SOFNET: Scheduling and Load Balancing Framework for Security Aware Fog Networks

Rahul Verma
2021csm1004@iitrpr.ac.in
IIT Ropar
Punjab, India
Nitin Auluck
nitin@iitrpr.ac.in
IIT Ropar
Punjab, India

Abstract

IoT devices create a huge amount of data which needs to be processed, store and compute. In order to support the IoT, a powerful distributed system is required i.e., cloud computing. Some of the real-time applications need a quick response for which cloud computing is not sufficient. Thus, Fog computing comes into play. The middle layer between cloud and IoT devices. Fog computing is a new technology that executes jobs at the network edge to provide latency-sensitive services. A uniform distribution of jobs is critical for ensuring a high level of service SOFNET (Scheduling and Load Balancing Framework for Security Aware Fog Networks), our proposed approach, distributes job load across the fog network and schedules real-time jobs based on deadline, communication delay, size, resource utilisation, and security constraints. Jobs submitted by users are categorised as classified, restricted, or public. Cloud and fog nodes are classified as either native or public. Based on the privacy requirements, SOFNET assigns jobs to fog or cloud. Furthermore, we propose a security-aware migration procedure that reallocates the job after the initial job execution fails due to overloading. We employed a customised workload and compared the experimental findings with fog-only and cloud-only systems. According to experimental findings, our approach outperforms other comparable algorithms in terms of success rate, system cost, and resource utilisation.

1 Introduction

Cloud computing is an on-demand service that supports a wide range of user applications in healthcare, transportation, smart homes, and surveillance, among other areas. Because data centres are geographically located at a significant distance from the data generation source, running applications on cloud data centres (cdc) results in high communication and propagation delays. Fog computing is a network-edge computing paradigm that extends cloud-based storage, computation, and communication services. It allows users to use the computation and storage resources available at nearby fog nodes, as well as cloud services, on a pay-as-you-go basis. Low latency, mobility, geographical distribution, and location awareness are just a few of the features that make fog an ideal platform for a variety of time-critical applications.

In recent years, numerous researchers have investigated the scope of fog computing with various applications. A security camera at an agricultural farm that detects wild animals/intruders and immediately alerts authorities is an example of a time-critical job. These

types of jobs require processing in fractions of seconds, which cloud data centres cannot guarantee.

Although fog computing provides applications with a distributed processing infrastructure, its storage, communication, and computational capabilities are limited in comparison to cloud data centres. As a result, it is critical to make effective use of the available limited resources. Load balancing is an approximate uniform distribution of workload to cloud and fog nodes that is one such technique for resource utilisation.

Fog nodes within the same network may have different computation, storage, and communication latency depending on the hardware and software specifications. Because of these heterogeneous characteristics, it is extremely difficult to use simpler load-balancing techniques in fog computing. As a result, there is a need for a fog technique that not only distributes the load but also considers factors that directly affect performance and efficiency. To the best of our knowledge, no work has been done that investigates the load balancing of security-aware real-time jobs on fog networks. As a result, for a three-tier architecture, we propose the load-aware scheduling algorithm.

Each job category is assigned to a suitable resource based on its security tag, deadline, and space constraint. When the utilisation of a resource exceeds a certain threshold, jobs are migrated from one resource to another using a proposed migration algorithm. Figure 1 depicts a load-balancing layout for fog networks. This model's goal is to distribute network load based on job deadline, communication delay, space, job security, and resource utilisation.

The rest of the paper is structured as follows. Section 2 describes the existing work in fog-based network scheduling and job placement. Section 3 develops the proposed architecture's overview and design details. This section also describes the mathematical illustration of the optimization problem considered in this work. Section 4 discusses the proposed methodology and algorithm description. Section 5 compares our proposed approach to existing approaches and discusses the simulation results. Finally, in Section 6, we discuss the work's conclusion and future scope.

2 Related Work

Numerous studies that use fog computing features to improve the performance of IoT ecosystems have been proposed, which we will now discuss. Choudhari et al. [2] proposed a prioritised task scheduling approach for fog computing to improve the performance of user applications. The model includes a priority queue and a manager who assigns jobs based on their priority levels. However, because it only focuses on static modelling, the work necessitates a more dynamic and robust scheduling mechanism. [3] presents a survey of task scheduling approaches for fog computing.

Bitam et al. [1] presented an optimization technique for job scheduling based on bee life algorithms. This technique was created with the goal of determining the best point of trade-off between CPU execution time and memory requirements for fog computing services. Similarly, approaches to resource allocation and task scheduling for fog computing were proposed in works [5, 9, 10]. Mogi et al. [4] present an edge computing-based load balancing method for an IoT sensor system. A four-phase load balancing algorithm is used by the edge nodes to transfer their load to neighbouring edge nodes. However, some critical parameters, such as communication delay and job deadline, are not taken into account by the algorithm. Li et al. [11] propose a load-balancing strategy in which task allocation in the edge layer is

Fig. 1. Load balancing in fog networks

handled by intermediary nodes.

Load balancing has become an essential component of fog-based solutions as the number of end-users and IoT devices on the network has grown. The following section describes some of the existing load-balancing work in fog networks. Singh and Auluck proposed three load-balancing-aware scheduling algorithms [7]. This work focuses on job deadlines and communication delays but does not take into account fog node authentication in a real-time scenario. Puthal et al. [6] propose a secure load-balancing algorithm that identifies and assigns tasks to the network's less-loaded edge data centres. This work, however, ignores critical factors such as job deadlines, real-time workload, and energy consumption. Wan et al. [8] present a smart factory energy-aware load-balancing technique. The work presented by us focuses on job deadlines, space constraints, resource utilization, real-time jobs, communication delay and job security.

3 SYSTEM MODEL

This section describes how edge, fog, and cloud layers are integrated into a three-tier architecture. Figure 2 depicts a graphical representation of the architecture. The framework is proposed with the goal of scheduling real-time IoT jobs on fog and cloud resources while maintaining uniform load distribution.

3.1 Architecture Overview

An end-user layer is the lowest layer of the proposed three-tier architecture. It primarily consists of physical hardware devices used in various IoT applications, such as sensors, actuators, and communication devices. This layer's physical devices collect real-time IoT data as required by user applications.

TABLE I
SYMBOL TABLE FOR SYSTEM MODEL

Symbol	Description
C, F	Set of cdcs, set of fdcs
EU, J	Set of end-users, set of IoT jobs
g, h, e	No. of cdcs, no. of fdcs, no. of end-users
N, N'	Total no. of jobs, no. of jobs completed before deadline
c_n, c_p	Native, public cdc
f_n, f_p	Native, public fdc
t_c, t_r, t_p	Classified, restricted, public security tag
j, r	A job, a resource (fdc or cdc)
t_{st}, t_{exec}, t_{cmp}	Start, execution, completion time
dl_{trans}, dl_{prop}, dl_{proc}	Transmission, propagation, processing delay
dl_{queu}, dl_{lat}, dl_{comm}	Queuing, network, communication delay
len, bw	Data packet length, network bandwidth
dis, vl	Distance, velocity
njq, awt	No. of jobs in queue, average waiting time per job
j_c, j_r, j_p	Classified, restricted, public job
j_c', j_r', j_p'	Classified, restricted, public jobs finished before deadline
$d_{j,r}$	Deadline for executing j on r
c_{set}, c_{exec}, c_{comm}	Setup, execution, communication costs
SR, SC, RU	Success ratio, system cost, resource utilization
τ, cap	Time unit, total execution capacity

Fog nodes serve as an interface between end-users and the cloud layer, forming the middle layer of a three-tier architecture. These fog nodes are geographically located in proximity to the end users. A fog node is sometimes referred to as a fog data centre (fdc).

The cloud layer is used for computation-intensive jobs because it provides virtually unlimited data storage and processing. When all of these cloud resources are combined, they are referred to as data centres. Data centres offer pay-as-you-go on-demand services to users.

3.2 Design Details

The given components of the architecture are explained in detail:

- Cloud data centres (cdcs) offer pay-as-you-go on-demand services to users. In our proposed architecture, we consider g number of cdcs, which are represented as C = $\{c_1, c_2, ..., c_g\}$. These cloud data centres have enough storage and computational capacity to handle any IoT task. They are divided into two categories based on their geographical location: native cdc (c_n) and public cdc (c_p). A native cdc is located within an end-local user's network, whereas a public cdc is located outside of the end-local user's network.

- The fog layer is made up of multiple fog data centres (fdcs) that are geographically distributed throughout the network. In our architecture, h fdcs are represented arithmetically as F = $\{f_1, f_2, ..., f_h\}$. Table I describes all of the symbols used in the system model's design. Every end-user is assigned a fog node, which is known as its native fdc (f_n). Except for the end user's native fdc, all other fog nodes are considered public fdc (f_p).

TABLE II
INITIAL JOB-RESOURCES ALLOCATION TABLE

tags	f_n	f_p	c_n	c_p
t_c	✓		✓	
t_r		✓	✓	
t_p		✓		✓

- In the proposed model, we consider e end-users at the end-user layer. They are denoted by EU = $\{u_1, u_2, u_3, ..., u_e\}$. All end-users create jobs that are executed cooperatively on either fdc or cdc. J = $\{j_1, j_2, ..., j_N\}$ represents the jobs to be executed formally. Based on its requirements, each job is assigned a security tag. Our framework employs three types of tags: classified tag (t_c), restricted tag (t_r), and public tag (t_p).

3.3 Problem Formulation

We now present a quantitative optimization model for our proposed architecture that schedules IoT jobs in a load balanced manner in this subsection. The completion time, t_{cmp}, is the time it takes to complete a job j on a resource r.

$$t_{\text{cmp}} = t_{\text{st}} + t_{\text{exec}} + dl_{\text{lat}} \tag{1}$$

Here, t_{st} represents the time when j begins executing on r, t_{exec} represents the time it takes to execute j on r, and dl_{lat} represents the network latency of executing j on r. In this case, network latency dl_{lat} is a collection of various network delays that occur when j is executed on r. This is written as

$$dl_{\text{lat}} = dl_{\text{trans}} + dl_{\text{prop}} + dl_{\text{proc}} + dl_{\text{queu}} \tag{2}$$

where dl_{trans} is the time required to transmit data from the user to the transmission medium, dl_{prop} is the time required for data propagation through the transmission medium, dl_{proc} is the time required by the processor to process the users' data, and dl_{queu} is the delay caused by data packets waiting in a queue. Because high-performance computational hardware is now widely available. As a result, in our work, we assume that $dl_{proc} \approx 0$.Thus, network latency can be calculated as follows:

$$dl_{\text{lat}} = dl_{\text{trans}} + dl_{\text{prop}} + dl_{\text{queu}} \tag{3}$$

$$dl_{\text{lat}} = \frac{len}{bw} + \frac{dis}{vl} + (njq \times awt) \tag{4}$$

such that len is the length of the data packet, bw is the bandwidth of the adopted IoT network, dis is the distance travelled by a data packet, vl is the velocity of the communication channel, njq is the total number of jobs in the queue, and awt is the average waiting time for one job in the queue.The communication delay in the network is represented by dl_{trans}, dl_{prop}, and dl_{queu} in this model. In this work, this delay combination is frequently used interchangeably with dl_{comm}. It is necessary to complete the execution of a job before the deadline expires. As a result, in order to execute j on r, the following condition must be met:

$$t_{\text{st}} + t_{\text{exec}} + dl_{\text{comm}} \leq d_{\text{j,r}} \tag{5}$$

TABLE III
SYMBOL TABLE FOR PROPOSED METHODOLOGY

Symbol	Description
UZI, SF, WB	Utilization, scope factor, waiting bit
d_{min}, d_{max}	Minimum, maximum deadline among all jobs
D1, S1	Deadline and space condition for *fdcs*
D2, S2	Deadline and space condition for *cdcs*
$t_{wt}(r)$	waiting time of j on r before execution

such that $d_{j,r}$ is the deadline of executing a j on r.

As previously stated, the jobs in the proposed architecture are classified, restricted, and public. Let j_c represent a classified job. Similarly, j_r and j_p can provide a restricted and public job, respectively. N represents the total number of jobs to be completed in such a way that:

$$N = \sum j_c + \sum j_r + \sum j_p \tag{6}$$

A classified, restricted, or public job that is completed before the deadline is denoted by $j_c{}'$, $j_r{}'$, and $j_p{}'$, respectively. N represents the total number of jobs that complete their execution before their specified deadlines, such that:

$$N' = \sum j_c{}' + \sum j_r{}' + \sum j_p{}' \tag{7}$$

We use three metrics to assess job execution performance: Success Ratio (SR), System Cost (SC), and Resource Utilization (RU). SR is the ratio of the number of jobs completed before their deadline to the total number of jobs available for completion.

$$SR = \frac{\sum j_c{}' + \sum j_r{}' + \sum j_p{}'}{\sum j_c + \sum j_r + \sum j_p} = \frac{N'}{N} \tag{8}$$

The software and hardware interactions within the network are another critical parameter for measuring the performance of an architecture. The SC determines this factor. It includes the cost of setup, total execution, and communication latency. SC is defined mathematically as

$$SC = c_{set} + c_{exec} + c_{comm} \tag{9}$$

The initial set-up cost is denoted by c_{set}, and the aggregated execution cost of all jobs executed on resource r is denoted by c_{exec}. Finally, c_{comm} denotes the total cost of network communication latency. Because SC is directly proportional to a resource's running time, it must be minimised. The ratio of total execution time to total network capacity is defined as RU. RU must be maximised in order to achieve optimal resource utilisation.

$$RU = \frac{\sum t_{exec}}{cap} \tag{10}$$

The total execution capacity of the available resources is given by cap. As a result, the optimization problem for maximising the performance parameter PR is as follows: maximise SR, minimise SC, and maximise RU.

4 Proposed Methodology for SOFNET

Before going to main algorithm, we require to understand few terms which will be useful in our algorithm.

Utilization (UZI): The utilisation of a resource is the ratio of current usage to total capacity of that resource. The combined UZI of all network resources will result in the proposed architecture's RU.

Scope Factor (SF): A resource's Scope Factor (SF) is the UZI value above which a specific category of jobs cannot be executed on it.

Z-score: A normalised score ranging from 0 to 1 that is calculated for each arriving job. Because the size of a job is directly proportional to its deadline, the z-score of a job indicates the size of the selected job in comparison to all the jobs available for execution. Jobs with z-scores less than 0.5 are considered small, whereas jobs with z-scores greater than 0.5 are considered larger.

$$z-score = \frac{d_{\mathrm{j,r}} - d_{\min}}{d_{\max} - d_{\min}} \tag{11}$$

Here $d_{j_i,r}$ = deadline of executing job j on resource r, d_{min} = deadline of executing job j_y on a resource r' such that y^{th} job when executed on r' has the minimum deadline amongst all the jobs in the network, and d_{max} = deadline of executing job j_z on resource r'' such that z^{th} job when executed on r'' has the maximum deadline amongst all the jobs in the network.

Deadline Constraints: If the total deadline of all the jobs (current job j + all the jobs in queue before j) executing on r is less than the deadline of j, the deadline constraint for executing a j on r is satisfied. D1 and D2 specify the deadline constraints for fdc and cdc, respectively.

$$D1 : (t_{\mathrm{st}} + t_{\mathrm{exec}} + dl_{\mathrm{lat}}) \leq d_{\mathrm{j,fdc}} \tag{12}$$

$$D2 : (t_{\mathrm{st}} + t_{\mathrm{exex}} + dl_{\mathrm{lat}}) \leq d_{\mathrm{j,cdc}} \tag{13}$$

Space constraint: A arriving job j's space constraint for a resource r is met when j's capacity requirement is less than r's available capacity. S1 and S2 represent the space constraints for fdc and cdc, respectively.

$$S1 : t_{\mathrm{exec}} \leq (cap_{\mathrm{fdc}} - \sum_{i=1}^{n} t_{\mathrm{exec_i}}) \tag{14}$$

$$S2 : t_{\mathrm{exec}} \leq (cap_{\mathrm{cdc}} - \sum_{i=1}^{n} t_{\mathrm{exec_i}}) \tag{15}$$

Waiting Bit (WB): A WB is a binary value that indicates whether or not a job must be migrated. Before migrating a job from one resource to another, it is critical to compare the job completion time on both resources (initially allocated, resource planned for migration). If a job j is initially assigned to resource r1 and is scheduled for migration to resource r2:

• WB is set to 1 when the completion time of j on r1 is less than the completion time of j on r2. In this case, the job remains on r1 for execution and is not migrated to r2.

• When j's completion time on r2 is faster than j's completion time on r1, the value of WB is set to be 0.

For WB = 1:

$$[t_{\mathrm{wt}}(r1) + t_{\mathrm{exec}}(r1)] \leq [dl_{\mathrm{lat}}(r2) + t_{\mathrm{wt}}(r2) + t_{\mathrm{exec}}(r2)] \tag{16}$$

For WB = 0:

$$[t_{\mathrm{wt}}(r1) + t_{\mathrm{exec}}(r1)] > [dl_{\mathrm{lat}}(r2) + t_{\mathrm{wt}}(r2) + t_{\mathrm{exec}}(r2)] \tag{17}$$

such that $t_{wt}(r)$ is the waiting time of job j on a resource r before its execution starts.

4.1 Algorithm

The proposed algorithm is intended to distribute workload across multiple resources based on the job's deadline, communication delay, size, resource utilisation, and security requirements.

Algorithm 1 SOFNET: Load Balancing & Scheduling

Input: IoT data in job queue sorted in increasing order of their deadlines.

Output: Data scheduled on fog or cloud node.

```
1: procedure ALGORITHM()
2:     Compute quantities (mention here specifically).
3:     Specify SF threshold values for fog and cloud nodes.
4:     for selected job j having tag = t_c do
5:         CLASSIFIED()
6:     end for
7:     for selected job j having tag = t_r do
8:         RESTRICTED()
9:     end for
10:    for selected job j having tag = t_p do
11:        PUBLIC()
12:    end for
13: end procedure
```

Procedure 1 Initial allocation of classified jobs

```
1: procedure CLASSIFIED()
2:     for selected job j having tag = t_c do
3:         if (D1 & S1 holds on f_n) && (z-score < 0.5) then
4:             schedule j on f_n
5:         else if (D2 & S2 holds on c_n) && (z-score > 0.5)
   then
6:             schedule j on c_n
7:         else if (D1 or S1 does not holds on f_n) then
8:             MIGRATION()
9:         else
10:            try scheduling j later
11:        end if
12:    end for
13: end procedure
```

4.2 Load Migration

When the demand for a particular resource grows significantly, it becomes difficult to meet the deadlines and space constraints on upcoming jobs. This occurs when an upcoming job cannot be executed on a resource because there are other jobs waiting for execution. To address this issue, Procedure 4 proposes a method for moving jobs from one resource to another.

Procedure 2 Initial allocation of restricted jobs

```
1: procedure RESTRICTED()
2:     for selected job j having tag = t_r do
3:         if (D1 & S1 holds on f_p) && (z-score < 0.5) then
4:             schedule j on f_p
5:         else if (D2 & S2 holds on c_n) && (z-score > 0.5)
   && (UZI < SF) then
6:             schedule j on c_n
7:         else if (D2 & S2 holds on c_n) && (z-score > 0.5)
   && (UZI > SF) then
8:             try scheduling j later
9:         else if (D1 or S1 does not holds on f_p) then
10:            MIGRATION()
11:        else
12:            try scheduling j later
13:        end if
14:    end for
15: end procedure
```

Procedure 3 Initial allocation of public jobs

```
1: procedure PUBLIC()
2:     for selected job j having tag = t_p do
3:         if (D1 & S1 holds on f_p) && (z-score < 0.5) &&
   (UZI < SF) then
4:             schedule j on f_p
5:         else if (D1 & S1 holds on f_p) && (z-score < 0.5)
   && (UZI > SF) then
6:             schedule j on c_p
7:         else if (D2 & S2 holds on c_n) && (z-score > 0.5)
   then
8:             schedule j on c_p
9:         else if (D1 or S1 does not holds on f_p) then
10:            MIGRATION()
11:        end if
12:    end for
13: end procedure
```

Procedure 4 Job Migration

```
1: procedure MIGRATION()
2:     if WB = 1 then
3:         do not migrate
4:     end if
5:     for job j has tag = t_c && WB = 0 do
6:         migrate j from f_n to c_n
7:     end for
8:     for job j has tag = t_r do
9:         if f_n has UZI < SF && WB = 0 then
10:            migrate j from f_p to f_n
11:        else if WB = 0 then
12:            migrate j from f_p to c_n
13:        end if
14:    end for
15:    for job j has tag = t_p do
16:        if f_n has UZI < SF && WB = 0 then
17:            migrate j from f_p to f_n
18:        else if c_n has UZI < SF && WB = 0 then
19:            migrate j from f_p to c_n
20:        else if WB = 0 then
21:            migrate j from f_p to c_p
22:        end if
23:    end for
24: end procedure
```

5 Simulation & Results

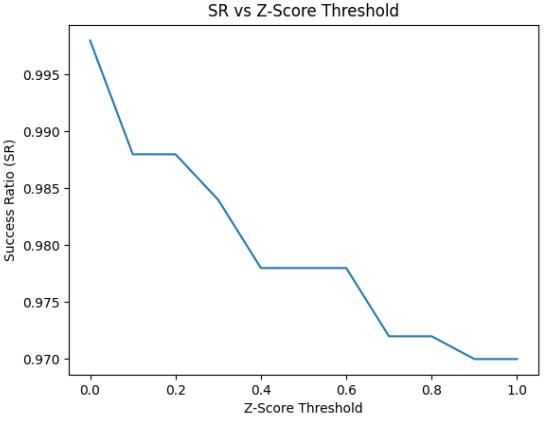
In this section, we will discuss the simulation results that will be carried out for the performance evaluation of the proposed algorithm SOFNET. We will look at some sample scenarios that correspond to the fog architecture depicted in Fig. 1. We have created a custom workload for this algorithm since it's a novel approach and required specific workload. The workload consists of mixed random jobs having different categories tags, number of instructions, arrival time and deadline.

For simulation we considered 50 cdcs, 150 fdcs, 300 end-users with total 500 jobs. The capacity of each cdc is ranging from 1000 to 50000 MIPS. The capacity of each fdc is ranging from 1000 to 5000 MIPS. The bandwidth from end user to fog is 2 to 32 mbps and from fog to cloud 64 to 1024 mbps. The storage capacity for cdc is taken as 1 to 8 GB and for fdc 128 to 1024 MB.

To evaluate the performance of our system, we analyzed the success ratio, system cost, and resource utilization for different z-score threshold values ranging from 0 to 1 with only fog and only cloud systems. The results are presented in the following charts:

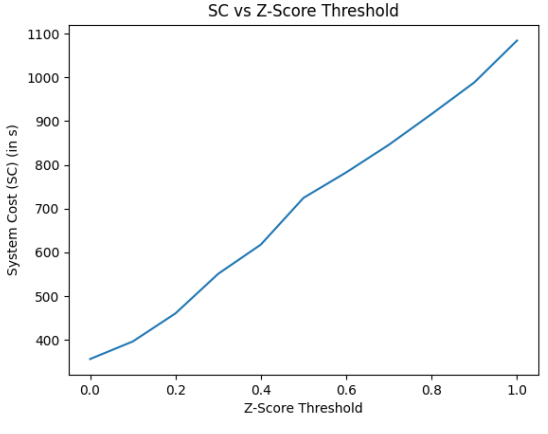
5.1 Our Proposed System

Success Ratio: The success ratio of our system was evaluated for different z-score threshold values ranging from 0 to 1. The results showed that the success ratio increased as the z-score threshold value increased, indicating that our system performed better when there were approximately equal numbers of small and big jobs. The most consistent success ratio was achieved at a z-score threshold of 0.4 to 0.6, with a value of 98%.



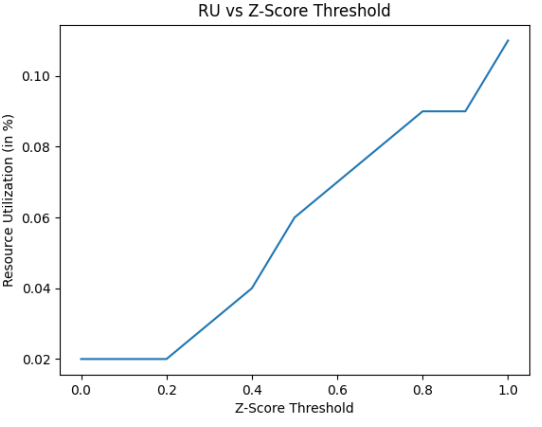
Further analysis of the results revealed that when the z-score threshold became higher, i.e. when there were more small jobs and fewer big jobs, the success ratio started to decrease. This finding is consistent with our understanding that fog devices have limited computation power and can become overloaded with jobs beyond their capacity.

System Cost: The system cost was evaluated for different z-score threshold values ranging from 0 to 1. The results showed that the system cost increased as the z-score threshold value increased, indicating that it was more expensive to run our system as the threshold became stricter. The ideal system cost was observed at a z-score threshold of 0.4 to 0.6, with a 600 sec to 800 sec value.



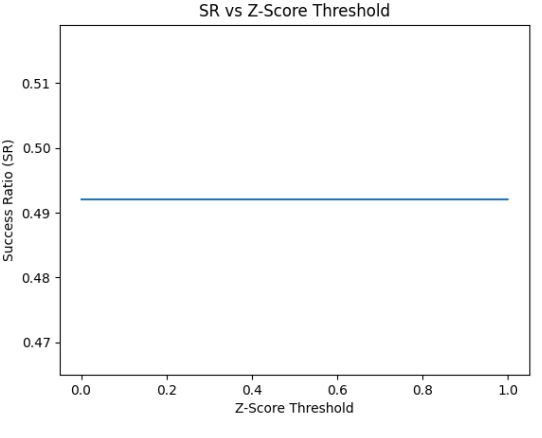
Further analysis of the results revealed that the ideal system cost, ranging between 600 to 800 sec, was observed for z-score threshold values ranging from 0.4 to 0.6. This suggests that a balance between cost and performance can be achieved by adjusting the z-score threshold value within this range.

Resource Utilization: Resource utilization increased as the z-score threshold value increased, suggesting that more resources were needed to run the system as the threshold became stricter. The ideal resource utilization was observed at a z-score threshold of 0.4 to 0.6, with a 4% to 6% value.

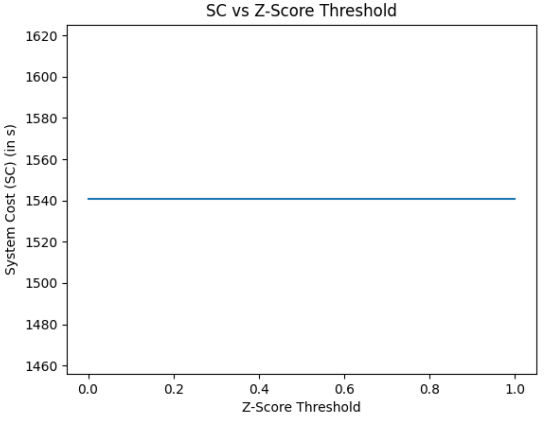


5.2 Only Fog System

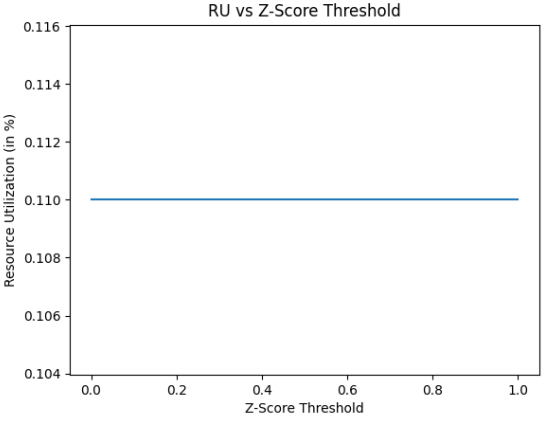
Success Ratio: The success ratio of only fog system was evaluated for different z-score threshold values ranging from 0 to 1. Since the z-score value is a term in our algorithm thus Success ratio is independent of z-score values here. Here, we observe 49% success ratio which means only half of the jobs meet their deadlines. Thus, the system performance is decreased here as compared to our algorithm.



System Cost: The system cost was evaluated for different z-score threshold values ranging from 0 to 1. As system cost is independent of the result so we have a constant system cost of 1540 sec. Here system cost is increased as fog devices have less computation power so it will take more time to execute a job. Thus, system cost is very high when compared to our algorithm.

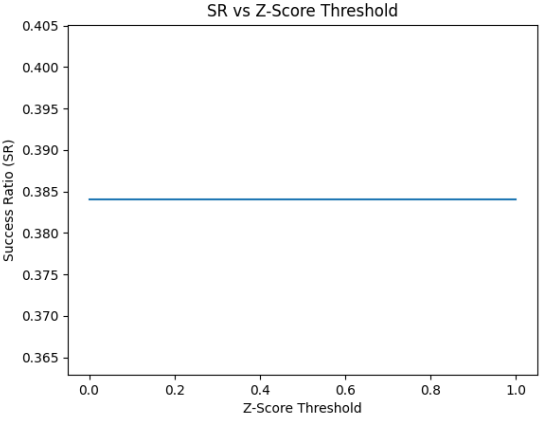


Resource Utilization: It is independent of z-score value so we have a constant resource utilization of 11%. Here resource utilization is increased when compared with our algorithm.

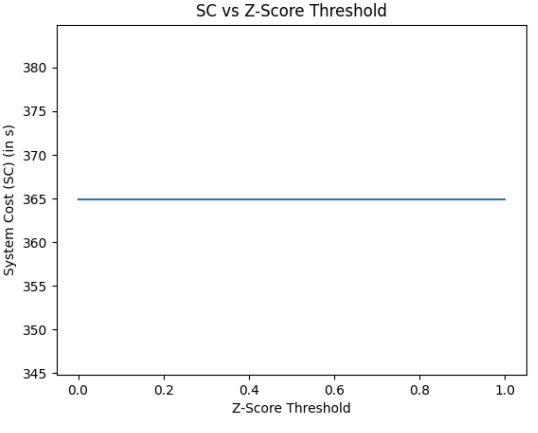


5.3 Only Cloud System

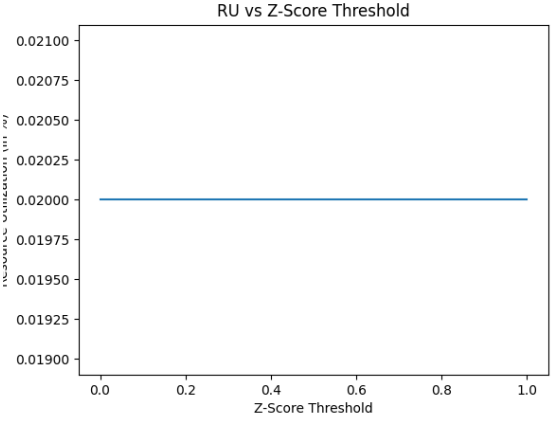
Success Ratio: The success ratio of only cloud system was evaluated for different z-score threshold values ranging from 0 to 1. Since the z-score value is a term in our algorithm, thus Success ratio is independent of the z-score values here. Thus, we observe 38% success ratio which means not even half of the jobs meet their deadlines. Thus, the system performance is decreased here as compared to our algorithm.



System Cost: The system cost was evaluated for different z-score threshold values ranging from 0 to 1. As system cost is independent of the result so we have a constant system cost of 385 sec. Here system cost is decreased as cloud devices have more computation power so it will take very less time to execute a job. Thus, system cost is low when compared to our algorithm.



Resource Utilization: It is independent of the z-score value, so we have a constant resource utilization of 2%. Here resource utilization is decreased when compared with our algorithm. Since everything is going on cloud irrespective of a big job or small job. And due to heavy storage capacity, resource utilization is much less than our algorithm. So, overall there is a trade-off between success ratio and system cost and resource utilization in every aspect.



6 Conclusion and Future work

In this paper, we presented our proposed approach for distributing job load across the fog network and scheduling real-time jobs based on various constraints, such as deadline, communication delay, size, resource utilization, and security. We categorized jobs submitted by users as classified, restricted, or public, and classified cloud and fog nodes as either native or public.

Our results showed that our algorithm achieved better real-time performance in terms of success ratio, system cost, and resource utilization. We also observed that job categories and size had an impact on real-time performance. Specifically, we found that increasing the z-score threshold value resulted in an overload of the fog beyond its capacity, leading to deadline misses and lower success ratio values.

As future work, we plan to investigate the impact of random job arrival times with random intervals on system performance. This could provide insights into how the system performs under unpredictable conditions and could help us develop more robust scheduling algorithms.

Overall, our proposed approach shows promise in improving real-time performance in fog computing systems. Our findings can inform the design of future fog computing systems and can guide the development of more effective scheduling algorithms.

References

[1] Salim Bitam, Sherali Zeadally, and Abdelhamid Mellouk. Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems*, 12(4):373–397, 2018.

[2] Tejaswini Choudhari, Melody Moh, and Teng-Sheng Moh. Prioritized task scheduling in fog computing. In *Proceedings of the ACMSE Conference*, ACMSE '18. Association for Computing Machinery, 2018. ISBN 9781450356961. doi: 10.1145/3190645.3190699. URL https://doi.org/10.1145/3190645.3190699.

[3] Pejman Hosseinioun, Maryam Kheirabadi, Seyed Reza Kamel Tabbakh, and Reza

Ghaemi. atask scheduling approaches in fog computing: a survey. *Transactions on Emerging Telecommunications Technologies*, page e3792, 2020.

[4] Ryuta Mogi, Taichiro Nakayama, and Takuya Asaka. Load balancing method for iot sensor system using multi-access edge computing. In *2018 Sixth International Symposium on Computing and Networking Workshops*, pages 75–78. IEEE, 2018.

[5] Lina Ni, Jinquan Zhang, Changjun Jiang, Chungang Yan, and Kan Yu. Resource allocation strategy in fog computing based on priced timed petri nets. *IEEE Internet of Things Journal*, 4(5):1216–1228, 2017. doi: 10.1109/JIOT.2017.2709814.

[6] Deepak Puthal, Mohammad S. Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P. Mohanty, and Albert Y. Zomaya. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Communications Magazine*, 56(5):60–65, 2018. doi: 10.1109/MCOM.2018.1700795.

[7] Anil Singh and Nitin Auluck. Load balancing aware scheduling algorithms for fog networks. *Software: Practice and Experience*, 50(11):2012–2030, 2020.

[8] Jiafu Wan, Baotong Chen, Shiyong Wang, Min Xia, Di Li, and Chengliang Liu. Fog computing for energy-aware load balancing and scheduling in smart factory. *IEEE Transactions on Industrial Informatics*, 14(10):4548–4556, 2018. doi: 10.1109/TII.2018.2818932.

[9] Juan Wang and Di Li. Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing. *Sensors*, 19(5), 2019. ISSN 1424-8220. doi: 10.3390/s19051023. URL https://www.mdpi.com/1424-8220/19/5/1023.

[10] Luxiu Yin, Juan Luo, and Haibo Luo. Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 14(10):4712–4721, 2018. doi: 10.1109/TII.2018.2851241.

[11] PeiYun Zhang, AiQing Zhang, and Ge Xu. Optimized task distribution based on task requirements and time delay in edge computing environments. *Engineering Applications of Artificial Intelligence*, 94:103774, 2020.