An Autoscalable Approach to Optimize Energy Consumption using Smart Meters data in Serverless Computing

Jasmine Kaur[a], Inderveer Chana[b], Anju Bala[a,b]

[a]*Computer Science and Engineering Department , Thapar Institute of Engineering and Technology, Patiala, 147004, Punjab, India*
[b]*Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, 147004, Punjab, India*

Abstract

Serverless computing has evolved as a prominent paradigm within cloud computing, providing on-demand resource provisioning and capabilities crucial to Science and Technology for Energy Transition (STET) applications. Despite the efficiency of auto-scalable approaches in optimizing performance and cost in distributed systems, their potential remains underutilized in serverless computing due to the lack of comprehensive approaches. So an auto-scalable approach has been designed using Q-learning, which enables optimal resource scaling decisions. This approach proves useful for adjusting resources dynamically to maximize resource utilization by automatically scaling up or down resources as needed. Further, the proposed approach has been validated using AWS Lambda with key performance metrics such as probability of cold start, average response time, idle instance count, energy consumption etc. The experimental results demonstrate that the proposed approach performs better than the existing approach by considering the above parameters. Finally, the proposed approach has also been validated to optimize the energy consumption of smart meters data.

Keywords: Serverless Computing, Autoscaling, Q-learning, Performance, Energy Consumption

1. Introduction

Serverless computing is developing as an emerging paradigm in cloud computing designed to make it easier for cloud service providers to use the cloud by handling all the management tasks and optimizing resource utilization, resulting in cost savings and energy efficiency. The main characteristic of the serverless computing approach is dynamic scaling and thus serverless instances have faster startup times as compared to VM-based instances but still show low and unpredictable performance metrics [1]. Serverless computing services are not adaptive to the workloads and use the same management policies for all executed functions in parallel and distributed environments. Adapting the platform to varied workloads has the potential to greatly improve infrastructure cost, performance, and energy consumption[2].
By using the proposed approach, serverless providers can create auto-scalable and predictive platforms, improving the quality of service (QoS) and reducing wasted computing resources. Application developers would also benefit from such auto-scalable approaches by achieving the required quality of service that enables them to migrate more workloads into serverless computing platforms.

1.1. Motivation

The research motivation for this paper is outlined as follows:

- The motivation behind this work lies in the fact that there is a need to develop an autoscaling mechanism for serverless applications that are implemented by Q-learning technique [1],[3],[4]. This is because serverless computing offerings are not adaptive to the workload and use the same management policies for distributed computing applications [2],[5],[6],[7].
- Our motivation for this research is driven by the need to optimize energy usage, particularly focusing on Science and Technology for Energy Transition (STET) applications[8],[9].
- Based on recent studies [10],[11],[12], a lack of evaluation of various performance metrics has been identified, indicating a need to compare the results of the proposed approach with the existing approach based on these evaluation metrics. Additionally, the absence of validation of an auto scalable model on serverless computing platforms further emphasizes the significance of our research.

1.2. Our contribution

The main contribution of this paper is to propose an auto scalable approach to enhance performance and optimize energy consumption in serverless computing.

- The significant contribution of this paper is to propose an auto scalable approach using Q-learning to optimize resource allocation in serverless computing environments. The approach addresses the dynamic nature of workloads by allocating instances to incoming requests and when

Email addresses: jkaur_phd20@thapar.edu (Jasmine Kaur), inderveer@thapar.edu (Inderveer Chana), anjubala@thapar.edu (Anju Bala)

there are no available instances in the pool, it intelligently adds new function instances to meet the requirement. It also incorporates a mechanism to scale down resources if the demand is less than the available resources, ensuring efficient resource utilization and adaptability to varying workloads.

- One distinguishing aspect of this approach is to optimize electricity consumption in real-world applications, specifically focusing on smart meters for residential buildings. This approach addresses the critical need to optimize energy usage, leading to improved energy efficiency and cost savings in serverless computing environments.

- To ensure the applicability, the proposed approach has been verified on AWS Lambda and assessed using various performance parameters such as probability of cold start, average response time, average number of function instances, energy consumption and utilization. Additionally, a comparative analysis has been conducted against the base approach to provide a comprehensive outlook on the effectiveness of the approach.

The remainder of the paper is structured as follows: Key research studies are highlighted in Section 2. Section 3 discusses the preliminaries employed in the proposed approach. Section 4 outlines the details of the proposed Q-learning approach. Section 5 demonstrates the experimental validation of the proposed approach. Finally, Section 6 concludes the paper and presents future research directions.

2. Related Work

Serverless computing has gained a lot of attention from researchers but no auto-scalable approach has been proposed that enhances performance and captures various aspects and challenges in serverless computing. Jawaddi et al. [13], Mahmoudi et al. [10] and Mahmoudi et al. [11] presented queuing theory in recent years to address autoscaling in serverless computing. To dynamically manage the required number of containers, Suresh et al. [14] utilized M/M/k approaching assumptions alongside the square root staffing method. Scaling decisions are determined by assessing the arrival of incoming function requests and the current container count. Shankar et al. [15] introduced an approach for scaling resources in advance by analyzing the number and size of tasks and scaling the number of workers periodically with a predefined factor to meet the task workload. Mahmoudi et al. [16] investigated Markov chain approaches for modeling queueing systems in serverless computing environments i.e. Continuous-Time Markov Chain (CTMC) and Discrete-Time Markov Chain (DTMC). Zhao et al. [17] investigated alternative approaches, such as the simple moving average (SMA) and the exponential moving average (EMA).
Wen et al. [18] presented an in-depth investigation into the challenges faced by developers in building serverless-bassed applications. Perez et al. [19] introduced an open-source framework showing the elasticity and resource efficiency of the framework under varying workloads. Based on the results presented by Kim et al. [20], it is observed that the visibility and predictability of network and disk I/O performance must be made mandatory as in the case of CPU and memory. Enes et al. [21] introduced an innovative platform for dynamic scaling of container resources in real-time demonstrated through the evaluation of big data workloads. The platform showcases increased CPU utilization with less execution time overhead. This scalability is validated using a 32-container cluster, challenging initial perceptions of serverless suitability for Big Data applications. Jackson et al. [22] examined how the choice of language runtime affects both the performance and cost of serverless function execution. The paper introduces a novel serverless performance testing framework, evaluating metrics for AWS Lambda and Azure Functions. The findings show that Python is the optimal choice on AWS Lambda for achieving optimal performance and cost efficiency in serverless applications. Shafiei et al. [23], Singh et al. [24] proposed energy-aware scheduling to reduce energy consumption. The main purpose of this type of scheduling is to put the execution environment or inactive containers in a cold-state mode.

Table 1 evaluated the work related to performance metrics i.e. cost, scalability, cold start, energy consumption, resource utilization and response time in serverless computing. As per our literature review, some authors have considered specific metrics in their studies. Wen et al. [18] evaluated cost, cold start and resource utilization. Perez et al. [19] considered scalability and resource utilization. Kim et al. [20] examined cost. However, as per our knowledge, no author has comprehensively addressed all performance metrics simultaneously. This gap in the current literature highlights the need for further research to evaluate the proposed approach across all relevant metrics.

3. Preliminaries

To develop a comprehensive auto scalable approach for serverless computing platforms, firstly there is a need to understand the functioning and management of function instances in which the computations occur. In serverless computing platforms, each request is managed by a function instance which acts as a tiny server.

3.1. Funtion Instance States

Recent research [10], [11], [12] indicated that function instances undergo six distinct states: initializing, cold-start, warm-start, running, idle and expired as shown in Fig. 1.
When a new request arrives for the first time, firstly it goes to the initializing state. The initializing state signifies the phase during which the infrastructure initializes new instances, including the setup of virtual machines or containers to accommodate increased workload. Instances remain initializing until they become capable of handling incoming requests. The serverless provider does not charge during the initializing state. Based on recent research studies [47],[38], [39], [48], a request that requires initialization steps because of inadequate provisioned capacity is referred as cold start. This process includes deploying a new function, initiating a new virtual machine, or

Table 1: Evaluation of performance metrics in Serverless Computing

Author, Ref.	Year	C	S	CS	EC	RU	RT
Wen et al. [18]	2021	✓	✗	✓	✗	✓	✗
Perez et al. [19]	2019	✗	✓	✗	✗	✓	✗
Kim et al. [20]	2020	✓	✗	✗	✗	✗	✗
Enes et al. [21]	2020	✗	✓	✗	✗	✓	✗
Jackson et al. [22]	2018	✓	✗	✗	✗	✗	✗
Shafiei et al. [23]	2022	✗	✗	✗	✓	✗	✗
Golec et al.[25]	2021	✗	✓	✗	✗	✓	✗
Singh et al.[24]	2022	✓	✓	✗	✓	✓	✗
Grafberger et al.[26]	2021	✓	✓	✗	✗	✓	✗
Li et al.[27]	2022	✓	✓	✗	✗	✗	✗
Bebortta et al.[28]	2020	✓	✓	✗	✗	✗	✗
Gill et al.[29]	2021	✗	✗	✗	✗	✗	✗
Mateus et al.[30]	2022	✓	✓	✗	✗	✗	✗
Marin et al. [31]	2022	✓	✗	✓	✗	✗	✗
Mahmoudi et al.[10]	2020	✓	✓	✓	✗	✗	✓
Yussupov et al.[32]	2019	✓	✗	✗	✗	✗	✗
Van Eyk et al.[33]	2018	✓	✗	✗	✗	✗	✗
Cordingly et al.[34]	2020	✓	✗	✗	✗	✗	✗
Bardsley et al.[35]	2018	✗	✗	✓	✗	✗	✗
Jackson et al.[22]	2018	✓	✗	✓	✗	✗	✗
Rajan et al.[36]	2018	✓	✓	✓	✗	✓	✗
Grogan et al.[37]	2020	✓	✗	✗	✗	✗	✗
Vahidinia et al.[38]	2022	✗	✗	✓	✗	✗	✗
Liu et al.[39]	2023	✗	✗	✓	✗	✗	✗
Fuerst et al.[40]	2021	✗	✗	✓	✗	✗	✗
Mampage et al.[41]	2021	✗	✗	✗	✗	✓	✓
Kaur et al.[42]	2019	✗	✗	✗	✗	✓	✗
Datta et al.[43]	2024	✗	✗	✗	✗	✓	✗
Naranjo et al.[44]	2020	✓	✗	✗	✗	✗	✗
Sarroca et al.[45]	2024	✓	✗	✗	✗	✗	✗
Zuk et al.[46]	2022	✗	✗	✗	✗	✗	✓
Proposed Approach		✓	✓	✓	✓	✓	✓

C-Cost, S-Scalability, CS-Cold Start, EC-Energy Consumption, RU-Resource Utilization, RT-Response Time

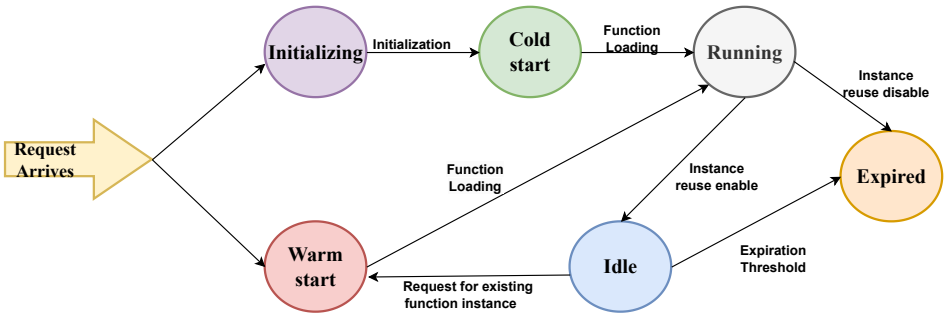

Figure 1: State diagram of function instance

a new function instance on an existing virtual machine, which affects the response time that is experienced by the users. Extensive research is conducted to mitigate cold start in serverless computing [38], [49]. In the warm start, when a new request comes and the platform has an idle instance instead of spinning up a new function instance it will reuse the existing one [10].
Upon receiving a request, an instance transitions into the running state, which processes the request until a response is dispatched to the client. The duration an instance spends in the running state is subject to billing by the serverless provider. After the completion of the request, an instance enters into the idle state. During this period, the serverless platform maintains instances in a warm state for a certain duration to address potential future spikes in workload and developers are not billed for idle instances. If the warm instance in the idle state is not used for some time (expiration threshold), it is automatically shut down and goes to the expired state.
After exploring function instance states, it becomes necessary to explore autoscaling patterns as they offer strategies for dynamically adjusting resources based on the lifecycle of function instances.

3.2. Autoscaling Patterns

Three autoscaling patterns are generally seen in the most widely used serverless computing platforms: scale-per-request scaling, concurrency value scaling, and metrics-based scaling.
In scale-per-request autoscaling, no queuing is involved and it follows synchronous scaling as the new request will be served by one of the idle and available instances which is called a warm start. Otherwise, the platform will instantiate a new instance for that specific request, a process referred to as a cold start. AWS Lambda, Apache OpenWhisk, Google Cloud Functions, IBM Cloud Functions, and Azure Function use this pattern [10],[11],[12],[50],[51]. Concurrency value autoscaling pattern has a shared queue and follows asynchronous scaling in which each function instance can receive multiple requests at the same time. In this scenario, the user defines a maximum limit for the number of requests allowed to enter the instance concurrently. Once this threshold is reached, any new incoming request triggers a cold start, leading to the instantiation of a new function instance. Google Cloud Run, Knative uses this pattern [52],[4].
In metrics-based autoscaling, the system tries to keep metrics such as memory, CPU usage, latency, or throughput within a predefined range. OpenFaaS, AWS Fargate, Kubeless, Azure Container Instances, and Fission use this pattern [16].

3.3. Maximum Concurrency Level

After exploring autoscaling patterns, it is essential to consider the concept of maximum concurrency level, which defines the maximum number of function instances that can be concurrently running. Upon reaching the maximum concurrency level [11], a new request will result in an error status indicating that the server cannot fulfill the request at that moment. This concept underscores the significance of the efficient request routing mechanism that is explained in the next subsection.

3.4. Request Routing

Requests are first sent to recently formed instances to facilitate scaling in. If recently created instances are busy only then the request will use the older containers [7].

4. The proposed autoscalable approach

An auto scalable approach utilizing Q-learning has been introduced in this research, aimed at dynamically adjusting resource allocation in response to real-time demand. This adaptive approach significantly enhances performance and optimizes energy consumption in serverless computing environments. Q-learning, a widely recognized reinforcement learning algorithm in machine learning and artificial intelligence[1], has been employed to empower agents to make decisions within an environment, aiming to maximize cumulative rewards over time [3]. This learning algorithm is particularly effective in situations where the agent doesn't have prior knowledge of the environment and must learn from trial and error [4].
The proposed approach has been demonstrated in Section 4.1. The proposed algorithm is explained in Section 4.2, followed by the detailed calculation of different parameters within the proposed approach in 4.3.

4.1. Proposed Approach

In the proposed approach illustrated in Fig. 2, the scale-per-request auto-scaling strategy has been adopted to efficiently manage the allocation of instances based on incoming requests. This dynamic approach ensures that the system scales up or

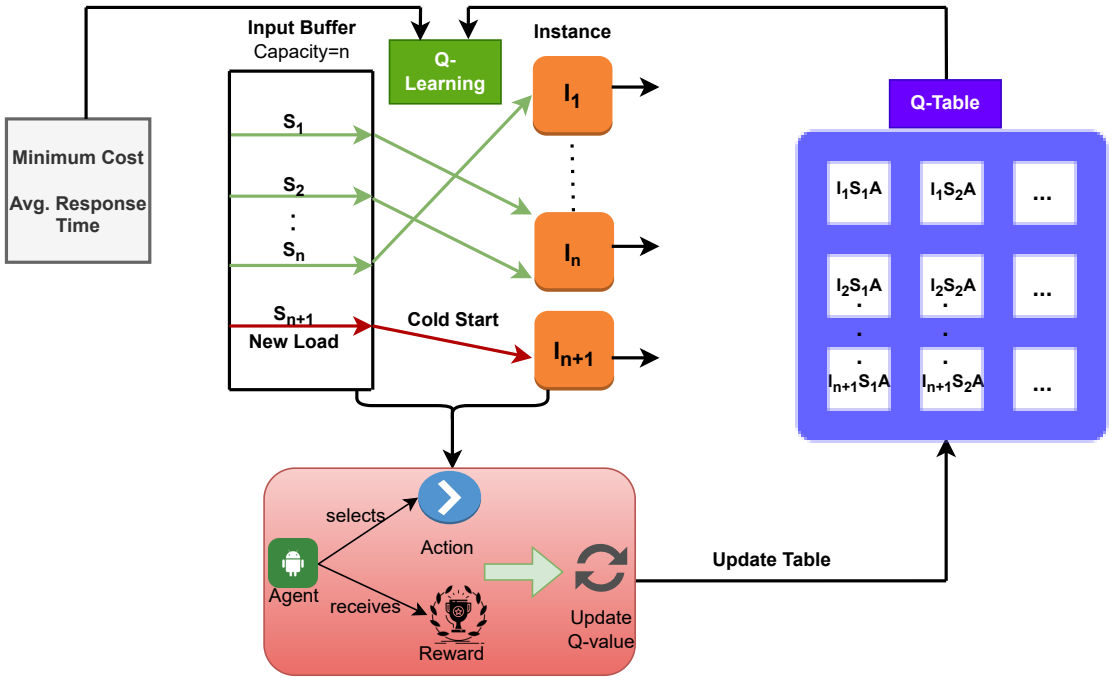

Figure 2: An overview of the proposed autoscalable approach using Q-Learning

down in response to demand fluctuations, optimizing resource utilization.

The approach verifies the availability of instances when a request arrives and searches for an available instance. If instances are available, they have been allocated to fulfill the request. However, if a new request arrives and there are no available instances, this triggers a cold start scenario. To address this, a new function instance has been introduced to the warm pool, effectively scaling up the instances to meet the increased demand. On the other hand, if there has been no request for a specific duration, the approach implements a scale-in mechanism. Instances that have been idle for a predetermined amount of time have been considered expired and terminated, leading to a reduction in the number of active instances. The allocation of instances to requests has been governed by a minimum cost and average response time. This ensures a balanced approach that considers both cost efficiency and timely responsiveness. Recognizing the potential limitation of minimum cost allocation in achieving actual cost savings, the proposed approach has incorporated a Q-learning algorithm. Initially, the Q-learning algorithm will start with identifying the current state (S) of the serverless application including factors like response time, resource utilization, and request rate. Further, the Q-table has been designed to store Q-values for state-action pairs and specify the Q-learning parameters like the learning rate (α), and discount factor (γ). Then, the exploration strategies has been implemented to balance exploration (try a random action) and exploitation (choose the action with the highest Q-value). Next, an action (A) will be chosen based on scaling decisions i.e. adding or removing function instances in serverless and measures reward (R) that could be based on maintaining performance and cost savings. During each episode, the agent selects an action, receives a reward, and updates the Q-value using the Bellman equation as given in Eq. 1:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a} Q(s',a') - Q(s,a) \right) \quad (1)$$

where $Q(s,a)$ is the Q-value for the (state,action) pair, α is the learning rate, $R(s,a)$ is the reward for the current allocation, γ is the discount factor and $\max_a Q(s',a')$ is the maximum Q-value for the next state.
By guiding the allocation based on q-values associated with maximum reward values, the approach aims to enhance its overall performance, achieving cost savings and improved efficiency in resource allocation.

4.2. Proposed Algorithm

In this section, the algorithm has been proposed to address resource scheduling and evaluate performance parameters.
Algorithm 1 initializes the data structures necessary for the subsequent algorithms. It sets up the Q matrix to store (σ, ν) pairs and creates lists of σ and ν with their respective attributes. It adjusts the demands δ_c, δ_r, and δ_b based on a given parameter θ and takes a list of σ with attributes as input and outputs the adjusted demands for each δ_i in each σ_i.

Algorithm 2 assigns resources ν_c to tasks δ_n based on performance parameters. It iterates over each σ_i in the list of σ with

Algorithm 1 Initialization and Load Adjustment

Initialize Q for (σ,ν) pairs.
Create lists of σ and ν with attributes.
for each σ_i in σ **do**
    for each δ_i in σ_i **do**
        Adjust δ_c, δ_r, δ_b demands based on θ.
    end for
end for

attributes, creates a new task δ_n, and then either explores or exploits the available resources ν_c for that task. Finally, it assigns the chosen resources to the task if they meet certain conditions and calculates $R(x)$ for the task.

Algorithm 3 collects data based on performance parameters. It

Algorithm 2 Job Allocation

Input: List of σ with attributes.
Output: Assigned ν_c to δ_n, collected data based on performance parameters.
for each σ_i in σ **do**
    Create δ_n for σ_i.
    if $r_u(0,1) < \epsilon$ **then** ▷ Exploration
        $\nu_a \leftarrow \nu_c$ for δ_n.
        if $\nu_a \neq 0$ **then**
            $\nu_c \xleftarrow{rand} \nu_a$.
        end if
    else ▷ Exploitation
        $\nu_a \leftarrow \nu_c$ for δ_n.
        if $\nu_a \neq 0$ **then**
            $\nu_c \xleftarrow{argmax} \nu_a$.
        end if
    end if
    if $\nu_c \neq 0$ and $\nu_c \subseteq \delta_n$ **then**
        Assign ν_c to δ_n.
        Calculate $R(x)$ for δ_n.
        Call Equation 1.
    end if
end for

takes performance parameters as input and collects data based on them. It iterates over each θ in a set θ_s, initializes σ and ν, executes the Load Adjustment and Job Allocation algorithms, and collects results for each iteration in totalIterations.

The output of the above algorithm is the data collected based on performance parameters that are further evaluated in the next section.

4.3. Evaluation Metrics

In the following subsections, the calculation of different metrics in the proposed approach has been presented and the symbols used in the proposed approach are defined in Table 2.

4.3.1. Probability of Cold Start (P_c)

In Eq. 2 probability of a cold start can be calculated by dividing the number of requests causing a cold start by the total

Algorithm 3 Result Collection

Input: Performance parameters.
Output: Collected data based on performance parameters.
for each θ in θ_s **do**
    Initialize σ and ν.
    Execute Algorithm 1.
    for each iteration in totalIterations **do**
        Execute Algorithm 2.
        Perform Result Collection.
    end for
end for

Table 2: Symbols and their corresponding description

Symbol	Description
P_c	Probability of cold start
R_c	Requests causing cold start
R_n	Total number of requests
$\overline{X}$	Average response time
I_w	Mean number of warm instances
$I_{w,n}$	Number of warm instances
I_n	Total number of instances
I_{idle}	Mean number of idle instances
I_c	Number of instances in cold state
I_r	Number of running instances
$\overline{U}$	Mean utilization
E_c	Energy consumption
T_{exp}	Expiration Threshold

number of requests made during the experiment.

$$P_c = \frac{R_c}{R_n} \tag{2}$$

4.3.2. Average Response Time ($\overline{X}$)

Eq. 3 calculates this metric by finding the average response time for completed jobs (n) across multiple users. This parameter takes a list of users as input and each user has a list of jobs. The response time is accumulated for computed jobs and the average response time is computed.

$$\overline{X} = \frac{\sum_{i=1}^{n} \overline{X}_i}{n} \tag{3}$$

4.3.3. Mean Number of Warm Pool Instances(I_w)

The average number of instances in the warm pool can be calculated as shown in Eq. 4 based on the number of warm instances ($I_{w,n}$) and the total number of instances (I_n)

$$I_w = \frac{I_{w,n}}{I_n} \tag{4}$$

4.3.4. Mean Number of Idle Instances (I_{idle})

This can be measured in Eq. 5 as the ratio of number of instances in the cold state to the total number of instances.

$$I_{idle} = \frac{I_c}{I_n} \tag{5}$$

4.3.5. Mean Number of Running Instances (I_r)

As shown in Eq. 6, a mean number of running instances can be calculated by subtracting the number of idle instances from the total number of instances.

$$I_r = I_n - I_{idle} \tag{6}$$

4.3.6. Utilization ($\overline{U}$)

This is defined as in Eq. 7 the ratio of instances in running state relative to all instances.

$$\overline{U} = \frac{I_r}{I_n} \tag{7}$$

5. Experimental Results

To show the effectiveness, reliability, and adaptability of the proposed approach, the results outlined in this section are applied to AWS Lambda. The approach follows scale-per-request i.e. having no queue using AWS Lambda. The serverless computing offerings are not adaptive to the workload that is being executed on them but by optimizing the expiration threshold (T_{exp}), after which being idle causes the instance to be expired and terminated is the one way by which serverless computing platform is adaptive to the executed workload. Figs. 4 to 9 depict the effect of (T_{exp}) on different performance parameters for different workloads as shown in Table 3. It can be seen, that the increase in T_{exp} would improve the performance. However, as the average response time ($\overline{X}$) is the main parameter of performance by using this approach we try to decrease cost and energy consumption to the maximum.

Table 3: Workload Analysis for Cost and Energy Efficiency in Serverless Computing Environments

Name	Application
L1	A CPU and disk-intensive benchmark [6]
L2	A range of benchmarks with different configuration [53]
L3	A startup test with echo on Apache OpenWhisk [54]
L4	A Fibonacci calculation on AWS Lambda [55]

- **Case 1: Probability of Cold Start** (P_c)

Fig. 3 shows the probability of a cold start over the expiration threshold (T_{exp}). When determining the quality of service, users mostly look at this measure. Reducing the probability of a cold star is critical for many applications as having a larger probability of a cold start could affect the experience of the user. The probability of cold start for workloads L1, L2, L3, and L4 are 3.11 %, 3.00 %, 2.77 %, and 2.66 % respectively

- **Case 2: Average Response Time** ($\overline{X}$)

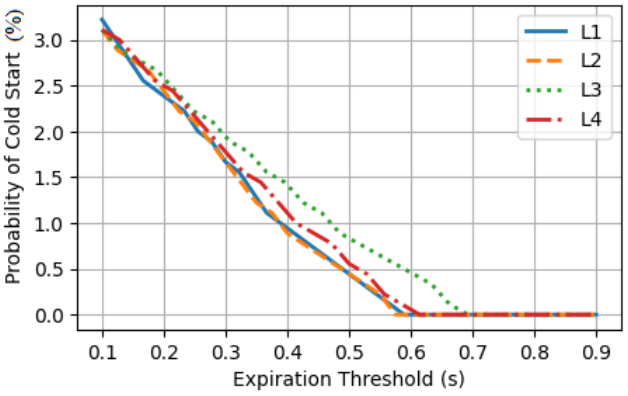

Figure 3: Probability of cold start against the expiration threshold

As can be seen in Fig. 4, the proposed approach obtains the average response time of 30.25 ms for workload L1, 64.24 ms for workload L2, 37.72 ms for workload L3, and 25.42 ms for workload L4. The different workloads have different behavior when changing the expiration threshold (T_{exp}).

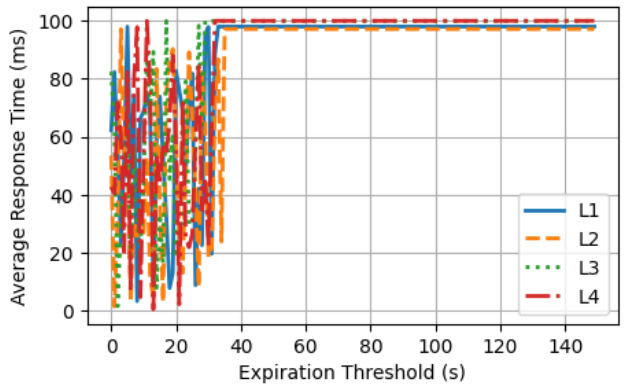

Figure 4: Average response time against the expiration threshold

- **Case 3: The Mean Number of Idle Instances** (I_{idle})

Fig. 5 depicts the variation in the number of idle instances with the expiration threshold. As the expiration threshold increases, a noticeable decrease in the number of idle instances is observed, suggesting a potential optimization in resource utilization. The graph further illustrates that, under varying workloads indicated by the contrasting lines, the Q-learning approach consistently performs well. These findings underscore the importance of considering expiration thresholds in optimizing system performance and resource allocation, with implications for overall efficiency and cost-effectiveness. The mean number of idle instances during different workloads L1, L2, L3, and L4 are 28, 27, 25 and 24 respectively.

- **Case 4: Job Completion Rate** (J_c)

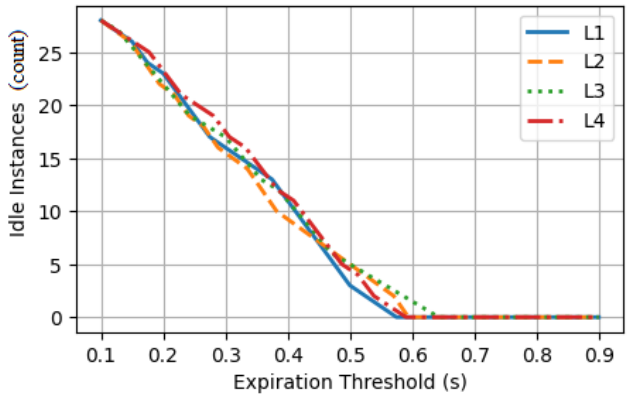

Figure 5: The number of idle instances against the expiration threshold

Fig. 6 shows the correlation between the expiration threshold and the job completion rate. As the expiration threshold increases, there is an observable trend of improvement in job completion rates, suggesting a positive impact on overall system efficiency. The rate of job completion for the proposed approach is 57.78 %, 55.59 %, 55.27 % and 53.28 % for L1, L2, L3, and L4 respectively.

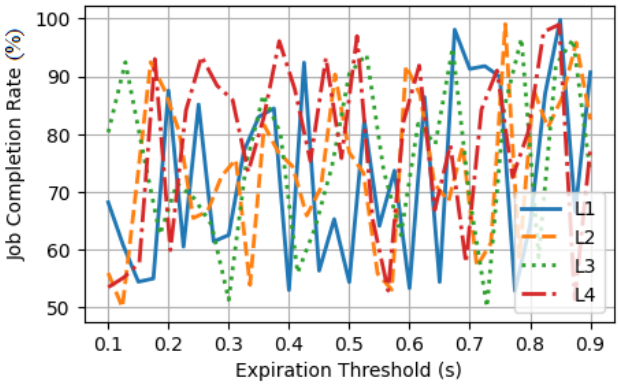

Figure 6: Job completion rate against the expiration threshold

- **Case 5: Energy Consumption**(E_c)

The proposed approach achieves reduced energy consumption because scaling down the resources effectively shuts down idle system components when they are not needed, thereby minimizing energy wastage. As shown in Fig. 7, the expiration threshold increases and there is a noticeable trend of increasing energy consumption. This trend holds consistent across varying workloads, suggesting that optimizing expiration thresholds can contribute to improved sustainability in resource usage. The total energy consumption estimated by different workloads L1, L2, L3, and L4 is 0.0084 mJ, 0.0085 mJ, 0.0092 mJ and 0.0087 mJ respectively.

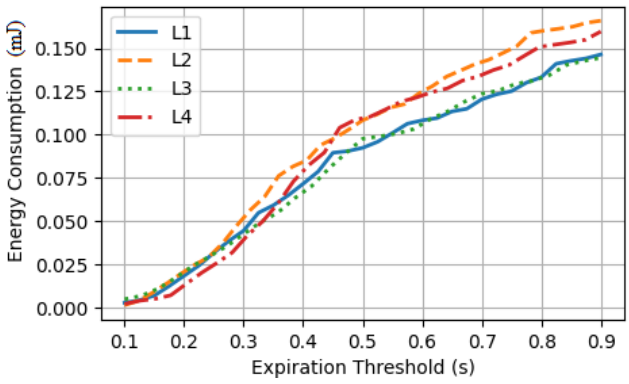

Figure 7: Energy consumption against the expiration threshold

- **Case 6: Utilization** ($\overline{U}$)

The utilization of the warm pool instances for various expiration threshold values is shown in Fig. 8 The average ratio of the number of running instances over all instances in the warm pool is represented by utilization in this context. Reduced utilization leads to additional instances being created and maintained, which increases cost. Serverless platforms would maximize average utilization to reduce cost and the values for L1, L2, L3 and L4 are 3.33 %,3.33 %,2.85 % and 3.03 % respectively.

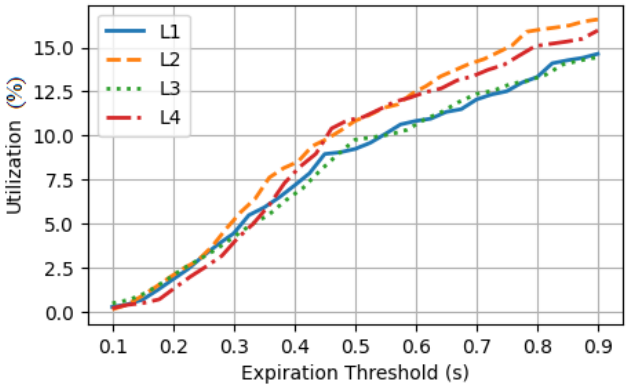

Figure 8: Utilization against the expiration threshold

- **Case 7: Cost** (C)

Fig. 9 presents a normalized estimate cost as perceived by the user. The expiration threshold can be changed to examine variations in the behavior of the workload. For example, increasing the expiation threshold from 0 to 140 causes an increase in user cost. This trend holds consistent across varying workloads, suggesting that optimizing expiration thresholds will lead to a decrease in user cost. This shows the potential savings that can be implemented by the approach presented and evaluated in this paper.

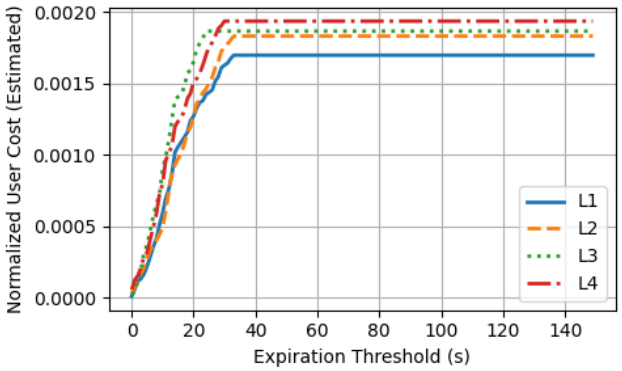

Figure 9: Normalized User Cost against the expiration threshold

5.1. *Comparison of proposed approach with existing approach*

The comparison of the proposed approach with the base approach [10] is done by taking various evaluation parameters such as P_c, $\overline{X}$, I_{idle}, $\overline{U}$, E_c. As can be seen in Fig.10, the proposed approach results better than as compared with the base approach.

Table 4 shows the improvement of different parameters obtained using the proposed approach and base approach. It is clear from Table 4 that the proposed approach improves P_c for different workloads by 38.79%, 26.11%, 35.17% and 53.91% respectively than the base approach. The proposed approach also outperforms the base approach by increasing $\overline{X}$ by 31.73%, 26.29%, 39.23%, and 46.24% for L1, L2, L3, and L4 respectively. The proposed approach shows improvement in I_{idle} by 3.20%, 3.23%, 3.81%, and 3.23% respectively for L1, L2, L3, and L4. The proposed approach improves E_c for workloads L1 by 51.33%, L2 by 46.97%, L3 by 40.95%, and L4 by 47.35% respectively than the base approach

Table 4: Comparative Analysis: Proposed Approach's Improvement Percentage

Metrics	P_c	$\overline{X}$	I_{idle}	E_c
L1	38.79	31.73	3.20	51.33
L2	26.11	26.29	3.23	46.97
L3	35.17	39.23	3.81	40.95
L4	53.91	46.24	3.23	47.35

5.2. *Verification and Validation: Smart Meters for residential buildings*

The proposed approach has been verified and validated to optimize energy consumption by taking smart meter data for residential buildings[8],[9]. To optimize energy consumption in the smart meters dataset for residential buildings, the dataset serves as input to the Q-learning algorithm as shown in Fig.11. Each input to a dataset represents a state, encapsulating relevant information such as energy consumption patterns, weather conditions and appliance usage. The agent utilizes state information to make decisions for optimizing energy consumption. By measuring rewards associated with different actions taken by the agent, the agent learns to select actions that lead to the greatest rewards over time. This approach aims to minimize energy waste, reduce costs for residents, and enhance overall energy efficiency in residential buildings.

As shown in Fig.12, the results illustrate that the proposed approach, utilizing real-time smart meter data, achieves lower energy consumption compared to actual usage by dynamic resource allocation and optimization. The case study involved monitoring energy consumption at different time intervals.The actual energy consumption values were 100 kWh, 120 kWh, 90 kWh, 110 kWh, 130 kWh, and 95 kWh, respectively. After implementing our proposed approach for resource scaling in serverless computing environments, the energy consumption improved to 95 kWh, 98 kWh, 84 kWh, 102 kWh, 115 kWh, and 87 kWh, respectively. Additionally, the implementation has led to improved energy efficiency, cost savings, and an average improvement of approximately 9.54% in energy consumption, showcasing the scalability and applicability of the approach.

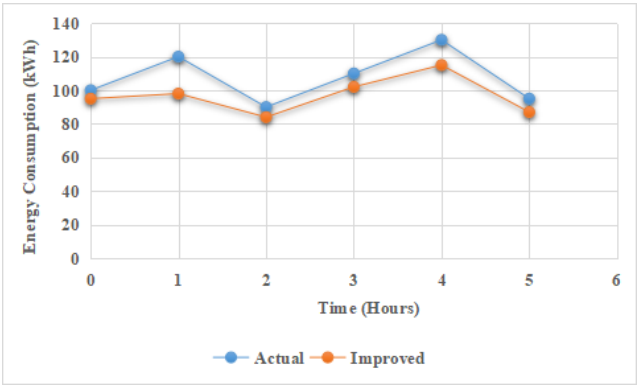

Figure 12: Actual and improved electricity consumption of residential buildings using proposed approach

6. Conclusion and future scope

In this paper, an auto-scalable approach has been proposed and experimentally validated for analyzing performance in terms of scalability, cost, and optimizing energy consumption, with inspiration from Science and Technology for Energy Transition (STET) applications. The proposed approach outperforms the existing approach as it improves the average response time by 35.62%, mean number of idle instances by 3.37% and reduces the probability of cold start and energy consumption by 38.5 % and 46.15 % respectively.The case study on optimizing energy consumption using the proposed approach further illustrates its practical applicability and effectiveness in real-world scenarios. The proposed approach uses a scale-per-request autoscaling pattern as its importance in serverless computing platforms. The presented approach can be used to improve the quality of service by improving their management policy and making their operations predictive. The proposed approach enables developers to handle enormous changes in their workload by providing scalable computing.

The proposed approach could be enhanced using the following future work:

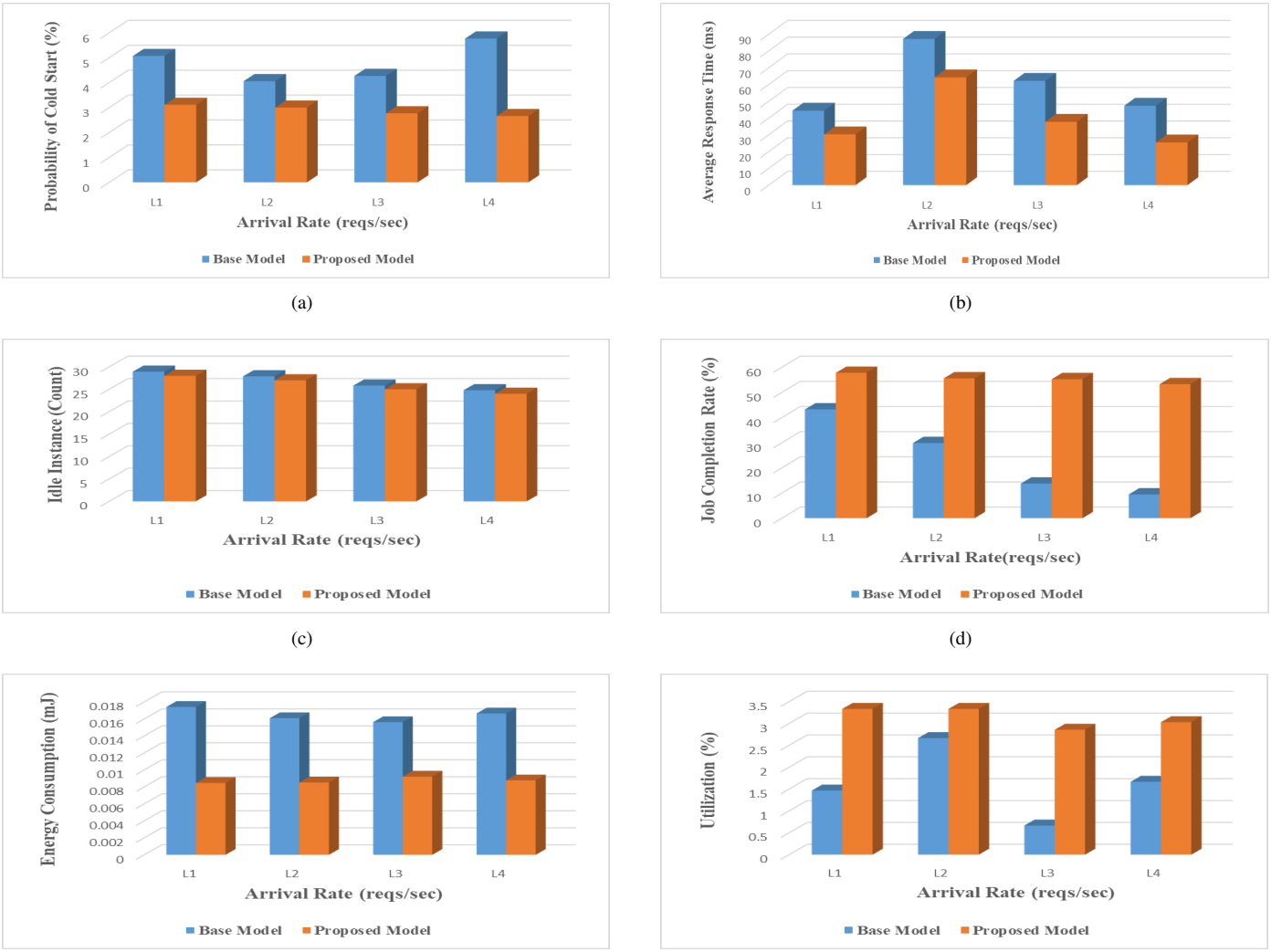

Figure 10: Comparative Analysis of Parameters Between Proposed and Base Approaches Across Various Workloads (a) Probability of Cold Start (b) Average Response Time (c) Idle Instances (d) Job Completion Rate (e) Energy Consumption (f) Utilization

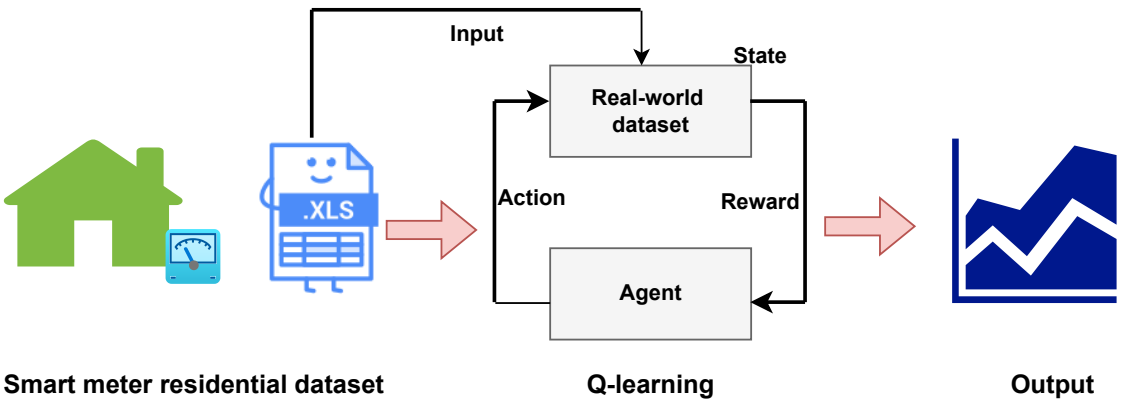

Figure 11: Smart meters for residential buildings using Q-learning

- In the future, the developed approach can also be improved using other reinforcement learning algorithms such as Deep Q-Networks (DQNs) to handle more complex state spaces and improve scalability decisions.

- To create effective auto-scaling solutions, additional factors like real-time monitoring, predictive approaching, and feedback control loops with Q-learning need to be considered in the future.

- Also, the use of machine learning or deep learning algorithms in the proposed approach would be able to improve more in terms of performance and energy consumption in the future.

References

[1] A. Zafeiropoulos, E. Fotopoulou, N. Filinis, S. Papavassiliou, Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms, Simulation Modelling Practice and Theory 116 (2022) 102461.

[2] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, R. Bianchini, Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider, in: 2020 USENIX annual technical conference (USENIX ATC 20), 2020, pp. 205–218.

[3] S. Agarwal, M. A. Rodriguez, R. Buyya, A reinforcement learning approach to reduce serverless function cold start frequency, in: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, 2021, pp. 797–803.

[4] L. Schuler, S. Jamil, N. Kühl, Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments, in: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, 2021, pp. 804–811.

[5] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, S. Eismann, A spec rg cloud group's vision on the performance challenges of faas cloud architectures, in: Companion of the 2018 acm/spec international conference on performance engineering, 2018, pp. 21–24.

[6] L. Wang, M. Li, Y. Zhang, T. Ristenpart, M. Swift, Peeking behind the curtains of serverless platforms, in: 2018 USENIX Annual Technical Conference (USENIX ATC 18), 2018, pp. 133–146.

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

[8] S. Kaur, A. Bala, A. Parashar, A multi-step electricity prediction model for residential buildings based on ensemble empirical mode decomposition technique, Science and Technology for Energy Transition 79 (2024) 7.

[9] Z. Aljicevic, S. Kasapovic, J. Hivziefendic, J. Kevric, S. Mujkic, Resource allocation model for cloud-fog-based smart grid, Science and Technology for Energy Transition 78 (2023) 28.

[10] N. Mahmoudi, H. Khazaei, Performance modeling of serverless computing platforms, IEEE Transactions on Cloud Computing 10 (4) (2020) 2834–2847.

[11] N. Mahmoudi, H. Khazaei, Temporal performance modelling of serverless computing platforms, in: Proceedings of the 2020 Sixth International Workshop on Serverless Computing, 2020, pp. 1–6.

[12] N. Mahmoudi, H. Khazaei, Simfaas: A performance simulator for serverless computing platforms, arXiv preprint arXiv:2102.08904 (2021).

[13] S. N. A. Jawaddi, A. Ismail, Autoscaling in serverless computing: Taxonomy and openchallenges (2023).

[14] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, A. Gandhi, Ensure: Efficient scheduling and autonomous resource management in serverless environments, in: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), IEEE, 2020, pp. 1–10.

[15] V. Shankar, K. Krauth, K. Vodrahalli, Q. Pu, B. Recht, I. Stoica, J. Ragan-Kelley, E. Jonas, S. Venkataraman, Serverless linear algebra, in: Proceedings of the 11th ACM Symposium on Cloud Computing, 2020, pp. 281–295.

[16] N. Mahmoudi, H. Khazaei, Performance modeling of metric-based serverless computing platforms, IEEE Transactions on Cloud Computing (2022).

[17] Y. Zhao, A. Uta, Tiny autoscalers for tiny workloads: Dynamic cpu allocation for serverless functions, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, 2022, pp. 170–179.

[18] J. Wen, Z. Chen, Y. Liu, Y. Lou, Y. Ma, G. Huang, X. Jin, X. Liu, An empirical study on challenges of application development in serverless computing, in: Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, 2021, pp. 416–428.

[19] A. Pérez, S. Risco, D. M. Naranjo, M. Caballer, G. Moltó, On-premises serverless computing for event-driven data processing applications, in: 2019 IEEE 12th International conference on cloud computing (CLOUD), IEEE, 2019, pp. 414–421.

[20] J. Kim, K. Lee, I/o resource isolation of public cloud serverless function runtimes for data-intensive applications, Cluster Computing 23 (2020) 2249–2259.

[21] J. Enes, R. R. Expósito, J. Touriño, Real-time resource scaling platform for big data workloads on serverless environments, Future Generation Computer Systems 105 (2020) 361–379.

[22] D. Jackson, G. Clynch, An investigation of the impact of language runtime on the performance and cost of serverless functions, in: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), IEEE, 2018, pp. 154–160.

[23] H. Shafiei, A. Khonsari, P. Mousavi, Serverless computing: a survey of opportunities, challenges, and applications, ACM Computing Surveys 54 (11s) (2022) 1–32.

[24] P. Singh, A. Kaur, S. S. Gill, Machine learning for cloud, fog, edge and serverless computing environments: comparisons, performance evaluation benchmark and future directions, International Journal of Grid and Utility Computing 13 (4) (2022) 447–457.

[25] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, R. Buyya, Ifaasbus: A security-and privacy-based lightweight framework for serverless computing using iot and machine learning, IEEE Transactions on Industrial Informatics 18 (5) (2021) 3522–3529.

[26] A. Grafberger, M. Chadha, A. Jindal, J. Gu, M. Gerndt, Fedless: Secure and scalable federated learning using serverless computing, in: 2021 IEEE International Conference on Big Data (Big Data), IEEE, 2021, pp. 164–173.

[27] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, M. Guo, The serverless computing survey: A technical primer for design architecture, ACM Computing Surveys (CSUR) 54 (10s) (2022) 1–34.

[28] S. Bebortta, S. K. Das, M. Kandpal, R. K. Barik, H. Dubey, Geospatial serverless computing: Architectures, tools and future directions, ISPRS International Journal of Geo-Information 9 (5) (2020) 311.

[29] S. S. Gill, Quantum and blockchain based serverless edge computing: A vision, model, new trends and future directions, Internet Technology Letters (2021) e275.

[30] N. Mateus-Coelho, M. Cruz-Cunha, Serverless service architectures and security minimals, in: 2022 10th International Symposium on Digital Forensics and Security (ISDFS), IEEE, 2022, pp. 1–6.

[31] E. Marin, D. Perino, R. Di Pietro, Serverless computing: a security perspective, Journal of Cloud Computing 11 (1) (2022) 1–12.

[32] V. Yussupov, U. Breitenbücher, F. Leymann, M. Wurster, A systematic mapping study on engineering function-as-a-service platforms and tools, in: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, 2019, pp. 229–240.

[33] E. Van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uță, A. Iosup, Serverless is more: From paas to present cloud computing, IEEE Internet Computing 22 (5) (2018) 8–17.

[34] R. Cordingly, W. Shu, W. J. Lloyd, Predicting performance and cost of serverless computing functions with saaf, in: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), IEEE, 2020, pp. 640–649.

[35] D. Bardsley, L. Ryan, J. Howard, Serverless performance and optimization strategies, in: 2018 IEEE International Conference on Smart Cloud (SmartCloud), IEEE, 2018, pp. 19–26.

[36] R. A. P. Rajan, Serverless architecture-a revolution in cloud computing, in: 2018 Tenth International Conference on Advanced Computing (ICoAC), IEEE, 2018, pp. 88–93.

[37] J. Grogan, C. Mulready, J. McDermott, M. Urbanavicius, M. Yilmaz, Y. Abgaz, A. McCarren, S. T. MacMahon, V. Garousi, P. Elger, et al., A multivocal literature review of function-as-a-service (faas) infrastructures and implications for software developers, in: Systems, Software and Services Process Improvement: 27th European Conference, EuroSPI 2020, Düsseldorf, Germany, September 9–11, 2020, Proceedings 27, Springer, 2020, pp. 58–75.

[38] P. Vahidinia, B. Farahani, F. S. Aliee, Mitigating cold start problem in serverless computing: a reinforcement learning approach, IEEE Internet of Things Journal 10 (5) (2022) 3917–3927.

[39] X. Liu, J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, X. Jin, Faaslight: general application-level cold-start latency optimization for function-as-a-service in serverless computing, ACM Transactions on Software Engineering and Methodology (2023).

[40] A. Fuerst, P. Sharma, Faascache: keeping serverless computing alive with greedy-dual caching, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 386–400.

[41] A. Mampage, S. Karunasekera, R. Buyya, Deadline-aware dynamic resource management in serverless computing environments, in: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, 2021, pp. 483–492.

[42] G. Kaur, A. Bala, I. Chana, An intelligent regressive ensemble approach for predicting resource usage in cloud computing, Journal of Parallel and Distributed Computing 123 (2019) 1–12.

[43] S. Datta, S. K. Addya, S. K. Ghosh, Esma: Towards elevating system happiness in a decentralized serverless edge computing framework, Journal of Parallel and Distributed Computing 183 (2024) 104762.

[44] D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, G. Moltó, Accelerated serverless computing based on gpu virtualization, Journal of Parallel and Distributed Computing 139 (2020) 32–42.

[45] P. G. Sarroca, M. Sánchez-Artigas, Mlless: Achieving cost efficiency in serverless machine learning training, Journal of Parallel and Distributed Computing 183 (2024) 104764.

[46] P. Zuk, K. Rzadca, Reducing response latency of composite functions-as-a-service through scheduling, Journal of Parallel and Distributed Computing 167 (2022) 18–30.

[47] K. Solaiman, M. A. Adnan, Wlec: A not so cold architecture to mitigate cold start problem in serverless computing, in: 2020 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2020, pp. 144–153.

[48] K. Suo, Y. Shi, X. Xu, D. Cheng, W. Chen, Tackling cold start in serverless computing with container runtime reusing, in: Proceedings of the Workshop on Network Application Integration/CoDesign, 2020, pp. 54–55.

[49] D. Bermbach, A.-S. Karakaya, S. Buchholz, Using application knowledge to reduce cold starts in faas services, in: Proceedings of the 35th annual ACM symposium on applied computing, 2020, pp. 134–143.

[50] Z. Jia, E. Witchel, Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 152–166.

[51] V. Mittal, S. Qi, R. Bhattacharya, X. Lyu, J. Li, S. G. Kulkarni, D. Li, J. Hwang, K. Ramakrishnan, T. Wood, Mu: an efficient, fair and responsive serverless framework for resource-constrained edge clouds, in: Proceedings of the ACM Symposium on Cloud Computing, 2021, pp. 168–181.

[52] H. Lee, K. Satyam, G. Fox, Evaluation of production serverless computing environments, in: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), IEEE, 2018, pp. 442–450.

[53] R. Vojta, Aws journey: Api gateway & lambda & vpc performance, Zrzka's adventures (2016).

[54] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, V. Hilt, {SAND}: Towards {High-Performance} serverless computing, in: 2018 Usenix Annual Technical Conference (USENIX ATC 18), 2018, pp. 923–935.

[55] J. Manner, M. Endreß, T. Heckel, G. Wirtz, Cold start influencing factors in function as a service, in: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), IEEE, 2018, pp. 181–188.