

Developing a Methodology for IOT Load Distribution within Edge Computing

Abdullah Farhan Mahdi¹, Aymen Mudheher Badr², Israa Adnan Mishkal³

¹Department of Computer Engineering, Faculty of Engineering, University of Diyala, Diyala, Iraq.

²Department of Political Science, College of Law and Political Science, University of Diyala, Diyala, Iraq.

³Department of Computer, Faculty of Science, University of Diyala, Diyala, Iraq.

Article Information

Article history:

Received: 09, 07, 2024

Revised: 25, 11, 2024

Accepted: 20, 09, 2024

Published: 30,12, 2024

Keywords:

Internet of Things

Edge computing

Load balancing

Standard deviation variance

Execution time

Abstract

Optimizing task distribution and resource allocation becomes crucial with the exponential growth of IoT devices and the proliferation of edge computing. On the other hand, building such a flexible model about resources inside a heterogeneous climate is difficult. Also, the increasing demand for IoT services necessitated working to reduce the time delay by accomplishing successful load balancing. The objective of this study is to enhance load balancing by ensuring equitable allocation of resources among workloads, thereby enhancing Quality of Service (QoS) in cloud computing and minimizing processing time (PT), hence decreasing response time (RT). Our methodology presents a decentralized system with multiple agents that utilize the nodes in the edge and the cloud to distribute the workload caused by incoming tasks and the cost of performing those tasks. A collaborative model is followed to allocate the tasks to the resources to increase the utilization of available resources.



Corresponding Author:

Aymen Mudheher Badr

Department of Political Science, College of Law and Political Science,

University of Diyala

Diyala, Baquba, Iraq

Email: Aymen.m.badr@uodiyala.edu.iq



1. INTRODUCTION

The Internet of Things (IoT) has an unprecedented effect on how data is shared and processed [1]. An estimated 125 billion devices are projected to be operational on the Internet of Things (IoT) by 2030. These devices produce a gigantic measure of data sent to the cloud for processing, increasing the load on data centers in the cloud and networks overall [2]. Notwithstanding the many benefits of computing, numerous IoT applications can't run on the cloud efficiently [3]. Moreover, IoT devices can be categorized in various ways based on their functionalities, communication protocols, and application domains, such as wearable devices, smart home devices, industrial IoT devices, smart appliances, connected health devices, smart cars and transportation, smart cities infrastructure, agricultural IoT devices, environmental monitoring devices, and retail and inventory management [4]. On the other hand, running applications on the cloud, which is far from users, leads to unpredictable latency, as well as security and privacy concerns of data traveling across public networks to remote cloud centers. However, many edge nodes have resources that we can utilize to decrease bandwidth and latency through the networks [5].

Edge computing is an architecture that is utilized to decrease traffic over the network and improve QoS for applications that are sensitive to delay [6]-[8]. The allocation of IoT services on the existing resources within the cloud-to-edge hierarchy is a critical obstacle in edge computing. This is because the dynamic nature of IoT services and their dispersed locations across a large geographical area are such that inadequate distribution of the load will result in a decline in Quality of Service (QoS) [9].

Some recent studies have addressed the problem of load balancing in the cloud. In [10], fostering an algorithm for load adjusting was introduced utilizing the PSO algorithm. The algorithm creates a group of individuals. Each potential solution for task allocation is represented by a vector of length n , where n denotes the job number, and each element is a random number ranging from 1 to m , where m is the number of the virtual machine. For each individual, the absolute execution time is calculated.

The individual exhibiting the shortest execution time is selected, and the tasks are allocated to the predetermined virtual machines inside the selected vector. An inherent problem of this approach is the premature convergence in reaching the solution when the search space is limited. This proposal focuses on identifying the optimal solution within the selected search space, rather than the most optimal solution that can be achieved inside the cloud center. If a large initial local area is used, it will result in a significant increase in the time required to find the best candidate, leading to a substantial increase in the waiting time for tasks until they are scheduled and ultimately an increase in the response time.

The article [11] proposed a load-balancing method using the Ant Colony Optimization (ACO) algorithm for dynamic cloud load balancing. In this method, virtual machines are evaluated based on the resources allocated to each of them. Then, tasks are assigned to the highest-rated virtual machine based on their arrival at the data center. This process does not consider the task's size, so the uneven distribution of tasks may cause some nodes to be overloaded and others under load conditions. The proposed method relies on dynamic load balancing to solve this problem, as after allocating tasks for the first time, the nodes begin to exchange information among themselves periodically, and each node maintains information about all network nodes, so if a particular node is under high load conditions, it will search within the stored information. It has to look up another node in the network with a low load and migrate some tasks to it. The problem with this method is that if the periodic time for information exchange is small, this will lead to a large load on the links and in turn will lead to a delay in the process of migrating tasks from one node to another, as well as increasing migrations leading to a delay in executing the tasks that are migrated and thus an increase in response time and total execution time.

In [12], a method was utilized for load adjusting between virtual machines inside a cloud center to diminish RT. The suggested approach architecture is based on three phases: firstly, the processing capacity of the virtual machines and the workload on each of them are assessed and categorized into four levels: Underload, Balance, High Balance, and Overload. Next, the evaluated execution time for the task is calculated for each virtual machine in the Underload condition. The objective is to assign the errand to the virtual machine that achieves the shortest possible execution time. If there are no virtual machines in the Underload state, the estimated execution time for the assignment is calculated based on the virtual machines still in the Balance state. Given the assumption that all virtual machines are in the Overload state, the job is placed in a queue until one of the virtual machines transitions to a different state. This approach fails to consider boundary conditions, such as internal and external stockpiling, which could fail in specific tasks, assuming that the memory is limited and insufficient to complete the assignment.

In article [13], an improved algorithm for the Throttled algorithm was suggested, in which the load is balanced by updating the Index Table, which contains information about the state of virtual machines, either 0, i.e. available, or 1, i.e. unavailable. Therefore, when a new task arrives at the cloud data center, a search is performed. Find the first available virtual machine within the index table and assign the task to it. The improved algorithm is named Throttled Modified Algorithm (TMA) to improve response time. The load in this algorithm is balanced by updating and maintaining two Index Tables, the first containing the ID of available virtual machines and the second containing the ID of unavailable virtual machines. When the cloud data center controller receives a new request, it sends a query to the load balancer for a new assignment. The load balancer selects the first available virtual machine and sends its ID to the controller, which assigns the task to the specified virtual machine. If there is no available virtual machine, the load balancer sends the value (-1) to the controller, which queues the task. This algorithm does not consider the resources of each VM and the load on it, and this may lead to the allocation of large tasks to virtual machines with limited resources, which leads to the occurrence of Trashing or increasing in RT.

In [14] A round-robin (RR) method was employed to minimize the reaction time (RT). The proposed approach relies on gradually distributing the execution period of projects for each cycle. The time allocation for the initial cycle is equal to the mean anticipated completion durations for the tasks. During the following cycle, the completed tasks are removed from the list, and the remaining execution times for the unfinished projects are determined at the halfway point. This cycle is rehased until all undertakings in the assignment list have been finished. This study depended on advancing the booking of errands allotted to virtual machines and the most common way of allocating assignments to virtual machines on the static Cooperative calculation. Consequently, this algorithm further develops handling time for little estimated errands. Yet, it doesn't consider the assets distributed to each virtual machine while designating undertakings, so it might prompt destruction or a critical expansion in handling time for enormous measured undertakings.

A QoS-aware service allocation method was proposed in [15] for fog ecosystems to reduce service latency while considering capacity limitations. The objective is a multi-dimensional knapsack problem to simultaneously minimize the cumulative delay in service execution and the excessive load on edge nodes, measured in processing capacity and energy consumption. This paper introduces a two-step resource management strategy that optimizes the response time for service delivery by minimizing the number of edge nodes used. Initially, a home edge and a group of backup edge nodes are selected for each device. They aim to identify the edge nodes to minimize the latency observed between them and the device. Following this, services requested by IoT are hosted on the designated edge nodes, ensuring the intended response time.

A review of resource management strategies applicable to cloud, fog, and edge computing was conducted in reference [16]. Firstly, it focused on the constraints of research on resource management strategies in that particular field. Therefore, it categorizes the existing research contributions to facilitate the implementation of an evaluation framework. An important contribution is the comprehensive review and analysis of research publications focusing on resource management approaches. In conclusion, this review emphasizes the potential for implementing resource management strategies inside the cloud/fog/edge paradigm. The current study is in its nascent stage of development, and it is imperative to surmount obstacles.

The work aims to enhance the quality and effectiveness of cloud computing by creating a systematic approach for allocating workloads among the processing nodes. Implementing this would facilitate a consistent and equitable workload in the cloud, enhance processing efficiency, and hence enhance the speed of response. The examined papers in this study mainly address load balancing and resource management in cloud computing. The strengths and limitations of these works are presented in Table 1.

Table 1. A comparison of the related works

Paper	Approach	Advantages	Limitations
[10]	Utilized PSO algorithm to assign tasks to VMs based on execution time	Considers execution time to optimize task assignment	Can converge to local optima if search space is small – Does not consider optimal solution within the entire cloud center
[11]	Used Ant Colony Optimization (ACO) algorithm for dynamic load balancing	Dynamically balances load by migrating tasks between nodes	Periodic task migration can cause network overhead and delay task execution
[12]	Classified VMs into Underload, Balanced, High Balanced, and Overload states Assigned tasks to VMs with lowest execution time	Considers VM processing capabilities to optimize task assignment	Does not consider other factors like storage, which can lead to thrashing
[13]	Improved Throttled algorithm by maintaining two index tables for available and unavailable VMs	Efficient task assignment by tracking available VMs	Does not consider VM resource utilization, which can lead to thrashing or increased response time
[14]	Used Round Robin algorithm with dynamically adjusted time slots	Enhanced efficiency in handling minor computing workloads	Fails to account for virtual machine resource allocation, which can impact the execution of large tasks
[15]	Modeled service allocation as a multi-dimensional knapsack problem to minimize latency and overloaded edge nodes	Considers both latency and edge node capacity constraints	Early-stage research, needs further development
[16]	Reviewed resource management techniques for cloud, fog, and edge computing	Provides a classification and evaluation framework for resource management research	Yet at the nascent stage of development, with obstacles to sur

2. COMPREHENSIVE THEORETICAL BASIS AND THE PROPOSED METHOD

The network will be represented as a diagram $G = (V; E)$ [17]. Where: $V = (C \cup F)$, The F: group consists of nodes situated at the periphery of the network, whereas the C: group comprises nodes positioned at the core of the cloud. E is the edge connecting all nodes, and $a_i \in A$: is a group that incorporates the Internet of Things (IoT) services.

The concept under consideration is founded upon an edge computing hierarchy and is structured in a three-level design. The first tier denotes the periphery of the network, encompassing a cluster of nodes near the geographical locations of Internet of Things devices, which are interconnected with them via a Local Area Network (LAN) [18],[19]. The second level contains nodes that are farther from the user than the first-level nodes and closer to the user than the nodes of the third level [20].

The third level includes the nodes furthest from the user. These nodes are located within the cloud center and have high specifications in terms of the hardware resources contained within them. Figure 1 represents the proposed network architecture. Load balancing means efficiently distributing incoming network traffic, application processes, or computing workloads across multiple servers, resources, or nodes within a cloud infrastructure. The primary objective is to optimize performance, reliability, and resource utilization by preventing any single server or resource from becoming overwhelmed, thereby ensuring a consistent and responsive user experience [21]-[22]. The assets designated to each node are distinguished and changed powerfully, relying upon the assets booked by the errands doled out to the nodes and the assets delivered when the node executes a specific task. The ability of every node is registered to rely upon the resources available to every node, as in Eqs. (1)-(4) [8].

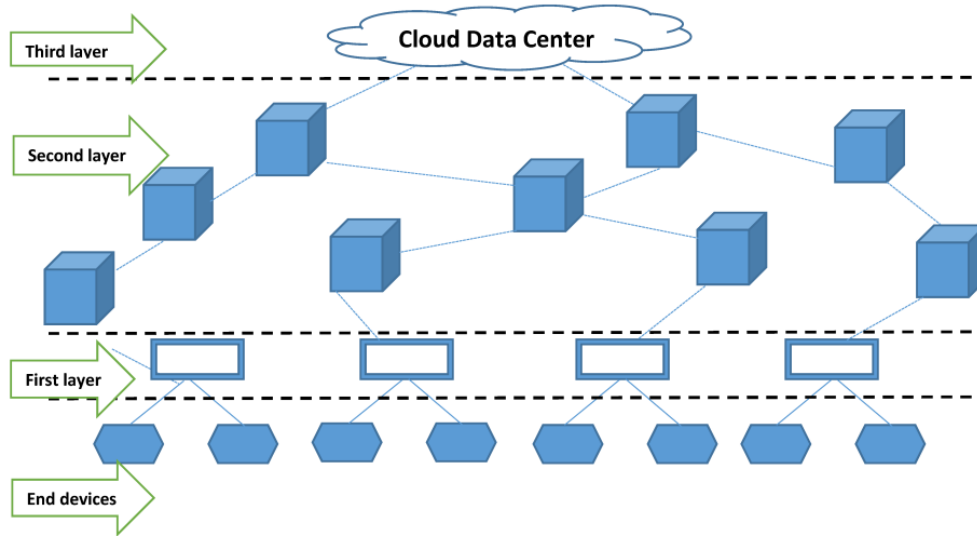


Figure 1. The proposed network three-level architecture

Where τ_j is the node's capability, P_{CPU} is the power of processing that is available to the node, m_i is the internal storage available of the node, $m_{i_{Max}}$ is the total internal storage available of the node, P_{Max} is the total power of processing of the node, $m_{e_{Max}}$ represents the overall external storage capacity of the node, whereas ϕ is a parameter used to adjust the resource's level of impact.

$$\tau_{CPU} = \frac{P_{CPU}}{P_{Max}} \times 100\% \quad (1)$$

$$\tau_{mi} = \frac{m_i}{m_{i_{Max}}} \times 100\% \quad (2)$$

$$\tau_{me} = \frac{m_{e_{Max}}}{m_{e_{Max}}} \times 100\% \quad (3)$$

$$\tau_j = (\phi_1 \times \tau_{CPU}) + (\phi_2 \times \tau_{mi}) + (\phi_3 \times \tau_{me}), \sum \phi = 1 \quad (4)$$

The ability of every node is changing continually, relying upon various factors. The node's capability τ_j decreases when allocating a new task to the node and the decreasing value $(1 - \mu)$ relies upon the consumed resources ratio of the node [11], [23] as in Eq. (5). Let μ be a coefficient that quantifies the ratio of the resources used to the total resources.

$$\tau_j(t+1) = (1 - \mu) \times \tau_j(t) \quad (5)$$

The node's capability increases upon finishing the execution of a task. The increasing value $(\nu+1)$ relies upon the ratio of the released resource [11], [23] as in Eq.(6). Where ν is a coefficient for characterizing the proportion of the delivered resources to the general resources.

$$\tau_j(t+1) = (\nu + 1) \times \tau_j(t) \quad (6)$$

Response time (RT) and estimated task execution time (ET) are essential metrics in computing and performance analysis, particularly in software applications, systems, and cloud computing. Let's define each term [24]. Response time, also known as latency, is the duration between initiating a request or task and completing the corresponding action or delivery of the response. It encompasses the time the system takes to process the request, perform necessary computations, and return the result to the requester.

A lower response time is generally desired, indicating a more responsive and efficient system, providing a better user experience. Estimated task execution time is a prediction or approximation of the time it will take for a specific task, job, or computation to complete. It's based on factors such as the complexity of the task, the capabilities and resources available to execute the task, historical performance data, and potentially other influencing parameters. Accurately estimating task execution time is crucial for resource allocation, scheduling, and load balancing to ensure efficient utilization of available computing resources.

$$ET = \frac{TL}{\text{Capacity} \times \text{Cores}(T)} \quad (7)$$

The main objective of our approach is to reduce the load variance among the processing nodes, aiming for a value close to zero. The variance, as applied in Equation (8), quantifies the dispersion of data around the mean value [12]. Let CL represent the current load on the f_j node, measured in terms of MIPS, and V represents the number of nodes. Variance quantifies the extent of dispersion within a certain dataset. The greater the dispersion of the data, the higher the variance around the mean [12]. Nevertheless, variance provides more precise information regarding variability compared to standard deviation and is employed for quantitative conclusions.

$$\text{Variance} = \frac{\sum_{j=1}^V (TL(f_j) - \text{Average}(TL))^2}{V} \quad (8)$$

3. Method

The IoT devices transmit requests to nodes located at the periphery of the network to select the specific node to allocate requests. Each node possesses knowledge of the requirements for the given work and the resources available to the neighboring nodes at the following level. Each node that receives requests participates in devising a strategy for allocating tasks and selecting the optimal configuration. The process of diagram production involves each node in the edge that gets a request generating a diagram to allocate tasks and calculate the anticipated total execution time in each outline, as shown in Equation (9).

$$ET_{Total} = \sum ET_{i,j} \quad (9)$$

Each node aims to find the scheme that achieves the least execution time, so if the closest end nodes are fit for executing the approaching errands, the tasks will be relegated to them to decrease deadline time infringement and lessen network traffic. The proposed method is explained as follows:

Input: {a: group that contain request services, n: group of nodes}

Output: { Δ : group of plans}

```

for (q=1 to  $\Delta$ ) do
Sort a in the order (Deadlinei - WTi from low to high;
h ← select neighboring nodes from n;
Sort h in term of proximity from low to high;
i,j ← 0;
while (a is not empty) do
Select ai from a and fj from h;
if (fj resources can hosted ai) then
assign ai to fj;
Update fj load according to (6);
elseif (the cloud node (ck) has enough capacity) then
assign ai to ck;
Update ck load according to (6);
end if;
Remove ai from a;
i++, j++;
end while;
calculate ETTotal according to (9);
calculate variance according to (8);
end for
Sort  $\Delta$  in the order of ETTotal from low to high;
Return  $\Delta$ ;

```

To create a single assignment schema, each edge node organizes incoming tasks based on the difference between the deadline and waiting time, arranging them from lowest to highest. Subsequently, the node randomly selects a group of neighboring nodes on the second level capable of receiving the task. These chosen second-level nodes are then ordered by distance from nearest to farthest. The node proceeds to assign the tasks sorted in the initial step one at a time, calculating the total estimated execution time and variance for each assignment, ultimately generating the schema. Finally, the resulting schemas are shared with all edge nodes to determine the optimal task allocation through optimization. Each edge node generates multiple proposals through the scheme selection process, each providing a comprehensive estimate of the overall implementation time and variance for that particular scheme. Subsequently, all edge nodes work together to ascertain the most advantageous plan among the feasible alternatives. The system administrator defines the iteration completion requirements and specifies a predetermined number of iterations. After completing all iterations, each agent distributes incoming tasks according to the selected scheme among all agents, assigning them to nodes located on the second level or the cloud center.

4. RESULTS AND DISCUSSION

In this study, an alternative network structure is crafted using two graphical representations: Barabasi Albert (BA) [18] and Erdos Renyi (ER) [20]. The network diagrams for these models concerning networks comprising (1000, 800, 600, 400, and 200) nodes are depicted in Figure 2.

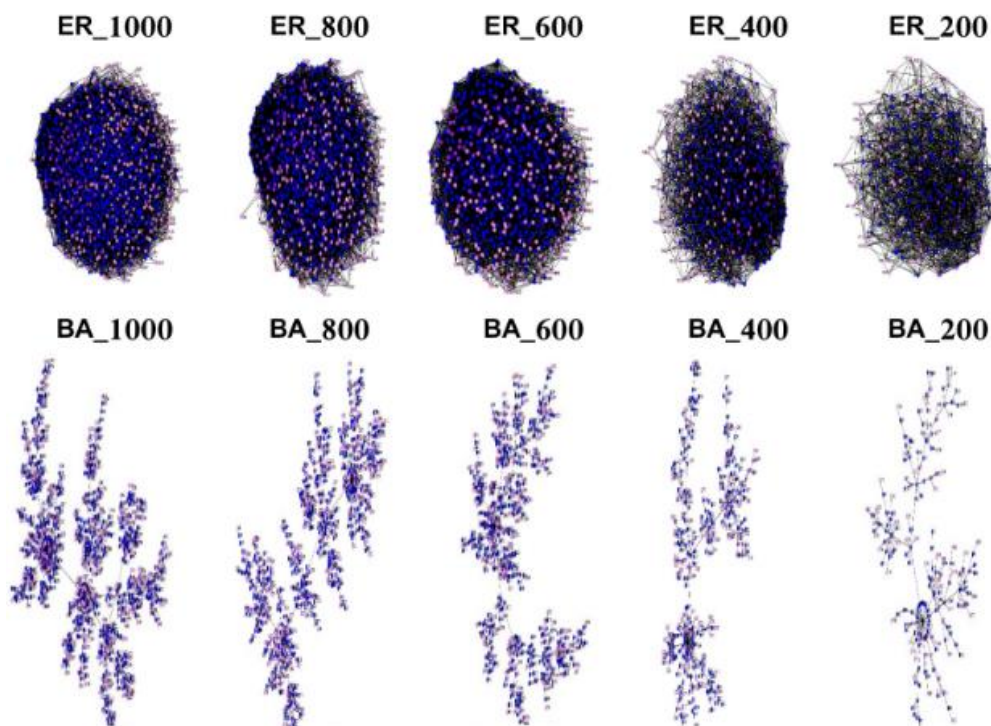


Figure 2. An analysis of network diagrams for two visual models, Barabasi Albert (BA) and Erdos Renyi (ER), in a network consisting of 1000, 800, 600, 400, and 200 nodes

Using Java within NetBeans, the simulation emulated a network of nodes from the edge to the cloud. The Java GraphicStream library facilitated the preceding graphic modeling. The workload input was derived from the Google Cluster Trace dataset [25]. Each agent was assigned 20 plans. The assessment was conducted over five intervals, employing the ER_1000, BA_1000, and BS_1000 topologies, with results extracted for each period. An evaluation compared the First Fit (FF) model [26] with a cloud center that did not include an edge component. The latter is dependent on reducing the number of transition delays between nodes. Every individual node tracks the transition delays, measured in hops, between itself and other nodes, generating a priority list among the adjacent nodes. Assigning priority to tasks is contingent upon the nodes having adequate resources to carry out the task. For both the suggested method and the FF model, the assessment involves determining the resource utilization ratio $(1 - \tau_j)$ for each node and then calculating the variance to achieve a variance as near to zero as feasible.

Table 2 presents the resource consumption variance (load) difference at various phases between the proposed model, the FF model, and the cloud center without an edge.

Table 2. The variance of the resources consumed (load) of each model

	FF model	Proposed model	Cloud model
ER_1000_P1	0.1521	0.005	0.2043
ER_1000_P2	0.1764	0.0519	0.2097
ER_1000_P3	0.0961	0.0529	0.1004
ER_1000_P4	0.0529	0.0510	0.0547
ER_1000_P5	0.051	0.0316	0.0345
BA_1000_P1	0.1576	0.0501	0.2043
BA_1000_P2	0.1722	0.024	0.2016
BA_1000_P3	0.1011	0.07	0.1017
BA_1000_P4	0.058	0.0723	0.059
BA_1000_P5	0.0625	0.0712	0.0691

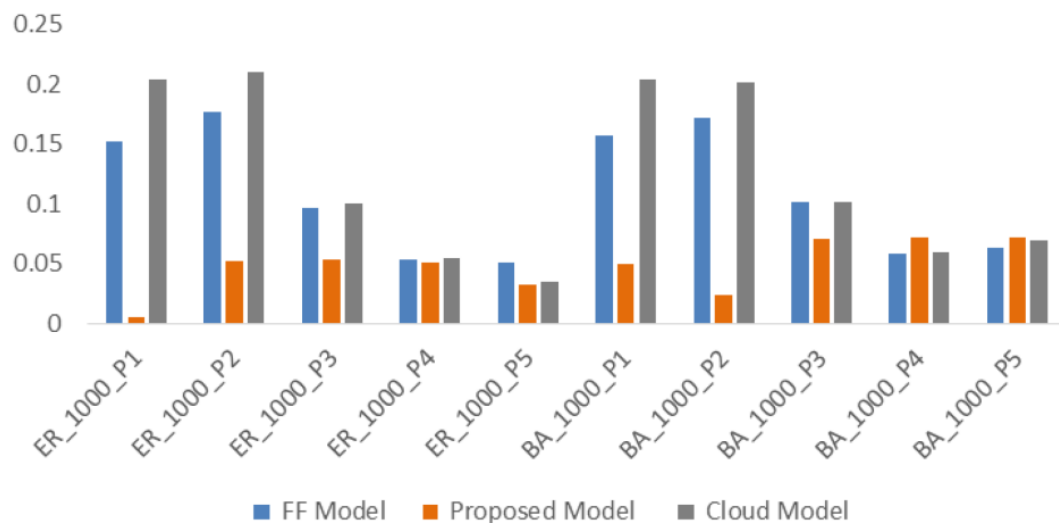


Figure 3 illustrates the differences in resource consumption (load) variations at different stages between the proposed model, the FF model, and a cloud center lacking an edge component.

The results demonstrate that the proposed model is superior than the FF model. The proposed model demonstrated a lower variance in load among nodes compared to the FF model, showcasing optimal utilization of available resources. This optimal resource utilization helps reduce costs by efficiently leveraging existing resources and minimizing the need for costly additional equipment. The proposed model achieves this by effectively balancing the load, ensuring fair and intelligent distribution of tasks based on task size, requirements, deadlines, and contractual resources, ultimately enhancing response times. Additionally, the proposed model mitigates overloaded and underloaded nodes, thus averting potential bottlenecks associated with high-load nodes.

Also, compared to the used topologies, the proposed model performs more efficiently if the ER topology is used. This is because the ER topology is somewhat random and represents large networks, while the WS topology is designed to create random networks. The goal was to simulate small world networks (inspired by sociology), or in general, networks that are embedded in some geometry and have a short-range basis and some long-range connections according to this geometry.

5. CONCLUSION

Effective allocation of resources in the emerging IoT infrastructure is crucial for addressing the challenges in cloud-based technologies while fulfilling a wide range of IoT management requirements. This work introduces an alternative approach to address load balancing in cloud computing within a diverse resource environment. The findings demonstrated that the suggested methodology yields a successful and optimal load, surpassing the performance of all the FF and cloud models. This paper concentrates on how optimizing the IoT administration position presents IoT benefits and gets a fair load dispersion while using resources on the network's edge. The proposed model presents a method to generate a local plan, and the plane selection is collaborative. The fair distribution of the resource increases the robustness of the system.

ACKNOWLEDGEMENTS












The authors would like to express their sincere gratitude to Diyala University for providing the necessary infrastructure and laboratory facilities to conduct the research presented in this paper.

REFERENCES

- [1] Fatima Zahra Fagroud, et al. "Internet of Things Search Engines: Toward a General Architecture." *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 28, no. 2, 1 Nov. 2022, pp. 1117–1117, <https://doi.org/10.11591/ijeecs.v28.i2.pp1117-1127>. Accessed 23 May 2024.
- [2] Atiqur, Rahman, Guangfu Wu, and Ali Md Liton. "Mobile edge computing for internet of things (IoT): security and privacy issues." *Indonesian Journal of Electrical Engineering and Computer Science (IJECCS)* 18.3 (2020): 1486-1493, DOI:10.11591/ijeecs.v18.i3.pp1486-1493.
- [3] Buyya, Rajkumar, and Satish Narayana Srirama, eds. *Fog and edge computing: principles and paradigms*. JohnWiley & Sons, 2019.
- [4] Sehrawat, Deepti, and Nasib Singh Gill. "Smart sensors: Analysis of different types of IoT sensors." 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). IEEE, 2019, DOI: 10.1109/ICOEI.2019.8862778.
- [5] Hazra, Abhishek, et al. "Fog computing for next-generation internet of things: fundamental, state-of-the-art and research challenges." *Computer Science Review* 48 (2023): 100549, <https://doi.org/10.1016/j.cosrev.2023.100549>.
- [6] Laroui, Mohammed, et al. "Edge and fog computing for IoT: A survey on current research activities & future directions." *Computer Communications* 180 (2021): 210-231, <https://doi.org/10.1016/j.comcom.2021.09.003>.
- [7] Mahadevappa P., Murugesan R.K. A data quarantine model to secure data in edge computing. *International Journal of Electrical and Computer Engineering* 2022; 12(3): 3309-3319, <https://doi.org/10.48550/arXiv.2111.07672>.
- [8] Kashani, Mostafa Haghi, and Ebrahim Mahdipour. "Load balancing algorithms in fog computing." *IEEE Transactions on Services Computing* 16.2 (2022): 1505-1521, DOI: 10.1109/TSC.2022.3174475.
- [9] Costa, Breno, et al. "Orchestration in fog computing: A comprehensive survey." *ACM Computing Surveys (CSUR)* 55.2 (2022):1-34, <https://doi.org/10.1145/3486221>.
- [10] Chalack, Vahid Asadzadeh, S. N. Razavi, and S. J. Gudakahriz. "Resource allocation in cloud environment using approaches based particle swarm optimization." *International Journal of Computer Applications Technology and Research* 6.2 (2017): 87-90.
- [11] Gao, Ren, and Juebo Wu. "Dynamic load balancing strategy for cloud computing with ant colony optimization." *Future Internet* 7.4 (2015): 465-483, <https://doi.org/10.3390/fi7040465>.
- [12] Dhari, Atyaf, and Khaldun I. Arif. "An efficient load balancing scheme for cloud computing." *Indian Journal of Science and Technology* 10.11 (2017): 1-8.
- [13] Phi, Nguyen Xuan, et al. "Proposed load balancing algorithm to reduce response time and processing time on cloud computing." *Int. J. Comput. Netw. Commun* 10.3 (2018): 87-98, DOI : 10.5121/ijcnc.2018.10307.
- [14] Stephen, A., BJ Hubert Shanthan, and Daks Ravindran. "Enhanced round Robin algorithm for cloud computing." *Int J Sci Res Comput Sci Appl Manag Stud* 7.4 (2018): 1-5.
- [15] Fadahunsi, Olamilekan, and Muthucumar Maheswaran. "Locality sensitive request distribution for fog and cloud servers." *Service Oriented Computing and Applications* 13 (2019): 127-140, <https://doi.org/10.1007/s11761-019-00260-2>.
- [16] Mijuskovic, Adriana, et al. "Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification." *Sensors* 21.5 (2021): 1832, <https://doi.org/10.3390/s21051832>.
- [17] Leus, Geert, et al. "Graph Signal Processing: History, development, impact, and outlook." *IEEE Signal Processing Magazine* 40.4 (2023): 49-60, DOI: 10.1109/MSP.2023.3262906.
- [18] Huang, Haiping. *Statistical mechanics of neural networks*. Singapore: Springer, 2021.
- [19] Xia, Yongxiang, Jin Fan, and David Hill. "Cascading failure in Watts–Strogatz small-world networks." *Physica A: Statistical Mechanics and its Applications* 389.6 (2010): 1281-1285, <https://doi.org/10.1016/j.physa.2009.11.037>.
- [20] Martínez-Martínez, C. T., et al. "Computational and analytical studies of the Randić index in Erdős–Rényi models." *Applied Mathematics and Computation* 377 (2020): 125137, <https://doi.org/10.1016/j.amc.2020.125137>.
- [21] Raghava, N. S., and Deepti Singh. "Comparative study on load balancing techniques in cloud computing." *Open journal of mobile computing and cloud computing* 1.1 (2014): 18-25.
- [22] Shafiq, Dalia Abdulkareem, N. Z. Jhanjhi, and Azween Abdullah. "Load balancing techniques in cloud computing environment: A review." *Journal of King Saud University-Computer and Information Sciences* 34.7 (2022): 3910-3933, <https://doi.org/10.1016/j.jksuci.2021.02.007>.
- [23] Liu, Chang, et al. "Solving the multi-objective problem of IoT service placement in fog computing using cuckoo search algorithm." *Neural Processing Letters* 54.3 (2022): 1823-1854, <https://doi.org/10.1007/s11063-021-10708-2>.
- [24] Shafiq, Dalia Abdulkareem, et al. "A load balancing algorithm for the data centres to optimize cloud computing applications." *IEEE Access* 9 (2021): 41731-41744, DOI: 10.1109/ACCESS.2021.3065308.

- [25] Alam, Mansaf, Kashish Ara Shakil, and Shuchi Sethi. "Analysis and clustering of workload in google cluster trace based on resource usage." 2016 IEEE Intl conference on computational science and engineering (CSE) and IEEE Intl conference on embedded and ubiquitous computing (EUC) and 15th Intl symposium on distributed computing and applications for business engineering (DCABES). IEEE, 2016, DOI: 10.1109/CSE-EUC-DCABES.2016.271.
- [26] D'osa, György, and Jirí Sgall. "First Fit bin packing: A tight analysis." 30th International symposium on theoretical aspects of computer science (STACS 2013). Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2013, <https://doi.org/10.4230/LIPIcs.STACS.2013.538>.

BIOGRAPHIES OF AUTHORS

	<p>Abdullah Farhan Mahdi: He holds a master's degree in 2012 from the University of Anbar, College of Computer Science and Information Technology, specializing in data warehouses. In 2023, he obtained a doctorate from the University of Anbar, College of Computer Science and Information Technology, specializing in artificial intelligence. He is a lecturer at the University of Diyala, College of Agriculture. He can be contacted at email: abdullahmahdi@uodiyala.edu.iq.</p> <p>Scopus®    </p>
	<p>Aymen Mudheher Badr: He obtained a Bachelor of Science in Computer Science in 2001. And an M.S. degree in computer science from Chongqing University, China, in 2015. He has worked as a lecturer at Diyala University since 2003 until now. He holds a PhD in Computer Science - Medical Informatics from Sfax University, Digital Research Center of Sfax (CRNS) Laboratory of Signals, Systems, Artificial Intelligence and Networks (SM@RTS), Tunisia, 2024. He can be contacted at email: aymen.m.badr@uodiyala.edu.iq.</p> <p>Scopus®    </p>
	<p>Israa Mishkhal was born in Iraq, in Baqubah. She obtained a bachelor's degree in computer science from Diyala University. She holds a master's degree in computer science from Ball State University (BSU) in the United States of America. She is a Ph.D. student at Universiti Sains Malaysia, Computer Science. She is a lecturer at College Science/Diyala University, Iraq (Diyala). She has many research papers in national and international conferences. email: israaadnan@uodiyala.edu.iq, israa_adnan85@student.usm.my</p>