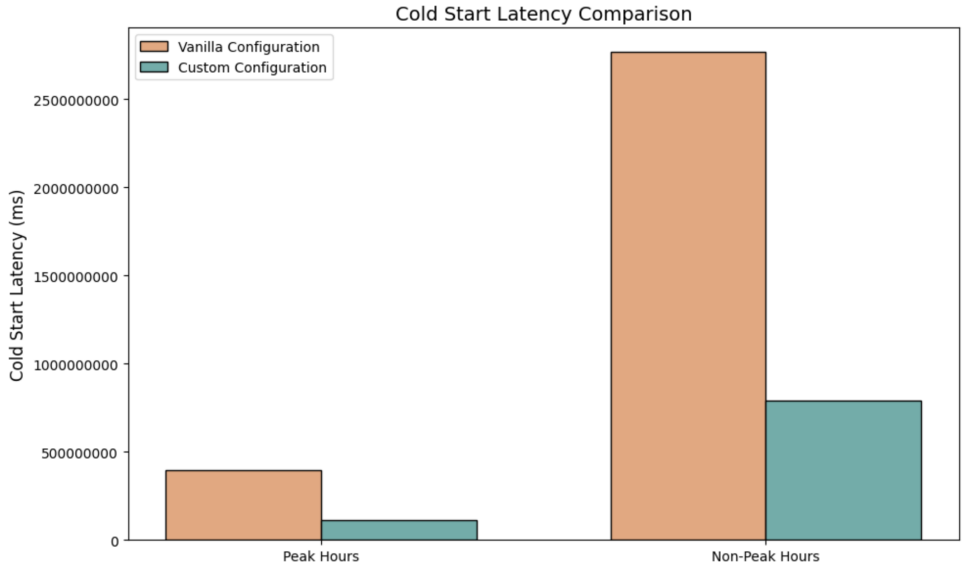

Figure 3: Bar chart of cold start latency comparison

With the location-aware algorithm scheduling the latency drastically reduced to 115,198,950 ms during peak hours reflecting a 70.8% improvement and to 788,191,650 ms during non-peak hours, achieving a 71.5% improvement. This further substantiates that dynamically pre-initializing containers in edge nodes nearer to users was an efficient proactive strategy to optimize cold start latency, and it also demonstrated effectiveness in workloads with unexpected patterns.

6.2 Experiment 2

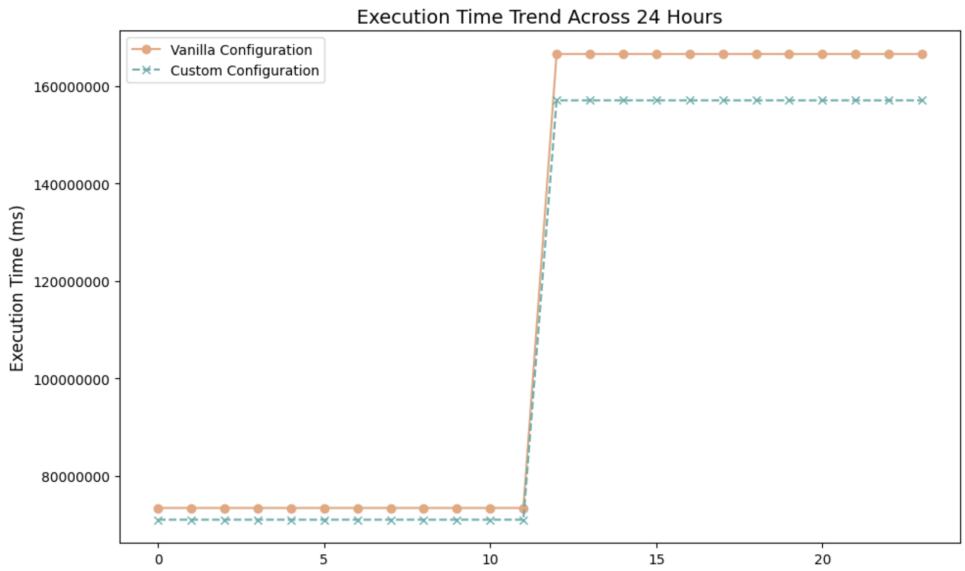

Figure 4: Execution Time Line graph comparison

Figure 4 revealed in a line graph the average total execution times of all functions at 73,419,040.79 ms during peak hours and 166,613,190.57 ms during off-peak hours for the vanilla or baseline configuration. Simulation iterated with the location-aware algorithm highlighted improved execution times at 71,011,427.06 ms during peak hours, reflecting about 3.3% improvement, and 157,094,302.70 ms during non-peak hours revealed an improvement of about 5.7%. This is due to the location-ware algorithm prewarming strategy's ability to trigger execution based on geo-location data ensuring containers are initialized in advance based on the event generated upon entry or exit of geofences around the edge nodes reducing the execution delay for subsequent invocations.

6.3 Experiment 3

The last part of the experiment examines the resource consumption of the proposed solution benchmarked against the vanilla configuration that utilized the default scheduling of the iFogSim edge environment. The simulation of serverless functions conducted with the vanilla configuration resulted in significant resource consumption of 282,619,880 MB during peak hours and 1,965,965,804 MB during non-peak hours. This reflects the default scheduling is often not suitable for handling container initialization and releasing resources, especially during spiky workloads and when containers are not actively in use during off-peak periods. Executing the same functions with the location-aware algorithm reduced resource consumption by 75.5% at 69,092,972 MB during peak hours and about 76% reduction during non-peak hours at 471,277,448 MB as illustrated in Figure 5. The significant improvement can be attributed to the location-aware algorithm's prewarming strategy ability to dynamically allocate or deallocate resources of containers in the edge nodes based on the event generated at entries and exits of the geofence defined around the nodes ensuring resources are not wasted while maintaining readiness for execution of serverless requests.

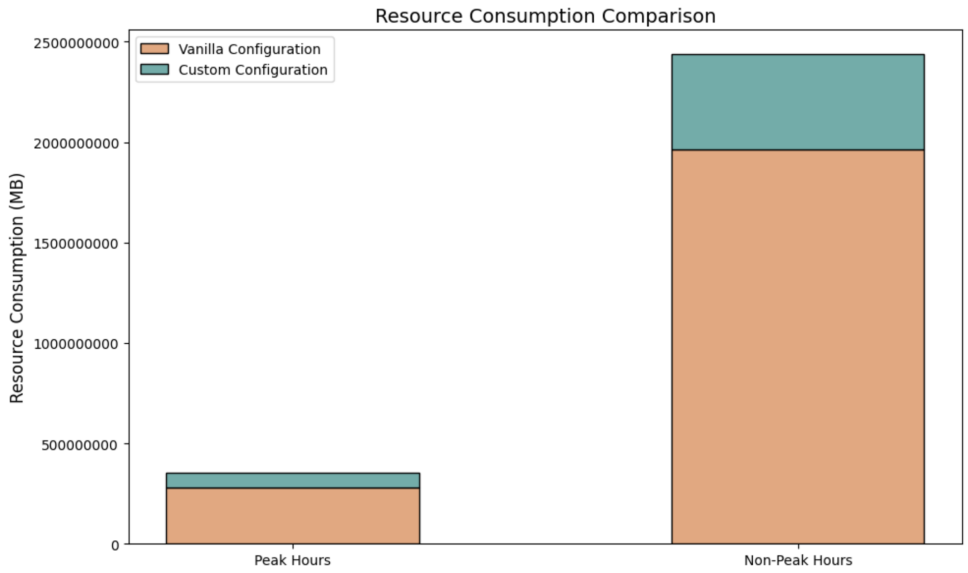

Figure 5: Resource Consumption comparison

6.4 Discussion

The research finding evaluated the location-aware algorithm and revealed significant improvement in cold start latency in workload patterns off and on peak hours. The result demonstrated and justified the effectiveness of the solution the research introduced by an approximate margin of 70.8% at peak period and 71.5% off-peak period addressing one of the most critical challenges in serverless computing in accordance with the research objectives. However, the difference between peak and non-peak hours indicates room for improvement, especially in optimizing pre-warming techniques for sporadic invocation patterns. Functions execution time also saw measurable improvements from 73,419,040.79 ms to 71,011,427.06 ms in peak hours and 166,613,190.57 ms to 157,094,302.70 ms off-peak periods. Although the improvement was relatively small compared to cold start latency, it proved the indirect benefit of reducing cold starts which typically lead to extended execution time. The use of real-world mobility data from the T-Drive dataset confirmed that the results were rooted in realistic contexts reinforcing the credibility of the research findings. The slight improvement revealed also indicates the need for additional optimization for low-traffic workload patterns even though the algorithm performed well during high-traffic periods. The resource consumption saw the biggest improvements with the custom algorithm which is down to the adaptive prewarm technique used in reducing the chances of over-provisioning resources for function execution.

Overall, the simulation experiments highlighted the practicality of the location-aware algorithm in real-world contexts. And while the location-aware strategy aligns well with existing literature advocating edge computing's proximity benefits, the dependence on predefined static geofences used in the implementation of the algorithm may not fully capture real-world mobility complexities necessitating a dynamic geofence approach to accurately model real-time user behavior and network conditions. Additionally, the lack of testing in real-world edge infrastructure limits the broader applicability of the conclusions as this is a common limitation in many simulation scenarios.

7 Conclusion and Future Work

This research aimed to optimize serverless computing by addressing key challenges such as serverless functions cold start latency, execution time, and resource consumption as this has been a major blocker in its adoption especially in latency-sensitive applications such as IoT, gaming, and healthcare. The methodology and architecture were based on the simulation of real-world serverless workload and mobility patterns from datasets provided by Microsoft through the iFogsim simulator toolkit making the research unique compared to other related works. Performance evaluation of the proposed location-aware algorithm was benchmarked against a baseline scheduling policy in the simulator. Key findings included a reduction in cold start latency of over 70% during both peak and non-peak hours, with the most significant improvement observed during peak hours and the research objectives were successfully achieved as the algorithm demonstrated improvements across the validation metrics when compared to the baseline configuration.

Beyond the research achievements, it comes with some limitations. The static geofence model while effective falls short in handling off-boundary users outside the predefined geofences and taking into consideration temporary events in real-world, ever-changing

environments. Moreover, the algorithm performance under real-world edge infrastructure remains untested leaving gaps in applicability evaluation.

Future work would focus on addressing these limitations by developing a dynamic model that integrates real-time mobility tracking, and predictive analytics that could improve the algorithm. Evaluating the algorithm in fault-tolerant scenarios and deploying it across various real-world edge environments would provide deeper insights into its applicability. In conclusion, this research has made an important contribution to the optimization of serverless computing by demonstrating the potential of location-aware scheduling at the edge of the network. Although improvements are necessary, the results confirm the algorithm's effectiveness and can promote further advancements in this field.

References

Ahmadi, S. (2024). Challenges and solutions in network security for serverless computing, *International Journal of Current Science Research and Review* **7**(1): 218–229.

Bannon, R. (2022). *Leveraging machine learning to reduce cold start latency of containers in serverless computing*, Master's thesis, National College of Ireland, Dublin.

Buyya, R. and Srirama, S. N. (2019). *Modeling and Simulation of Fog and Edge Computing Environments Using iFogSim Toolkit*, Wiley, pp. 433–465.

Chen, C., Nagel, L., Cui, L. and Tso, F. P. (2023). S-cache: Function caching for serverless edge computing, *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking*, pp. 1–6.

Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S. and Zomaya, A. Y. (2020). Edge intelligence: The confluence of edge computing and artificial intelligence, *IEEE Internet of Things Journal* **7**(8): 7457–7469.

Eismann, S., Scheuner, J., Eyk, E. V., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L. and Iosup, A. (2020). Serverless applications: Why, when, and how?, *IEEE Software* **38**(1): 32–39.

Gackstatter, P., Frangoudis, P. A. and Dustdar, S. (2022). Pushing serverless to the edge with webassembly runtimes, *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, pp. 140–149.

Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gill, S. S. and Uhlig, S. (2024). Cold start latency in serverless computing: A systematic review, taxonomy, and future directions, *ACM Computing Surveys* **57**: 1–36.

Htet, T. Y., Shwe, T., Mendonca, I. and Aritsugi, M. (2024). Pre-warming: Alleviating cold start occurrences on cloud-based serverless platforms, *2024 IEEE 10th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 66–72.

Hu, S., Qu, Z., Tang, B., Ye, B., Li, G. and Shi, W. (2023). Joint service request scheduling and container retention in serverless edge computing for vehicle-infrastructure collaboration, *IEEE Transactions on Mobile Computing* .

Lee, S., Yoon, D., Yeo, S. and Oh, S. (2021). Mitigating cold start problem in serverless computing with function fusion, **21**(24). Publisher: MDPI.
URL: *https://research.ebsco.com/linkprocessor/plink?id=92b5e55c-16b6-3e89-8d56-464e421a30b6*

Lin, C. and Khazaei, H. (2020). Modeling and optimization of performance and cost of serverless applications, *IEEE Transactions on Parallel and Distributed Systems* **32**(3): 615–632.

Liu, X., Wen, J., Chen, Z., Li, D., Chen, J., Liu, Y., Wang, H. and Jin, X. (2023). Faaslight: General application-level cold-start latency optimization for function-as-a-service in serverless computing, *ACM Trans. Softw. Eng. Methodol.* **32**(5).
URL: *https://doi.org/10.1145/3585007*

MAMPAGE, A., KARUNASEKERA, S. and BUYYA, R. (2022). A holistic view on resource management in serverless computing environments: Taxonomy and future directions., **54**: 1–36. Publisher: Association for Computing Machinery.
URL: *https://research.ebsco.com/linkprocessor/plink?id=7a7e5de8-e3ae-37a6-bac5-b54e95fecce0*

McGrath, G. and Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance, *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, pp. 405–410.

Microsoft-Research (2011). T-drive trajectory dataset. Licensed under the Microsoft Research License Agreement.
URL: *https://www.kaggle.com/datasets/arashnic/tdriver/data*

Microsoft-Research (2019). Azure functions dataset. Licensed under CC BY 4.0.
URL: *https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md*

Nastic, S., Rausch, T., Scekic, O., Dustdar, S., Gusev, M., Koteska, B., Kostoska, M., Jakimovski, B., Ristov, S. and Prodan, R. (2017). A serverless real-time data analytics platform for edge computing, *IEEE Internet Computing* **21**(4): 64–71.

Palade, A., Kazmi, A. and Clarke, S. (2019). An evaluation of open source serverless computing frameworks support at the edge, *2019 IEEE World Congress on Services (SERVICES)*, Vol. 2642, IEEE, pp. 206–211.

Sethunath, M. and Peng, Y. (2022). A joint function warm-up and request routing scheme for performing confident serverless computing, *High-Confidence Computing* **2**(3): 100071.
URL: *https://www.sciencedirect.com/science/article/pii/S266729522200023X*

Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M. and Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider, *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, Boston, MA.

Verma, P., Goel, P. and Rani, N. (2024). A review: Cold start latency in serverless computing, *2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, pp. 141–148.

Yousuf Khan, E. U., Rahim Soomro, T. and Nawaz Brohi, M. (2022). ifogsim: A tool for simulating cloud and fog applications, *2022 International Conference on Cyber Resilience (ICCR)*, pp. 01–05.

Yuan, J., Zheng, Y., Xie, X. and Sun, G. (2011). Driving with knowledge from the physical world, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*, ACM.

Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G. and Huang, Y. (2010). T-drive: driving directions based on taxi trajectories, *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '10)*, ACM, pp. 99–108.

Zhao, K., Zhou, Z., Jiao, L., Cai, S., Xu, F. and Chen, X. (2023). Taming serverless cold start of cloud model inference with edge computing, *IEEE Transactions on Mobile Computing* .

Zhao, K., Zhou, Z., Jiao, L., Cai, S., Xu, F. and Chen, X. (2024). Taming serverless cold start of cloud model inference with edge computing, *IEEE Transactions on Mobile Computing* **23**(8): 8111–8128.

Zhou, A. C., Huang, R., Ke, Z., Li, Y., Wang, Y. and Mao, R. (2024). Tackling cold start in serverless computing with multi-level container reuse, *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 89–99.