Journal of Positive Sciences (JPS) **eISSN: 2582-9351, https://positive-sciences.com** **JPS. 2023 July; 3(4): 1-7**

Article info **Research Article**
Received on: 02.07.2023
Accepted on: 28.07.2023
Published on: 31.07.2023
doi: https://doi.org/10.52688/ASP16586

Development fault tolerance for healthcare system on fog computing

Hadeel T. Rajab [1,*] , Manal Fadhil Younis [2]

[1] Department of Computer Engineering, University of Baghdad, Baghdad, Iraq

[*] h.tareq1205@coeng.uobaghdad.edu.iq

ABSTRACT

Due to the revolution in the Internet of Things (IoT) devices and the huge amount of messages sent to cloud servers, Fog Computing (FC) was used. Data in Motion (DM) technique was employed on FC to enhance cloud storage capacity by selecting essential messages to cloud while erasing the rest messages from FC. Dynamic Weighted Round Robin (DWRR) algorithm was used to distribute load among fog servers to increase throughput. Dynamic checkpoint interval was used on FC to provide efficient Fault Tolerance (FT) for Primary-Backup Replication (PBR) technique. Backup fog server performs DM when primary fog server stops working. This contributes to minimizing the effort on backup server by reducing the makespan of the DM and enhancing the storage capacity of backup server to about double, which in turn leads to enhance backup server performance. Moreover, proposed system contributes to advance backup server technique by providing two load balancers to prevent a Single Point of Failure (SPOF).

Keywords: Fault Tolerance, Data Replication, Checkpointing, Reliability, Fog Computing, SPOF

INTRODUCTION

The Internet of Things (IoT) is a popular technology in the field of wireless telecommunications that was invented by Kevin Ashton in 1999 to create a connection between the physical and digital world via the Internet. IoT technology is physical objects that refer to 'things' linked to the Internet like wireless sensors, actuators, and Microcontrollers (MCUs), which enable these things to collect and exchange data through the Internet connection [1]. IoT devices in general are connected to the cloud in order to collect and process data [2]. Cloud Computing (CC) is a ubiquitous technology that is the easiest way to gather and process data generated from IoT (edge) devices by connecting these devices to cloud servers. CC provides services through the Internet to share various software and hardware resources (i.e., storage, network, servers, and applications) with the user as on-demand [3]. According to Cisco, humans will connect more than 50 billion IoT (edge) devices to the Internet by 2020. Moreover, researchers conclude that the number of IoT devices will be arrived at 1 trillion by 2025 [3, 4]. A huge amount of data will be produced due to the increased use of IoT devices located nearby users. Thus, several challenges appear in IoT systems such as network congestion, higher latency, and data loss [5]. To limit the obstacles of IoT systems with CC, a computing paradigm named Fog Computing (FC) was used [6]. FC represents an intermediate layer existing between IoT (edge) devices and CC proposed by Cisco in 2012. It presents temporary storage, processing, and computing so that FC is a bit like a CC [7]. Therefore, the creation of FC led to the formation of three layers (IoT devices, fog, and cloud layer) of IoT applications to enhance the Quality of Service (QoS) as shown in Figure (1) [8].

***Corresponding author**
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

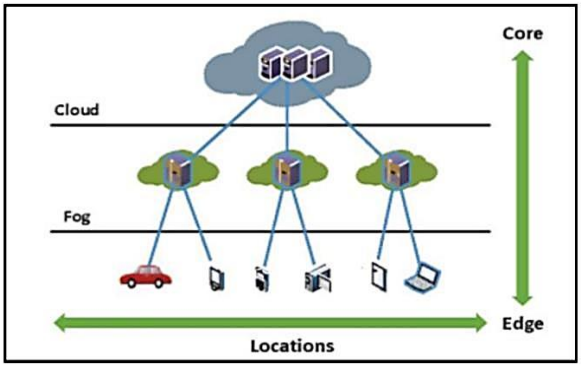

Figure 1: Architecture of FC [8].

A server may run slowly during data processing because of the huge amount of data. As a result, the Load Balancing (LB) technique was used. LB technique is a procedure to distribute the load over various servers in the CC [9]. Although the use of LB techniques, there is not much awareness and alertness to Fault Tolerance (FT) in recent times [10]. FT means the operation of executing system tasks continues even when a fault occurs [11]. Primary-Backup Replication (PBR) is a more efficient technique to provide FT compared to other types of replication because it avoids increasing traffic in-network and server overwork and improves data availability [12].

RELATED WORK

There are many researches that refer to providing FT, some of them are discussed as follows:

Rajab et al. [12] proposed to improve the FT technique on FC. As well as to increase the throughput of fog servers by DWRR algorithm and enhance the capacity of the cloud storage by using the DM technique that refers to selecting only minimum, maximum, and average values of pulse sensor messages to cloud storage. This proposed system performs a dynamic checkpoint interval on the PBR technique by reading the status of the CPU so that when the CPU value is 0.2 or less, the checkpoint process stops working and returns as soon as the CPU value is great than 0.2. The checkpoint process is recognized new messages by checking the message's time. However, this paper does not enhance Backup Fog Server (BFS) performance and also does not provide protection from SPOF that may occur on the load balancer.

Madsen et al. [13] proposed to improve the reliability of FC by performing checkpointing with the replication technique and selecting the optimal checkpointing interval to replicate new data. However, this system does not prevent bottlenecks occur on the primary server because it does not use a proxy to forward data to another fog server when the primary server fails.

Al-Joboury et al. [14] proposed to enhance the storage capacity of the cloud by using DM technique on the fog server and also performing the least connection load balancing algorithm with the MQTT protocol to increase system throughput. However, this proposed system does not distribute messages depending on the current status of servers and does not provide an FT for data in fog servers.

THE ARCHITECTURE DESIGN OF THE PROPOSED SYSTEM

IoT healthcare systems are one of the most critical systems and require an efficient architecture to receive and process data, therefore, the fog layer has been added. Accordingly, the proposed system architecture of this paper consists of three layers: IoT sensors, fog, and cloud layer.

IOT SENSORS

This layer was represented by a pulse sensor to read the patient's heartbeat. This sensor connects to Arduino Dock 2, which in turn is connected with Omega 2+ in order to send the patient's heartbeat rate to the fog layer by Wi-Fi connection as shown in Figure (2). Omega2++ sends one message to FC every 15 seconds by MQTT protocol.

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

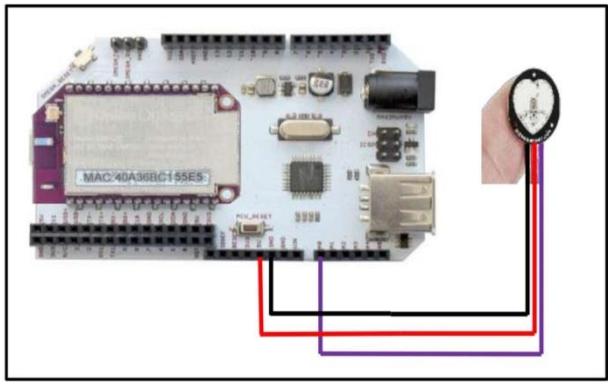

Figure 2: Pulse Sensor Connection with Arduino Dock 2

FOG LAYER

This layer is located between IoT and cloud layers. The fog layer was represented by a number of servers located within a particular building. On fog servers, the Data in Motion (DM) technique was used to minimize traffic on the cloud network and to store messages temporarily on the fog server. In this technique, messages live within the fog server for a specific time. So every 30 minutes, the messages inside fog servers have been filtered by selecting the minimum, average, and maximum values and then sent to the cloud. The rest of the messages are deleted after 1 minute from the filtering messages process in the fog server. Thus, cloud capacity for permanent storage has been enhanced because only three messages per patient will be sent to permanent cloud storage every 30 minutes instead of transmitting all messages.

Since a huge amount of messages were generated from the increased use of IoT-medical sensors, more than one fog server was used to distribute the workload among them. Dynamic Weighted Round Robin (DWRR) algorithm was used to distribute workload among fog servers based on the status of each server. The load balancer subscribes to the pulse sensor messages by Mosquitto broker and then distributes load among fog servers by the DWRR algorithm. DWRR algorithm uses the CPU and memory values of fog servers to distribute the load among them. Each fog server sends its CPU and memory to RedisDB located on the MQTT load balancer in order to sort these fog servers as shown in Figure-3. The sorting process is to compare the CPU and memory values of fog servers so that the best server is the fog server that has memory and CPU value higher than another fog server. So, the best fog server receives the higher weight of messages while another fog server receives the smallest weight of messages from the load balancer. Accordingly, the workload will be distributed across fog servers based on the ability of each fog server at that moment.

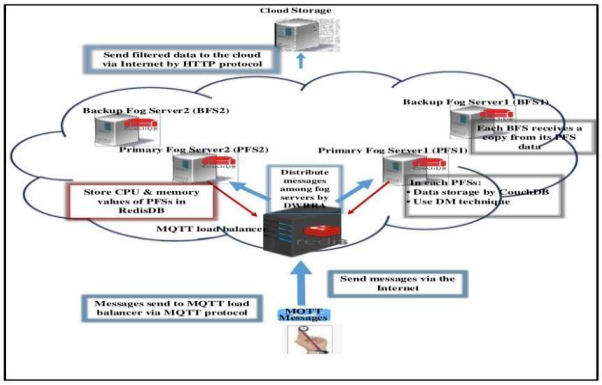

Figure 3: Proposed Healthcare System of How Distributed Data on Fog Servers.

Since pulse messages were stored in fog servers for a period of time, it was essential to increase the reliability and efficiency of FC. A fog server may stop working because a malfunction occurs in one of its physical components. Because data is stored in the fog servers for a limited period of time in order to filter it before sending it to the cloud for permanent storage, it is necessary to provide FT Technique to protect this critical data when failing fog servers.

Primary-Backup Replication (PBR) technique has been used to achieve FT in order to reduce overhead and also does not require waiting for failure repair to restart work. So if the server fails, another fog server can recover the work. To perform the PBR technique, two additional servers have been used. These additional fog servers are Backup Fog Servers (BFSs) for Primary Fog Servers (PFSs). The PFSs are the fog servers whose data is distributed on them by the DWRR algorithm from the load balancer as shown in Figure (3).

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

ON THE PRIMARY FOG SERVERS (PFSS)

A checkpoint has been performed in each PFS in order to avoid repeating sending all data that was previously sent to the BFS. The checkpoint process operates to distinguish data that are recently sent to PFSs in order to send only them to BFSs. The new data is distinguished by checking the time of the pulse (heartbeat) sensor messages. If the time of the last message in the checkpointing is less than the time of the incoming message, this means that the incoming message is new. As a result, only recent messages are sent to BFS. The repeated checkpoint processes every interval time without attention to the status of these PFSs may produce an additional effort that affects on DM process that must be sent the filtering messages every 30 minutes to the cloud storage. Thus, a dynamic checkpoint interval has been proposed on the primary servers. This dynamic interval is based on the CPU values of PFSs. From experiments, it has been observed that when the DM occurs with the checkpoint process for incoming data, the DM process time is greatly affected when the remaining CPU value is 0.2 or less on the PFSs. Accordingly, the dynamic checkpoint interval is stopped as soon as the remaining CPU value of PFSs is 0.2 or less. The checkpoint interval returns when the CPU value is greater than 0.2 [12]. Each PFS deletes the remaining data from its BFS after the DM process occurs in order to prevent the BFS from filtering data that has been previously filtered by the PFS.

BFSs have been employed to detect failure. So each BFS is responsible for monitoring its PFS. BFS1 is responsible for checking PFS1, and BFS2 is responsible for PFS2. Each backup server sends a request to its primary server every time period in order to receive a reply from its primary server. When a timeout period is finished without the PFS replying, it indicates that the PFS has stopped working. As a result, the BFS will recover the service automatically in two steps:

1- The BFS is filtering messages that were previously transmitted from its failed PFS. This BFS is using the DM immediately in order to reduce downtime during failures and minimize effort on BFSs.

2- The BFS replaces the IP address of its failure PFS with its IP address in RedisDB on the load balancer to receive messages from the MQTT load balancer based on the DWRR algorithm. BFS is sent its memory and CPU value to RedisDB after being connected to it as illustrated in Figure (4).

When PFS1 fails, BFS1 performs the DM process immediately, and at the same time, BFS1 is integrated with the DWRR algorithm. So that pulse sensor messages are distributed between PFS2 and BFS1 based on their current status. As soon as PFS1 replies to BFS1 and returned to work, PFS1 erases all messages stored on it in order to prevent messages filtering that were previously filtered by BFS1. BFS1 executes the DM process immediately for messages that have been obtained from the load balancer and replaces its IP address with the IP address of PFS1 in RedisDB. Thus, the MQTT load balancer resumes message distribution among PFS1 and PFS2. The same method is applied when PFS2 is stopped.

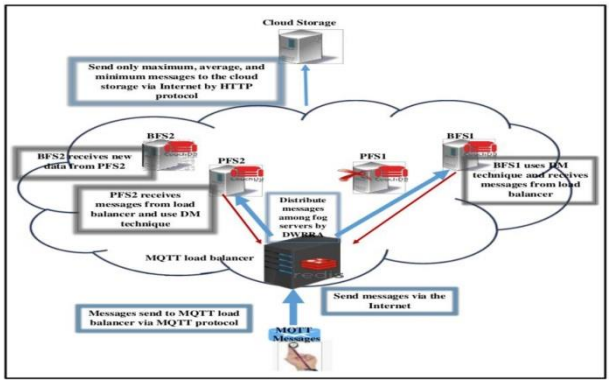

Figure 4: Performance BFS on FC.

To develop the FT technique within FC, it was necessary to prevent a bottleneck from occurring that leads to a Single Point of Failure (SPOF) for incoming pulse sensor messages. Therefore, another MQTT load balancer was employed. So Keepalived software and VRRP were executed for the primary and secondary MQTT load balancers. Keepalived software is written in C language and it operates to route messages. Keepalived and VRRP were performed by creating a single virtual IP address. A virtual IP address was shared between the two MQTT load balancers which are primary and secondary load balancers. In the beginning, messages are transmitted to the virtual IP address. Then this IP routes the pulse sensor messages to the active MQTT load balancer. When there is no failure occurs in each of the MQTT load balancers, the priority in sending pulse sensor messages will be to the primary MQTT load balancer as shown in Figure (5). So that messages are sent to the primary MQTT load balancer that distributes messages between PFSs depending on the DWRR algorithm. PFSs are sent their CPU and memory values to both primary and secondary load balancers.

If the primary load balancer fails, the virtual IP address will route pulse sensor messages dynamically to the secondary load balancer. A secondary MQTT load balancer works the same as a primary MQTT load balancer. So each of the RedisDB and Mosquitto brokers has been installed on the secondary MQTT load balancer. RedisDB employs receiving values of memory and

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

CPU of PFS1 and PFS2 to distribute messages between these PFSs based on the DWRR algorithm. The virtual IP address will route messages dynamically to the primary MQTT load balancer when this primary load balancer returns to its work. BFSs connect with two RedisDB (one on the primary and another on the secondary MQTT load balancer) by Node.js in order to resume receiving messages in case of the primary load balancer fails.

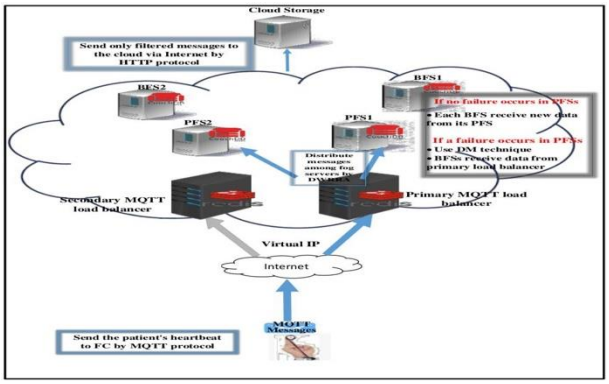

Figure 5: System Behavior to Prevent SPOF.

The cloud layer is represented by the IBM Cloudant DB instance that is used to permanently store for minimum, average, and maximum values of pulse messages that are sent from the fog servers by HTTP protocol.

EVALUATION AND DISCUSSION

It is necessary to provide a strong FT technique within FC. Therefore, it is very essential to maintain the performance of the BFSs and make them always ready to execute the same tasks of PFSs in the event of failure occur in the PFS. Moreover, prevention of SPOF from occurring to ensure protect pulse sensor messages within FC. Thus, this section contributes to minimizing the makespan of the DM process and improving BFS storage capacity. This occurs by filtering messages on the BFS as soon as its PFS is a failure instead of waiting a limited time (30 minutes) to perform the DM process on the BFS. Hence, Execution time was compared in the event of immediate filtering, and in the event of a wait for 30 minutes. The comparison was made with a number of sensors ranging from 20 to 80 as illustrated in Figure (6).

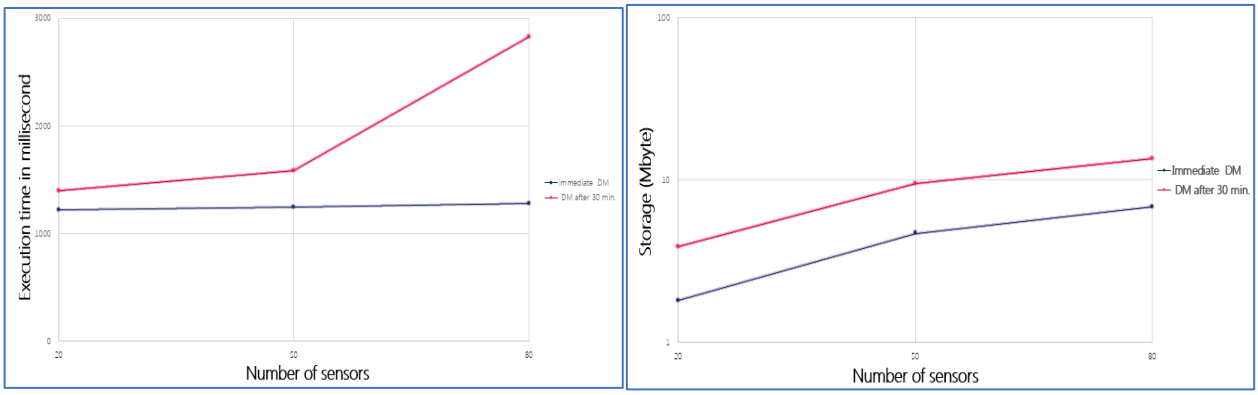

Figure 6: Execution Time of DM Technique in BFS.

To perform the comparison, the worst-case scenario was determined, which is the PFS stops working at 28 minutes. So this scenario depends on 1 message sent from the pulse sensor every 15 seconds. The immediate DM process occurs when PFS fails at 28 minutes and is then compared with the execution time of the DM process after 30 minutes from PFS failure which includes messages of suspended PFS through 28 minutes and messages coming from MQTT load balancer within 30 minutes. The results obtained in Figure -6 indicate that the execution time in immediate DM performing is less than the time of the DM process after 30 minutes.

The range of execution time in the immediate DM process and in the case of executing DM after 30 minutes from PFS failure are shown in Table -1.

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

Table 1: Comparison of DM Process Execution Time in BFS.

Number of sensors	Execution time of immediate DM	Execution time DM after 30 minutes
20	1220 ms	1393 ms
50	1247 ms	1582 ms
80	1275 ms	2824 ms

This improvement not only minimizes the effort and increases the performance of BFSs to receive pulse messages from the MQTT load balancer, but it also enhances the capacity storage of BFS as an illustrated result obtained in Figure (7). The consuming storage capacity of BFS when using the DM process after 30 minutes is approximately twice the consumption of storage capacity in the event of immediate DM.

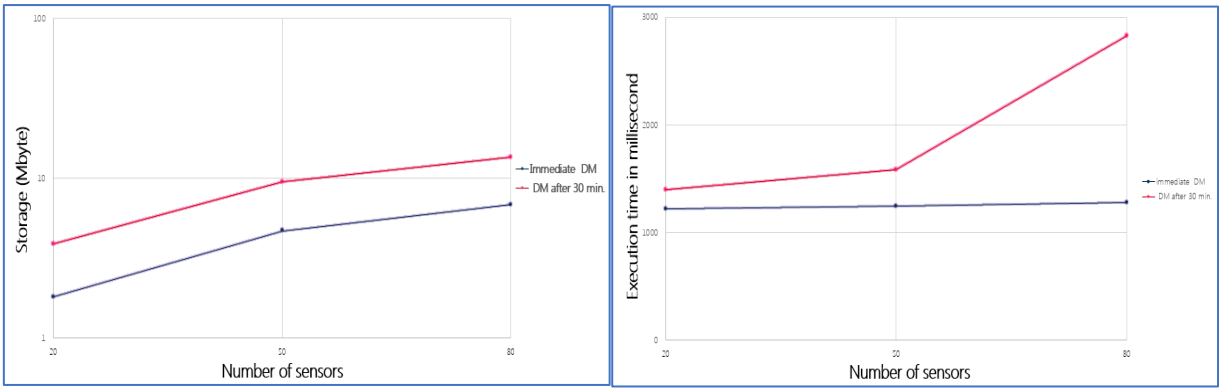

Figure 7: Storage Capacity Comparison.

Table-2 shows the number of messages transmitted from a number of sensors to the BFS every 15 seconds instead of losing these messages when the PFS stops working for 30 minutes.

Table 2: Range of Messages Sent from Sensors.

Number of sensors	Number of messages within 30 minutes
20	2400
50	6000
80	9600

Moreover, the system is more reliable and highly available by preventing SPOF from occurring in FC in order to ensure protect pulse sensor messages within FC.

CONCLUSION

The proposed system enhances the architecture of IoT healthcare systems. Integration between fog and cloud with DM leads to improving the storage capacity of the cloud, and using the DWRR algorithm participates to increase throughput on the fog servers. And provide efficient FT on FC by using dynamic checkpoint intervals for the PBR technique. Despite all these improvements, it was necessary to develop the FT technique more by enhancing the state of BFSs as much as possible to make them always ready to perform PFSs tasks in the event of any failure. Therefore, integration between the PBR technique on fog servers and the DWRR algorithm on MQTT load balancers has been implemented to maintain the pulse sensor messages directed from the load balancer in the event of PFS failure. Which makes the MQTT load balancer transmit messages to the BFS instead of sending messages to its failed PFS. Performing the DM technique on BFS once a failure occurs in PFS led to a reduced makespan of the DM process and also improve storage capacity to about double. Moreover, the reliability and availability of FC have been increased by adding another MQTT load balancer to prevent SPOF from occurring on FC.

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq

REFERENCES

[1] C. Stergiou and K.E. Psannis, "Recent Advances Delivered by Mobile Cloud Computing and Internet of Things for Big Data applications: a survey," International Journal of Network Management, vol. 27, pp.1-12, 2016.
[2] S. Balaji, k. Nathani and R. Santhakumar, "IoT Technology, Applications and Challenges: A Contemporary Survey," Wireless Personal Communications, Springer, vol.108, pp. 363–388, 2019.
[3] A. R. Suraj, S. J. Shekar and G. S. Mamatha, "A Robust Security Model for Cloud Computing Applications," International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), IEEE, pp. 018-022, 2018.
[4] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize Its Potential," in Computer, IEEE, vol. 49, pp. 112-116, Aug. 2016.
[5] B. Qin, J. Cai, Y. Luo, F. Zheng, J. Zhang and Q. Luo, "Research and Application of Intelligent Internet of Vehicles Model Based on Fog Computing," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), pp. 1777-1783, 2019.
[6] R. Mahmud, F. L. Koch and R. Buyya, "Cloud-Fog Interoperability in IoT-enabled Healthcare Solutions," Proceedings of the 19th International Conference on Distributed Computing and Networking, vol. 32, pp. 1–10, 2018.
[7] R. Mahmud, R. Kotagiri and R. Buyya, "Internet of Everything," Springer, pp. 103-130, 2018.
[8] M. Al Yami and D. Schaefer, "Fog Computing as a Complementary Approach to Cloud Computing," 2019 International Conference on Computer and Information Sciences (ICCIS), IEEE, pp. 1-5, 2019.
[9] A. Hota, S. Mohapatra and S. Mohanty, "Computational Intelligence in Data Mining," Springer, vol. 711, pp. 99-110, 2019.
[10] [10] S. M. Abdulhamid, M. S. Abd Latiff, S. H. H. Madni and M. Abdullahi, "Fault Tolerance Aware Scheduling Technique for Cloud Computing Environment Using Dynamic Clustering Algorithm," Neural Computing and Applications, Springer, vol. 29, pp. 279–293, 2018.
[l1] [11] R. Gupta, R. Kamal, U. Suman, "A QoS-Supported Approach Using Fault Detection and Tolerance for Achieving Reliability in Dynamic Orchestration of Web Services," International Journal of Information Technology, Springer, vol. 10, pp. 71–81, 2017.
[12] [12] H. T. Rajab, M. F. Younis, "Dynamic Fault Tolerance Aware Scheduling for Healthcare System on Fog Computing," Iraqi Journal of Science, vol. 62, pp. 308-318, 2020.
[13] [13] H. Madsen, B. Burtschy, G. Albeanu and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable Fog computing," 2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP), IEEE, pp. 43-46, 2013.

*Corresponding author
Hadeel T. Rajab,
Department of Computer Engineering, University of Baghdad, Baghdad, Iraq
e-mail: h.tareq1205@coeng.uobaghdad.edu.iq