Journal of Information Technology Management

Print ISSN: 2008-5893

Online ISSN: 2423-5059

A Dynamic Load Balancing Architecture for Fog Computing using Tree Base Resource Arrangement and Flexible Task Prioritization

Bikash Sarma

Department of Computer Science & Engineering, National Institute of Technology Nagaland, Nagaland, India. E-mail: bikash.sarma2010@gmail.com

R. Kumar*

*Corresponding Author, Department of Electronics and Instrumentation Engineering, National Institute of Technology Nagaland, Nagaland, India. E-mail: rajagopal.kumar4@gmail.com

Themrichon Tuithung

Department of Computer Science and Engineering, National Institute of Technology Nagaland, Nagaland, India. E-mail: t_tuithung@yahoo.com

Abstract

A greater community of researchers widely studies fog computing as it reduces the massive data flow to the existing cloud-connected network and performs better for real-time systems that expect a quick response. As the fog layer plays a significant role in a fog-cloud system, all of the devices participating in fog computing must be balanced with appropriate load to upstretched the system performance. The proposed method is founded on a tree-based dynamic resources arrangement mechanism that refreshes the fog clusters created using Fuzzy C Mean (FCM) to increase the speed of resource allocation. With the help of Fuzzy rule-based load calculation and intra-cluster job allocation, the load inside the group is maintained. The system also has the facility of inter-cluster job forwarding, which works on demand. A novel load balancing strategy, Real-Time Flexi Forwarded Cluster Refreshing System (RTFRS) is proposed by which all the tasks can be handled efficiently within the fog cloud system. The proposed system is designed so that overall complexity is not upraised and becomes suitable for fog computing architecture with low processing capacity by maintaining the quality of service. Experimental results show that the proposed model outperforms standard methods and algorithms used in fog computing concerning average turnaround time, average waiting time, resource utilization, average failure rate, and the load on the gateway.

Keywords: Fog Computing; Cloud Computing; Load Balancing; Fuzzy System; Clustering.

Journal of Information Technology Management, 2023, Vol. 15, Special Issue, pp. 43-66
Published by University of Tehran, Faculty of Management
doi: https://doi.org/ 10.22059/jitm.2023.91567
Article Type: Research Paper

Received: December 22, 2022
Received in revised form: January 15, 2023
Accepted: February 20, 2023
Published online: March 15, 2023

Introduction

The extensive growth and development of smart devices and intelligent systems make humans more reliant. They do not require human intervention or require much less involvement while collecting data, processing the collected data, and making the critical decision for that processed data (Fizza et al., 2021). The Internet of Things (IoT) is advancing with all these facilities and comfort in its application field (Donassolo et al., 2019). The reason behind this growth is its diversification in various applications, which includes simple smart home applications (Lin & Bergmann, 2016). IoT systems produce helpful and precise output by enabling device-to-device and human-to-device communication, maintaining the required real-time behaviour (Lee & Lee, 2015) The main components of IoT include sensors and end devices, connectivity, data processing units, and user interfaces. Sensors collect complex multidimensional data every minute, which are then forwarded to the proper data processing unit, the cloud system, over the internet. The required output is produced convincingly with the help of a user interface (Zaidan et al., 2018).

Clouds ensure the availability and reliability of IoT systems by enabling the central computing facility. An important decision is the critical point of a flourishing industry. It can be achieved using proper analysis of previous data where the cloud shows leading behavior with its service like Infrastructure as a Service (IaaS). Other standard services provided by cloud are Platform as a Service (PaaS) and Software as a Service (SaaS) which outfit the intention of achieving Anything as a Service (Alhaddadin et al., 2014). Accessibility, scalability, flexibility, Quality of Service (QoS), Pay as you use are the key attributes that make cloud computing very different from traditional computing (Höfer & Karagiannis, 2011). The scalability feature of the cloud makes the IoT system more robust, and the accessibility creates a confidence layer before deploying any IoT application. The concept of pay per use gives more economic strength and establish confidence in industries and individual using IoT application.

Cloud as a computing, storage and analytic engine with IoT would create an impeccable blend unless the real-time, latency-sensitive application started using IoT. Latency-sensitive applications are those where delays in response are not accepted at all. The required answers must come back within the stipulated period; otherwise, the entire transaction will be discarded. Real-time systems are of three types – complex real-time systems, firm real-time

systems, and soft real-time systems (Buttazzo, 2006). In the case of a firm's real-time system, once in a while deadline may miss but the cause of degraded quality of service with no use of delayed output. The soft real-time system may use delayed production but, of course, at the cost of compromising the quality of service and overall performance. Cloud resides geologically far away from IoT applications for which the latency-sensitive applications suffer as the expected results take time to get delivered because of network and other communication delays.

Moreover, the amount of data transferred over the network is growing exponentially, and the overall growth of connected devices is expected to be 27 billion by 2024 (Shin & Ramanathan, 1994). A vast number of devices sending continuous data will lead to a congested network which affects the latency-aware applications by not getting the responses in time, causing the degraded quality of service expected to be high in today's competitive world (Sarkar & Misra, 2016).

To handle latency-sensitive applications, a new concept called fog computing was introduced by researchers of Cisco Systems (Bonomi et al., 2012). Unlike cloud systems, fog computing is not a centralized computing facility. Fog computing is a decentralized computing mechanism where devices with computing capacity and storage facilities are used to participate (Ni et al., 2017). These devices work in between the massive cloud system and tiny end devices. The entire concept of fog computing is to use all the devices responsible for sending the data from end devices to the cloud system, along with some other dedicated computing systems such as Cloudlet (Yi et al., 2015). These devices work for a stable network system and a better computing facility near the end devices. The ideal resources near the end devices can contribute to the entire processing system by reducing the network delay, usually occurring while sending the data to the cloud system. Under any circumstances, fog cannot replace the typical cloud system as fog can provide limited processing and storage facilities compared to cloud computing. Still, it can extend similar behavior by enriching computing facilities and improving the overall quality of service. Usually, once a device sends a data packet, it will get forwarded from switch to router, then router to router and finally reach the destination. While giving the response, the data packet must also pass through a similar set of devices. As a result, the entire data packet has to suffer from a communication delay along with its processing delay. The overall idea of fog computing is to stop flooding the available bandwidth with end-user data and to do the required processing of individual requests near the request generator to reduce the communication delay in getting the response (Dadashi Gavaber & Rajabzadeh, 2021).

For that, nearby devices should be equipped enough to handle such requests. Nowadays, almost all devices have high computing and storage facilities compared to the olden systems. The software used in those devices is also competent and ready to connect or support such add-on activities. Even though fog computing can help latency-sensitive applications by

deducting the communication delay from the total delay, it also has some challenges to consider.

Literature Review

Presently many researchers are contributing to the field of fog computing and its improvisation. Here in this section, a brief discussion is presented about a few previous efforts in fog computing which enlighten us to construct our proposed model. Clustering is a beneficial old technique used in many research directions.

Sun et al., (2018) the author presented the scheduling of resources for fog computing in two stages. In the first stage, they identified a fog cluster among a pool of collections by considering the proximity, available resources, and communication delay. After selecting the fog cluster, they initialize the process of selecting the appropriate resource for that particular job request inside the selected cluster. Choosing the proper help is considered a multi-objective optimization problem in fog computing as they try to minimize the service latency and maximize the system's overall stability. They have used improved non-dominated sorting genetic algorithm II (NSGA-II) to do this intracluster resource scheduling. Pertinent resource allocation is crucial to fog computing as the system has limited resources. Appropriate resource allocation may increase the system's overall performance, which in turn offers a better QoS.

Alqahtani et al., (2021) did resource scheduling based on the type of service request. Based on the kind of request, the appropriate resources are getting allocated. Resources are arranged concerning their reliability rate to give the higher reliable resources to the real time service request. They also maintain the order of algorithms so that if a higher-order algorithm deteriorates to allocate resources, the request passes on to the lower-order algorithm, imbalance load on resources used to pull the system to the back foot concerning the performance.

Singh et al., (2020) devised software to define a traffic splitter with the help of a fuzzy load balancer for a fog computing environment. They have discussed different levels of fuzzy load balancer design and also did tuning of fuzzy controllers. Their research introduced three design categories: three levels, five levels, and seven levels. These levels are the output values of the Fuzzy system. While designing the fuzzy load balancer, they considered traffic load, delay sensitivity, energy consumption, and link saturation as input variables. Finally, they conclude that the 3-level design is most effective as it reduces overhead and fewer intervals in innovation and gives better responses. Heterogeneous behavior concerning processing power, available memory, and network bandwidth are the key characteristics of fog nodes.

Beraldi et al., (2020) introduced two load-balancing algorithms for resource management in the fog computing environment. These algorithms are distributed as each node can decide whether to execute the job locally or offload it to neighboring nodes. The key idea of the first algorithm is to achieve the job request locally as soon as it arrives, but while doing so, forwarded jobs may remain abundant as the system load might reach its threshold limit. They also established a correlation between the mathematical model and simulation work. A vehicular ad-hoc network is also contributing to the betterment of the fog computing environment.

Hameed et al., (2021) proposed distributed fog computing system with vehicles as fog nodes. They make the cluster vehicle by considering its current position, speed, and direction. For efficient dynamic clustering, they predicted the vehicle's future position as once a car moves out from one cluster, it may be suitable for another collection. Each group will have a cluster head to communicate with the gateway to receive job requests and replies. After constructing the cluster, they used a capacity-based load-balancing algorithm to balance the load inside the group and among the clusters (vehicular cluster). While designing the load balancing algorithm, the author considers processing power, residual energy, and queue length as capacity parameters. The trustworthiness of fog servers plays a vital role in fog computing as fog nodes are mostly dynamic concerning latency, availability, and reliability.

Rahman et al., (2020) the author emphasizes the trustworthiness of each fog node with the help of a broker, which helps fulfil the job assigned to that fog node. While evaluating the trustworthiness, they consider availability, Quality of Service (QoS), user feedback, and cost involved. The fog server and the user will have their information registered with the broker before any transaction between them. Li et al., (2018) discusses a self-similarity-based load-balancing algorithm for a large-scale fog computing environment. The author considers the run time characteristics while addressing the load balancing issue in the fog environment. They infuse the concept of a centralized and decentralized system by empowering each fog node for their own decision of task scheduling. Fog nodes are clustered into cells; each cell has one seed responsible for communication among cells. They calculated the load limit of each cell by using a threshold policy. Once the load exceeds its limit, task distributing and task grasping algorithms take charge of the overall improvement of the system. Utilization of load balancing concerning energy consumption in the intelligent factory is essential.

Wan et al., (2018) the author discussed an energy-aware load balancing model in an intelligent factory considering the energy consumed by the device and the workload on the device. In this load balancing system, they initially built an energy consumption model for the equipment concerning the workload using fog nodes. An improved Particle Swarm Optimization (PSO) algorithm is devised to get the optimal energy consumption combination and prioritize them according to the output of the PSO model. Finally, they introduced a multi-agent-based dynamic scheduling system to distribute the workload. They also did a case

study for their proposed method in a prototype of a candy packaging unit. A secure load balancing mechanism for fog computing is addressed by Puthal et al., (2018). They consider the Edge Data Centers (EDCs) to respond to the edge request while authenticating other EDCs with the help of the cloud. They did the load balancing among the EDCs in a decentralized manner even though the cloud is involved for security purposes only. While sharing the load information among EDCs, each EDC is used to authenticate the incoming request before giving any response in the system. In their proposed system, jobs will not move around to get executed as the parent EDC does not offload the job to another EDC without knowing its present load.

Fog computing is often engaged with real-time systems even though the healthcare system is always considered mission-critical. A dynamic load-balancing approach with optimization and reinforcement learning is introduced in Talaat et al., (2020) A load-balancing agent uses a genetic algorithm to get the weight or priority of each fog server for handling client requests. On the other hand, resource allocators implement reinforcement learning methods to allocate appropriate resources to the incoming jobs. While allocating resources, the highest weight fog server used to get the high priority.

Motivation and Challenges in the Existing System

The researcher innovates various models to solve load-balancing issues of fog computing in literature. The fog computing environment's fundamental property consists of devices with low processing power, storage capacity, and power consumption. Running a complex algorithm for load balancing on a fog node may not be suitable and may decrease the overall performance in a practical scenario found in the literature. Many decentralized systems were introduced during the evaluation of the fog computing environment. Still, they did not emphasize execution time as the job used to rotate from one fog node to another to get executed. Task priority is one of the prime factors while designing the load balancing instead of device priority, as fog computing mainly deals with real-time systems. If the respective job can be leisurely executed, then fog is impractical. The task prioritization model needs to be flexible enough to handle the dynamic behavior of job requests originating from the same device, which was not discussed explicitly in the literature. In a real-world scenario, job requests will come from various sources, and all of them will have a different dimension of priority. The fog load balancing system should be capable enough to handle and prioritize the input request, improving the overall service quality. Resource allocation time must be reduced to enhance total response time. If the resources are correctly arranged, the same can be allocated to the respective job requests in less time as the property of resources is disclosed and well-known while organizing the resources. Fog load used to be very volatile as the expected job requests were real-time, and the average execution time was also significantly less. Because of its volatile nature, a dynamic load-balancing system is essential to achieve a better result.

In the proposed model, we have considered most of the issues faced in the fog environment along with a load balancing mechanism that will increase the overall performance. Similar resources are clustered together for better and quick resource match finding. The model consists of a mix of centralized and decentralized architecture. Fog Cluster Manager (FCM) gives the flavor of decentralization as there will be several FCMs, and each FCM can communicate and share load/information among themselves (inter-cluster communication). The same FCMs introduce the concept of centralization, where each FCM take care of its own cluster's fog nodes (intracluster communication). The model can prioritize and deprioritize the same task in a different situation. The Fuzzy rule-based system is used to calculate the current load of each node instead of using any complex algorithm. A tree data structure is utilized for resource arrangement inside a fog node cluster to reduce the searching time while allocating resources.

Methodology

A load balancing mechanism is one of the critical aspects of a fog computing environment, which usually solve the problem of overloaded and underloaded fog node (Kaur & Aron, 2021). Suppose loads are not distributed according to the capacity of fog nodes. In that case, it may create an inverse effect on the system's overall performance as Quality-of-Service parameters are used to get an impact highly with improper load distribution (Harnal et al., 2022). A proper resource arrangement can lead to better resource allocation, which is directly liable for a better load balancing mechanism in the fog computing environment. Minimal effort is observed in the literature concerning the arrangement of fog resources. In our model, we propose a Real-Time Flexi Forwarded Cluster Refreshing System (RTFRS) where a novel tree-based dynamic resources arrangement has been introduced to refresh the fog cluster resources. The entire mechanism of our proposed model is categorized into different stages where few are responsible for the system's initial setup, and few take care of the computational part once job requests start arriving.

Resource Clustering

The objective of this subsystem is to cluster the fog nodes based on their native properties. These properties include computational speed, storage capacity, and available bandwidth. Considering these properties, the j^{th} fog node from the set all fog nodes R is represented as R_j.

$$R_j = \{r_{computation}, r_{storage}, r_{bandwidth}\} \quad (1)$$

Fuzzy C-means clustering is a soft clustering technique. With the help of this clustering technique, the subsystem generates the belongingness value of each fog node to a particular

cluster. The degree of belongingness is represented by μ_{ij}, the membership function of i^{th} resources to j^{th} cluster. The objective function of Fuzzy C-means is represented as

$$J = \sum_{j=1}^{n}\sum_{i=1}^{c} \mu_{ij}^{m} \left\| o_j - c_i \right\|^2 \quad (2)$$

In the objective function, n represents the number of fog resources, and c represents the number of fog clusters. The Euclidean distance of the fog node o_j and the cluster centre c_i is signified with $\left\| o_j - c_j \right\|^2$. The fuzzifier is represented by the parameter m with $1 \leq m \leq \infty$. The fuzzifier is used to control the fuzziness of the output cluster.

$$c_i = \frac{1}{n_j}\sum_{j=1}^{n} (\mu_{ij})^m o_j, \qquad where\ n_i = \sum_{j=1}^{n} (\mu_{ij})^m \quad (3)$$

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}}\right)^{\frac{2}{m-1}}}, \qquad where\ d_{ij}^2 = \left\| o_j - c_i \right\|^2 \quad (4)$$

In Equation (3) calculates the cluster center, and Equation (4) calculates the membership value iteratively to minimize the objective function. The objective function is minimized when a fog node is assigned with a high membership value concerning a cluster center having almost similar properties. The Euclidean distance and low membership value in respect of other cluster centers having different properties.

Once the cluster is formed, each group's fog node with the highest resources will be marked as Fog Cluster Manager (FCM). FCM does not execute a job request. Instead, it will assign it to its cluster members. The FCM remains the same unless a re-cluster happens or it fails. If it failed, the neighboring FCM used to take charge, as all FCMs share their resource information with their neighboring FCMs.

Priority Assignment

The main objective of this subsystem is to assign a priority to the incoming request. We have derived our prioritization mechanism from the very known least laxity first algorithm called least slack schedule (Leung, 1989). The basic concept of this algorithm is to find out the process with less laxity time.

$$Lx_i = DT_i - ExT_i \quad (5)$$

In Equation (5), Lx_i is the laxity, DT_i is the deadline time and ExT_i is the execution time of the i^{th} Job. Here waiting time was not considered separately as the ExT_i include waiting

time as well to return the total execution time. Usually, the priority of a task is inversely proportional to the laxity of that task which means that if a job can be executed leisurely (more laxity), that task can be considered less critical. Based on this concept, we have calculated the value of P_i.

$$P_i = \frac{1}{Lx_i}, \qquad where\ P_i\ is\ the\ priority\ of\ the\ Job \tag{6}$$

Considering the diversity of sources from which job requests are expected, we introduce a default priority for each section. For example, life-saving medical equipment will prioritize getting any dependent data over a standard car requesting traffic details. In the priority configuration model, we are setting up a default priority for each category of devices from where the job request generates.

$$RP_i = \frac{1}{Lx_i} * Default\ Priority \tag{7}$$

In a use case scenario, a similar device may have a different priority based on the current situation around that particular device. For example, a fire station vehicle (fire truck) to control a fire incident will have higher priority over regular traffic. Still, the same fire truck returning to its base station may not have the same importance. An additional perimeter has been introduced as priority controller to reduce the priority when the emergency device is working in a normal situation so that other high priority jobs get executed in time.

$$RP_i = \left(\frac{1}{Lx_i} * Default\ Priority\right) * Priority\ Controller, \tag{8}$$

Cluster Selection

Once the job request arrives at the gateway, it will forward the same to a cluster containing similar resources as per the request of the job. The gateway maintains a resource table of clusters where each FCM update according to their latest available resources. The maintenance and working of the resource table are similar to the routing table a router maintains. In the resource table, FCM updates its cluster's highest and lowest resources. Accordingly, the gateway checks the resource table, gets the best match, and forward the job request to the concern cluster's FCM.

Load Calculation

Load calculation of each fog node is a continuous process unless and until the fog node is dead. The FCM calculates a load of its cluster member whenever a fog node is assigned with a job request and when the fog node finishes executing a job request. Load recalculation is essential when the fog node is assigned with any job request because its available resource

will be reduced, and the available load will be increased. In the same way, the available load will be less, and resources will be more when the fog node finishes executing a job, so another recalculation is required.

$$R: A\left(Avl_{CPU}, Avl_{Storage}, Avl_{Bandwidth}\right) \rightarrow B(Current_{Load}) \quad (9)$$

A fuzzy rule-based load calculation mechanism has been used to calculate the current load of a fog node. The fuzzy rule-based system uses if-then statements, also known as conditional statements, to give any output. Our expected output is the current load based on the input parameter such as available CPU, storage, and bandwidth, which is represented as a fuzzy rule in Equation (9). Each FCM will take care of its cluster for load calculation using this proposed method.

Resource Arrangement

The proper arrangement of resources will improve the performance of fog computing as the required resources can be identified and allocated in less time. The resource allocation problem is not just about finding available resources but also allocating that in a minimal amount of time. In the literature, we found that resource arrangement is limited to resource clustering only. We did not find any literature focusing on the resource arrangement inside the cluster for better performance. In our proposed model, we use an AVL tree data structure to keep the load information of each fog node.

The main motive for using an AVL tree data structure is to keep the fog node arranged in respect of their load within the fog cluster. When a job request arrives at FCM, it is easy for FCM to identify the best or ideal resource which can execute the job inside the resource cluster. Conventionally, once the job request arrives at FCM, it has to check each fog node to get the most available resources. Still, in our proposed model, FCM picks the best match or most available resources by traversing the AVL tree, which is much more efficient concerning time complexity.

Resource Allocation and Offloading

Resource allocation is a runtime activity in any computational system as it entirely depends on the dynamic job request where all the incoming tasks are forwarded to servers for execution. A balance resource utilization is very much essential to improvise the performance of a resource allocation model. In fog computing, resources are used to get allocated once available and match the requirement for that particular job request, as fog resources are limited in nature. A balance resource utilization also improves the performance of a fog computing system. In our proposed model, we implemented two-way resource allocation, namely intra-cluster resources allocation and inter-cluster offloading. In intra-cluster resource allocation, the FCM will allocate the best available resources to the input job request based on

the fuzzy load calculation and resource arrangement in the AVL tree. Once the resource gets allocated, the FCM will update the load of that resource and rearrange it in the AVL tree considering the current load. In certain situations, the FCM may not be able to find the required resource inside its cluster. In that case, the respective FCM will have communication with its neighboring FCM for the availability of required resources. Suppose the requested resources are available in another resource cluster and agreed to take over the job request. In that case, the current FCM will offload the job request (inter-cluster offloading) to that particular FCM where the resources are available. If the current FCM receives no response within a period, it will offload the job request to the cloud.

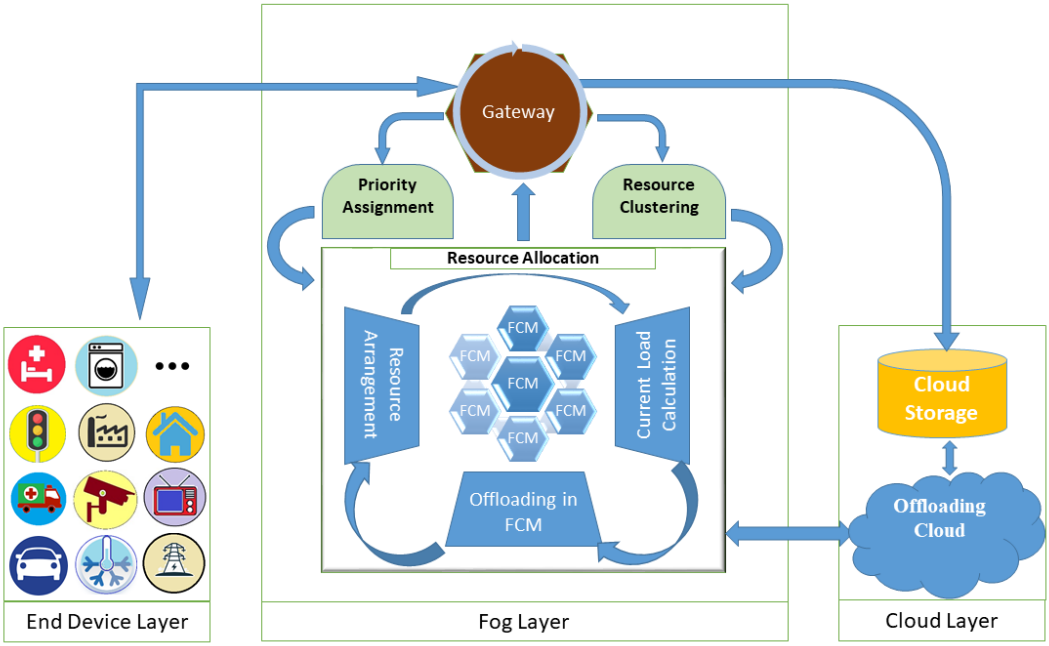

Figure 1. Architecture of the proposed model

System Architecture

The entire system architecture is subdivided into three layers: the end device layer, fog layer, and cloud layer, which are the essential components of fog computing. The end device layer consists of the consumer devices, which generate the job request and submit it to the fog layer for execution to get the required output. The fog layer executes the received job with the help of fog servers according to their priority using various algorithms. Meanwhile, the cloud layer provides support for additional computing facilities and permanent data storage. When fog servers cannot execute a received job for their lack of computing facility or overload, it offloads the job to the cloud so that the respective job gets executed in time.

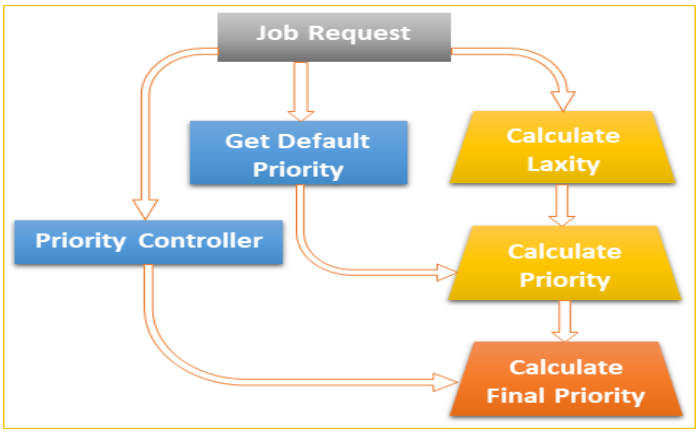

Figure 2. Priority calculation system with a control mechanism

The proposed system architecture is represented in Figure 1, and the step-by-step working strategy is explained below

• In the end device layer, devices generate the Job request and send it to their respective gateways.

• Upon receiving the job request, the gateway does its internal processing by allocating the priority to each job request using our proposed priority assignment algorithm.

• In the fog layer, all the available resources are clustered according to their computational speed, storage capacity, and bandwidth using the Fuzzy C Mean clustering technique.

• The device with the cluster's highest resources is considered the FCM for that particular cluster.

• Each resources cluster will have an FCM which directly communicates with the gateway.

• All the FCMs advertise their high and low-capacity fog nodes concerning their resources periodically.

• Gateway assigns the job request to the FCM, considering its priority and resource requirement. Before assigning the job request, the gateway checks the resource advertisement given by FCM to verify resources.

• Each FCM arranges the fog resources in the AVL tree considering its computational speed inside the resource cluster once the cluster development is over.

• As soon as the job request arrives, FCM gets the fog server with the highest resources assigned.

• If the requested resources are unavailable in the resource cluster, FCM checks other clusters' FCM for offloading.

• If other clusters are not ready to take the load, FCM offloads the task to the cloud.

Design and Implementation

In this part, we will discuss implementing our proposed load balancing and resource allocation algorithm, where utmost care has been taken to realize the real scenario of practical utilization using programming tools. All the subsystems which contributed to the proposed system are explained with their algorithms and respective parameters. A proper simulated environment has been created using appropriate functions and structured libraries of MatLab and Python programming tools.

Algorithm 1: Resource Clustering (Fuzzy C Mean)

Input: Input: Fog nodes (computational speed, storage capacity, available bandwidth), No of fog cluster K, Maximum iteration

Output: Clusters of Fog Nodes

1. Initialize the Fog_Node Weight
2. FWij=Random weight to each (K) Cluster
3. While (Itr<Max_Itr OR Prv_Centriod != Cur_Centroid)
4. While(Cluster_no<Max_Cluster)
5. Compute the cluster centre using equation 3
6. End While
7. While(Fog_Node<Max_Fog_Node)
8. Compute the new membership function using equation 4
9. End While
10. End While

Resource clustering used to take place while setting up the system. Our proposed model uses the fuzzy C Mean clustering technique for resource clustering. The implementation of the clustering system is stated in Algorithm 1. Assignment of priority is the first operation of an incoming task so that it can be identified among the pool of earlier tasks waiting in the queue. Figure 2 shows the process of priority assignment of our proposed model.

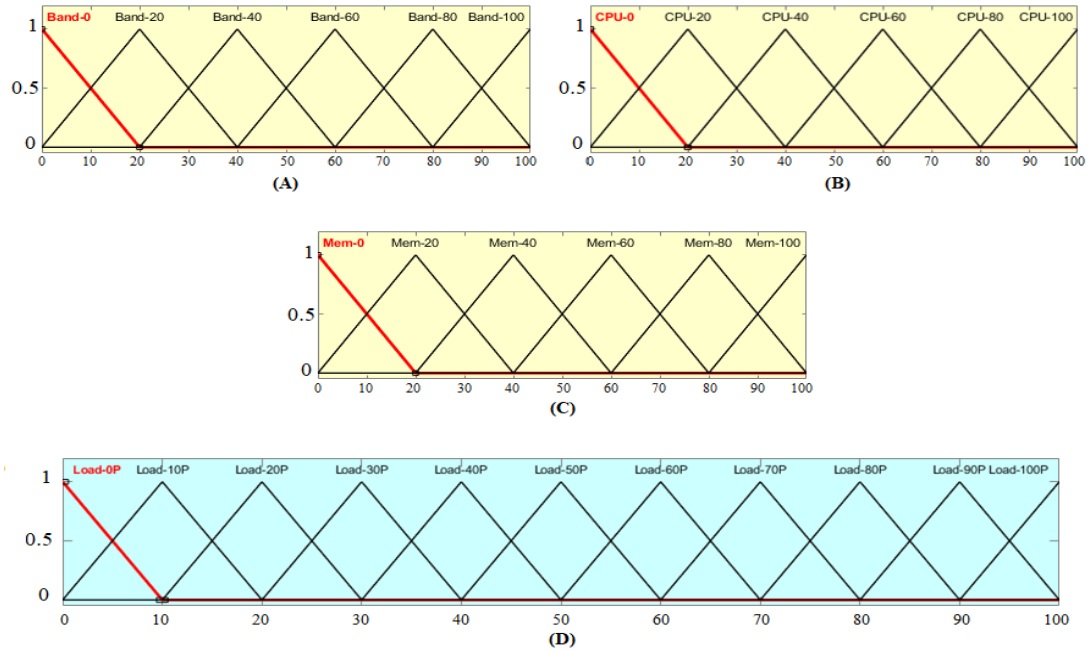

Figure 3. Membership Function for input variable (A) Bandwidth, (B) CPU, (C) Storage and (D) Variable Output Load

During the process of priority calculation, once the job arrives, the system calculates its laxity. Each job request will carry its default priority with which the initial priority is calculated. Finally, with the priority controller mechanism, the absolute priority of the job request is calculated. The priority controller mechanism is introduced to reduce the default priority by considering the present status of that particular job request. The priority assignment mechanism is implemented in our proposed model using Algorithm 2. The load calculation process was implemented using Matlab Fuzzy Logic Toolbox. During the design of fuzzy rules, CPU, Storage and bandwidth were considered input parameters. All the input parameters are converted into five different linguistic variables (0%, 20%, 40%, 60%, 80% and 100%) based on their uses. The membership function for the input variable (CPU, Storage, bandwidth) and the output variable (Load) is shown in Figure 3. With the help of fuzzy rules base written in If-Then format, we have established the relation between the input and output variables. The defuzzification dialogue window of the Matlab Fuzzy Logic designer shown in Figure 4 represents the rules and defuzzification of linguistic input values to the crisp output value.

Algorithm 2: Priority Assignment Algorithm

Input: Job Request with Deadline & Execution Time, Default Priority, Priority Controller

Output: Priority of the Job Request

1. Get the Job_Request to calculate the priority
2. Calculate the Laxity of the Job using equation no 5
3. Calculate the initial priority using equation 6

4. Calculate Default Priority based on device category
5. If(Priority_Controller ==1)
6. Calculate the priority using equation 7
7. Else if(Priority_Controller>0 && Priority_Controller<1)
8. Calculate the priority using equation 8
9. Else
10. Invalid Input
11. End if

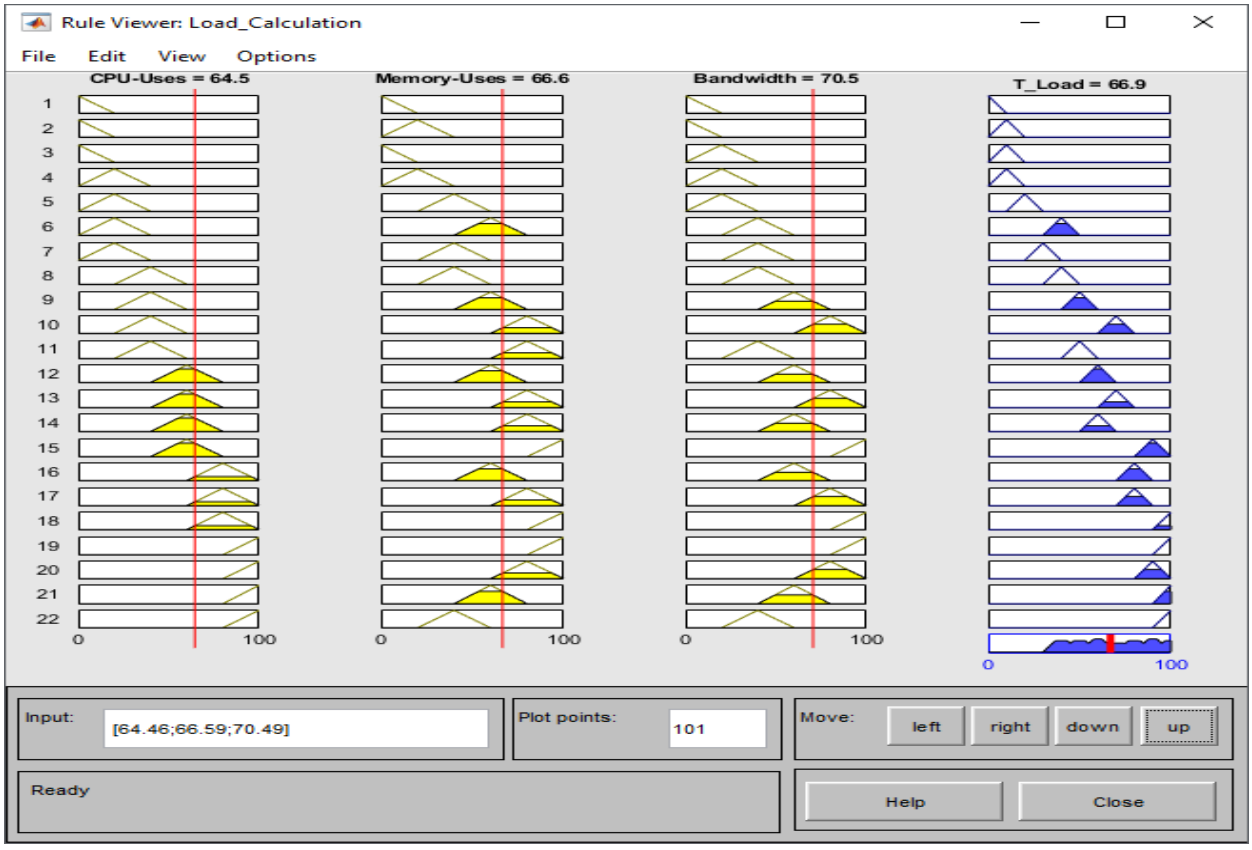

Figure 4. Rules with Defuzzied output

All the resources are appropriately arranged using an AVL tree inside the cluster. The arrangement is based on available resources to get the best suitable resources per the demand of individual job requests. The tree gets updated for each job assignment to a Fog node as the available resources reduce. The same happens when the job execution gets over. For both situations, Algorithm 3 updates the AVL tree to arrange the resources inside the cluster.

Algorithm 3: Resource Arrangement Inside Cluster
Input: Fog Node
Output: Arranged list of Fog Nodes
1. Function AVL_Update(Fog_Node)
2. Traverse the AVL Tree to get the Fog_Node.

3. Delete the Fog_Node
4. Fog_New_Load = Fuzzy_Load_Cal(Fog_Node)
5. Insert the Fog_Node concerning the Fog_New_Load.

The proposed model categorizes the job assignment process into two different subsystems. As soon as a job request arrives, the FCM tries to assign the job to the fog nodes inside the cluster, termed an inter-cluster job assignment. But suppose for unavailability of resources at runtime, the FCM fails to assign the job to any fog node inside the cluster. In that case, it will try to get the resources from neighboring clusters, considered under inter-cluster offloading. The entire job assignment process with both subsystems is implemented using Algorithm 4.

Algorithm 4: Job Assignment / Resource Allocation
Input: Job_Request, Req_CPU, Oth_Req_Res, Avl_Fog, Cloud_Exe_Time
Output: Assign the requested job to a Fog Node

```
1.  Get Most_ Available _CPU and Lest_Available_CPU from each Fog_Cluster
2.  If(Req_CPU<Most_Avlbl_CPU && Req_CPU>Lest_Avlbl_CPU)
3.      For All Clusters check
4.          Select_Fog_Cluster=Cluster with more(Avg_Avlbl_CPU – Req_CPU)
5.      End For
6.      Cluster_Head=Get_Cluster_Head(Secect_Fog_Cluster)
7.      Assign the job request to Cluster Head.
8.      Fog_Load_List=inorder_traverse(Avl_Fog)
9.      For All Fog_Node in Fog_Load_List with Req_CPU<Tot_Avl_CPU
10.         If(Req_CPU<Cur_Avl_CPU)
11.             If(Oth_Req_Res<Cur_Oth_Avl_Res)
12.                 Assign the Job_Request to the Fog_Node
13.                 AVL_Update(Fog_Node)
14.                 Assign = True
15.             End If
16.         End If
17.     End For
18.     If(Assign !=True)
19.         Check with neighbouring Fog_Nodes. With the same Algorithm, Steps 6 –17
20.     Else if(Dead_Time>Cloud_Exe_Time)
21.         Forward the Job to Cloud
22.     Else
23.         Drop the Job
24.     End If
25. Else if(Dead_Time>Cloud_Exe_Time)
26.     Forward the Job to Cloud
27. Else
28.     Drop the Job
29. End If
```

Results

This section will present a comparative evaluation of our proposed model, a Real-Time Flexi Forwarded Cluster Refreshing System (RTFRS) with some standard methods, algorithms, and specific situations in fog computing. While doing the comparative analysis of our proposed method, we consider Decentralized Systems (DS), Smart Gateway Systems without Resources Clustering (SGS), Standard Round Robin Mechanism (RRM) of job distribution among fog nodes, and real time flexi forwarding fog computing. The resource arrangement inside the cluster is a novel approach in fog computing introduced in this work. We compare our load-balancing approach with and without resource arrangement inside the fog cluster to realize the efficiency of this approach. A Decentralized System (DS) works with a non-smart lightweight Gateway. In this system, the gateway is responsible for forwarding the job request to any random fog node. Gateway does not maintain any state of the job request. Each fog node can decide whether to execute or forward the incoming job request. In this way, the system becomes autonomous, and decision taking responsibility is distributed among the fog nodes. RRM is the typical Round Robin method of job distribution where jobs are allocated to a stack of resources one after another without considering either the capacity of the resources or the requirement of the job request. Round Robin's mechanism of job distribution is one of the popular methods as it is less complex and straightforward to implement. Smart Gateway System without Resources Clustering (SGS) is the traditional system of fog computing, where the gateway is used to take care of the entire system.

The fog nodes directly communicate with the gateway, and the load distribution is only taken care of by the gateway. Real-Time Flexi Forwarded System (RTFS) is a subsystem of our proposed model without including a resource arrangement mechanism (refreshing) inside the cluster. We consider this system for our comparative analysis to represent the critical observation of resource arrangement inside a fog cluster. One of the significant roles of fog computing is to deliver the service in a minimal amount of time, reducing network delay. Turnaround time refers to the time to complete a job. Figure 5 displays the turnaround time of various approaches we had simulated, including our proposed method with a range of job requests starting from 100 to 700. The proposed method has the lowest average turnaround time compared to the other methods. As the gateway assigns the job request to the resource cluster based on its resource requirement and the FCM periodically updates the gateway about the availability of its resources inside the cluster, the system has a high probability of assigning the required resources to the job request in the first run itself. Moreover, once the job arrives inside the cluster, FCM immediately assigns the job to the most available fog resources reducing the execution time. FCM is used to maintain the load ratio of all its fog resources inside the cluster.

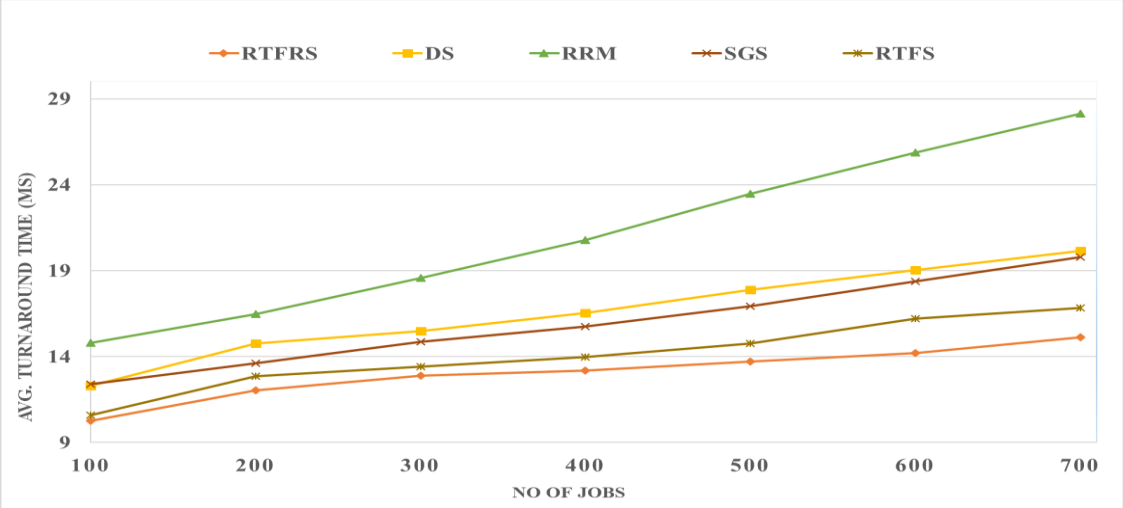

Figure 5. Average turnaround time of each job

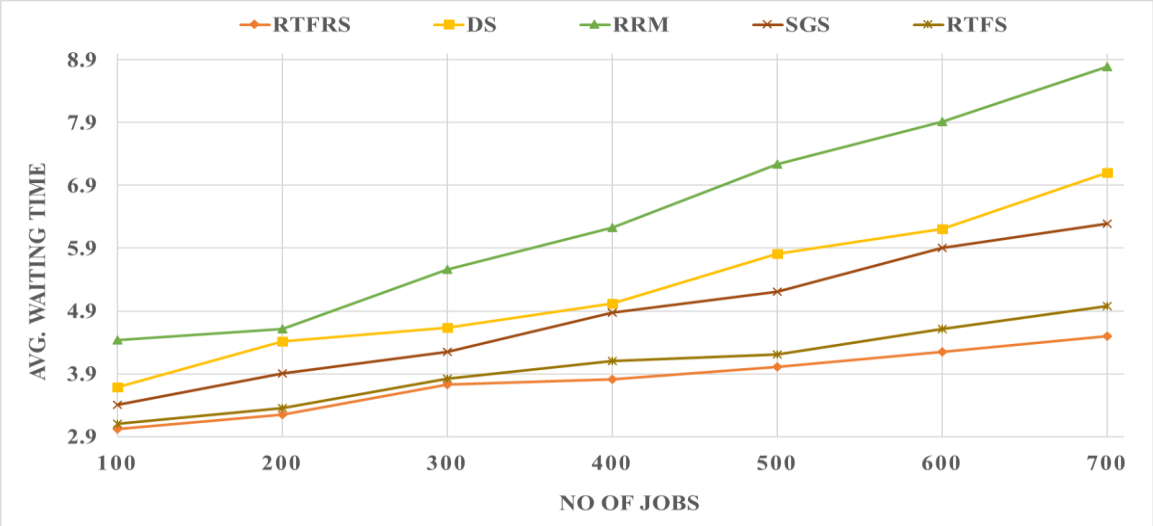

Figure 6. Average waiting time of each job

The average turnaround time is proportional to the waiting time of a job. If waiting time increases eventually, the turnaround time for a job request will also increase. Figure 6 shows the average waiting time for a range of jobs (100-700). From the figure, it can be easily identified that the proposed method outperforms all other traditional methods considered during the experiment and presented. The working principle of FCM is also one of the critical points for reducing waiting time for our proposed method. FCM first tries to allocate the job request to its resources inside the fog cluster, where resources are arranged for easy allocation. If FCM does not find matching resources, it immediately checks from its neighboring FCMs. If no FCM replies, the job gets forwarded to the cloud. During the entire process, the parent FCM keeps ownership of the job unless it finds a suitable resource or is forwarded to the cloud.

Table 1. Average load for group of nodes

Methods	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9	Group 10
RTFRS	70	55	62	75	53	65	59	57	73	67
DS	90	29	34	80	63	84	25	44	51	62
RRM	81	45	61	20	37	57	90	55	71	80
SGS	65	71	68	76	55	68	73	60	79	72
RTFS	68	57	60	55	70	74	62	69	51	65

Proper utilization of resources is also one of the significant attributes while developing a fog computing system, as fog has limited resources. Figure 7 represents the resource utilization behavior of nodes using different algorithms and approaches. Here we made a group of nodes having similar loads after the execution of our proposed algorithm. These groups were created to demonstrate the load difference of the same resources while using different approaches. All the algorithms were executed, for which Table 1 shows the results. Here it was observed that for RTFRS, the average load of Group 4 is 75% and Group 5 is 53%, whereas in the case of DS, the load for Group 1 is 90%, and Group 7 is 25%.

On the other hand, for RRM, Group 4 has a 20% load, and Group 7 has a 90% load. The outcome of this scenario is the difference between the loads among resources. From Table 2, it is observed that for RTFRS, this difference in average load is low (22), whereas, for DS (65) and RRM (70), it is high. Usually, in an ideal case, all fog nodes should have almost similar loads to make the system more efficient. Unbalance load reduce the efficiency of the entire system where some nodes are overloaded with jobs, increasing execution time, and some are ideal, waiting for job request to arrive. From Figure 7, we can understand the load difference among the group of nodes, and it was observed that the proposed RTFRS system had distributed the load in a better way. RTFRS can distribute the load in a reformed way as it has two levels of resource allocation filter. The first one is in the gateway, and the second is in FCM. The FCM periodically updates the gateway about its available resources and present loads. Accordingly, the gateway assigns/redirects the job request to the available or underloaded resource cluster. The simulation was carried out with a job load of 350 – 400 and 50 fog nodes having different types of resources.

Table 2. Difference of load for various methods

Method	Highest Load	Lowest Load	Difference
RTFRS	75	53	22
DS	90	25	65
RRM	90	20	70
SGS	79	55	24
RTFS	74	51	23

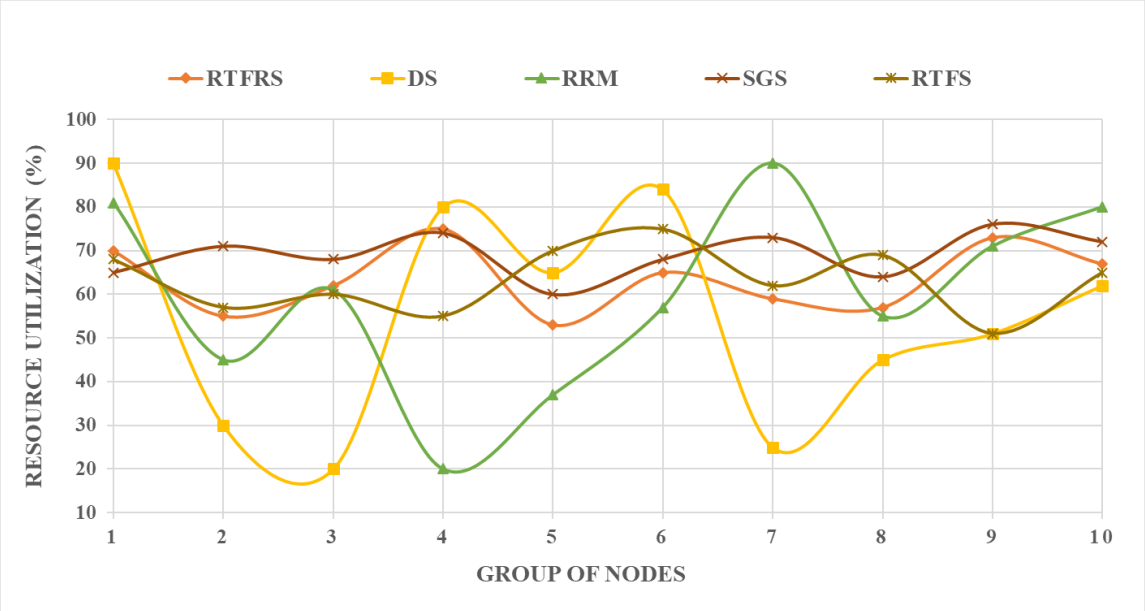

Figure 7. Utilization of resources in percentage

All the jobs will not get executed in a single request, but the system should be capable enough to reduce the failure rate. Figure 8 shows the graphical representation of average failure rates where the proposed method, RTFRS, has the lowest, 2.67%. Table 3 gives detailed statistics for the number of jobs that failed to execute when job loads increased from 100 to 700 for different approaches. Here it has been observed that when the job load is low, the RTFRS and RTFS performance is almost the same, but when the load increases, the RTFRS performs better. It is mainly because of the arrangement of the resources inside the fog cluster using the tree structure. Load distribution is easy and faster where resources are arranged properly, which helps our proposed model perform better. Moreover, the priority assignment model of our proposed model is efficient as it can alter the priority of similar job requests with less importance allowing the most critical job request for execution.

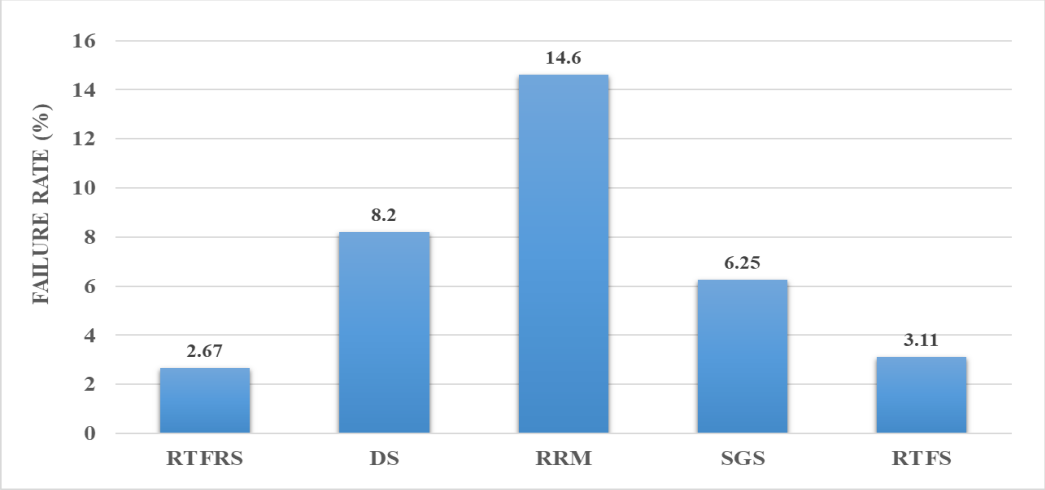

Figure 8. Average failure rate of incoming job request

Table 3. Statistics of failed job requests when loading increases

Job Load	RTFRS		DS		RRM		SGS		RTFS	
	Failed Task	Rate (%)	Failed Task	Rate (%)	Failed Task	Rate (%)	Failed Task	Rate (%)	Failed Task	Rate (%)
100	2	2	3	3	11	11	2	2	2	2
200	4	2	11	5.5	23	11.5	6	3	4	2
300	6	2	26	8.67	39	13	11	3.67	6	2
400	10	2.5	37	9.254	62	15.5	21	5.25	10	2.5
500	14	2.8	49	9.8	73	14.6	38	7.6	17	3.4
600	21	3.5	63	10.5	98	16.34	57	9.5	27	4.5
700	27	3.86	75	10.72	142	20.28	89	12.71	38	5.42

Gateway is one of the essential components of the fog computing system, which is responsible for overall performance. The workload on the gateway should be adequately maintained to save it from crashing with overload because if the gateway fails, the entire system fails. Of course, redundancy will back the system up, but the available system may fail for the exact cause. The simulated result of our experiment in Figure 9 shows the loads on the gateway with 100 to 700 job requests where SGS has the highest load and DS has the lowest load on the gateway. Our proposed system has a moderate load of 48.22% on average. In SGS, the gateway is the single point where all the jobs are accumulated for execution. Accordingly, the jobs get distributed, maintaining load among fog nodes, and finally, get back the results after execution. It is a centralized system. But in the case of DS, the gateway forwards the task to a random fog node without worrying about the present load of that system. In the case of RTFRS, the gateway has less responsibility than SGS, as FCM takes care of its local fog nodes.

Figure 9. Load on Gateway based on the job request

The experiment results found that the proposed system has maintained a decent load on the gateway while forwarding jobs and maintaining the fog cluster load records. The gateway has an extensive role in the success of the fog system, and overburdening the gateway may

lead to the failure of the entire system. Clustering all the resources makes it easy for the gateway and Fog Cluster Manager (FCM) to find the best available resources for allocation to the requested job. The produced results represent a balance resource utilization because of the coordination between the gateway and FCM. The concept of task prioritization with flexible priority assignment makes the system more realistic. Implementing the same will not increase the complexity of the entire system, which was reflected in output results having fewer failure rates for our proposed system.

The arrangement of resources inside the resource cluster, as well as the working principle of FCM, makes the system convenient. Implementing the AVL tree data structure increases the search and resource allocation speed, reducing the simulated results' waiting time and turnaround time. The proposed system has outperformed most of the parameters considered in the simulated result. However, consideration of other things may be helpful in fog computing. Reclustering of the resources has not been considered during the experiment because of time constraints and complexity, which may be a system's shortcoming. During the experiment, the gateway failure situation has also not been simulated.

Conclusion

The recent development of connected devices increased the use of the internet and cloud systems, where real-time systems may suffer due to bandwidth bottlenecks. Fog computing has the potential to support real-time systems, and effort was given to justify the use of this system in realistic situations. The proposed RTFRS method mainly focused on the load balancing among the resources using distributed characteristics of fog computing where multiple fog manager takes care of the job assignment based on the current load. The flexible job prioritization method increases the system's reliability, and the resource arrangement inside the cluster enhances the performance by increasing the allocation speed. Even though the proposed model was not implemented using real hardware and accurate data, the utmost care has been taken to depict the real-time situation while doing the simulated experiment.

Conflict of interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article

References

Alhaddadin, F., Liu, W., & Gutiérrez, J. A. (2014, December). A user profile-aware policy-based management framework for greening the cloud. In 2014 IEEE Fourth International Conference on Big Data and Cloud Computing (pp. 682-687). IEEE.

Alqahtani, F., Amoon, M., & Nasr, A. A. (2021). Reliable scheduling and load balancing for requests in cloud-fog computing. Peer-to-Peer Networking and Applications, 14(4), 1905-1916.

Beraldi, R., Canali, C., Lancellotti, R., & Mattia, G. P. (2020). Distributed load balancing for heterogeneous fog computing infrastructures in smart cities. Pervasive and Mobile Computing, 67, 101221.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012, August). Fog computing and its role in the internet of things. In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (pp. 13-16).

Buttazzo, G. (2006). Research trends in real-time computing for embedded systems. ACM SIGBED Review, 3(3), 1-10.

Dadashi Gavaber, M., & Rajabzadeh, A. (2021). BADEP: bandwidth and delay efficient application placement in fog-based IoT systems. Transactions on Emerging Telecommunications Technologies, 32(8), e4136.

Donassolo, B., Fajjari, I., Legrand, A., & Mertikopoulos, P. (2019, January). Fog based framework for IoT service provisioning. In 2019 16th IEEE annual consumer communications & networking conference (CCNC) (pp. 1-6). IEEE.

Fizza, K., Banerjee, A., Mitra, K., Jayaraman, P. P., Ranjan, R., Patel, P., & Georgakopoulos, D. (2021). QoE in IoT: a vision, survey and future directions. Discover Internet of Things, 1(1), 1-14.

Hameed, A. R., ul Islam, S., Ahmad, I., & Munir, K. (2021). Energy-and performance-aware load-balancing in vehicular fog computing. Sustainable Computing: Informatics and Systems, 30, 100454.

Harnal, S., Sharma, G., Seth, N., & Mishra, R. D. (2022). Load Balancing in Fog Computing Using QoS. In Energy Conservation Solutions for Fog-Edge Computing Paradigms (pp. 147-172). Springer, Singapore.

Höfer, C. N., & Karagiannis, G. (2011). Cloud computing services: taxonomy and comparison. Journal of Internet Services and Applications, 2(2), 81-94.

Kaur, M., & Aron, R. (2021). A systematic study of load balancing approaches in the fog computing environment. The Journal of Supercomputing, 77(8), 9202-9247.

Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business horizons, 58(4), 431-440.

Leung, J. Y. T. (1989). A new algorithm for scheduling periodic, real-time tasks. Algorithmica, 4(1), 209-219.

Li, C., Zhuang, H., Wang, Q., & Zhou, X. (2018). SSLB: self-similarity-based load balancing for large-scale fog computing. Arabian Journal for Science and Engineering, 43(12), 7487-7498.

Lin, H., & Bergmann, N. W. (2016). IoT privacy and security challenges for smart home environments. Information, 7(3), 44.

Ni, J., Zhang, K., Lin, X., & Shen, X. (2017). Securing fog computing for internet of things applications: Challenges and solutions. IEEE Communications Surveys & Tutorials, 20(1), 601-628.

Puthal, D., Obaidat, M. S., Nanda, P., Prasad, M., Mohanty, S. P., & Zomaya, A. Y. (2018). Secure and sustainable load balancing of edge data centers in fog computing. IEEE Communications Magazine, 56(5), 60-65.

Rahman, F. H., Au, T. W., Newaz, S. S., Suhaili, W. S., & Lee, G. M. (2020). Find my trustworthy fogs: A fuzzy-based trust evaluation framework. Future Generation Computer Systems, 109, 562-572.

Sarkar, S., & Misra, S. (2016). Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. Iet Networks, 5(2), 23-29.

Shin, K. G., & Ramanathan, P. (1994). Real-time computing: A new discipline of computer science and engineering. Proceedings of the IEEE, 82(1), 6-24.

Singh, S. P., Sharma, A., & Kumar, R. (2020). Design and exploration of load balancers for fog computing using fuzzy logic. Simulation Modelling Practice and Theory, 101, 102017.

Sun, Y., Lin, F., & Xu, H. (2018). Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. Wireless Personal Communications, 102(2), 1369-1385.

Talaat, F. M., Saraya, M. S., Saleh, A. I., Ali, H. A., & Ali, S. H. (2020). A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment. Journal of Ambient Intelligence and Humanized Computing, 11(11), 4951-4966.

Wan, J., Chen, B., Wang, S., Xia, M., Li, D., & Liu, C. (2018). Fog computing for energy-aware load balancing and scheduling in smart factory. IEEE Transactions on Industrial Informatics, 14(10), 4548-4556.

Yi, S., Li, C., & Li, Q. (2015, June). A survey of fog computing: concepts, applications and issues. In Proceedings of the 2015 workshop on mobile big data (pp. 37-42).

Zaidan, A. A., Zaidan, B. B., Qahtan, M. Y., Albahri, O. S., Albahri, A. S., Alaa, M., ... & Lim, C. K. (2018). A survey on communication components for IoT-based technologies in smart homes. Telecommunication Systems, 69(1), 1-25.

Bibliographic information of this paper for citing:

Bikash, Sarma; Kumar, R. & Themrichon, Tuithung (2023). A Dynamic Load Balancing Architecture for Fog Computing using Tree Base Resource Arrangement and Flexible Task Prioritization. *Journal of Information Technology Management*, 15 (Special Issue), 43-66. https://doi.org/10.22059/jitm.2023.91567