Iraqi Journal of Science

ISSN: 0067-2904

Dynamic Fault Tolerance Aware Scheduling for Healthcare System on Fog Computing

Hadeel T. Rajab*, Manal F. Younis
Department of Computer, College of Engineer, University of Baghdad, Baghdad, Iraq

Received: 4/12/2020 Accepted: 15/3/2020

Abstract

Internet of Things (IoT) contributes to improve the quality of life as it supports many applications, especially healthcare systems. Data generated from IoT devices is sent to the Cloud Computing (CC) for processing and storage, despite the latency caused by the distance. Because of the revolution in IoT devices, data sent to CC has been increasing. As a result, another problem added to the latency was increasing congestion on the cloud network. Fog Computing (FC) was used to solve these problems because of its proximity to IoT devices, while filtering data is sent to the CC. FC is a middle layer located between IoT devices and the CC layer. Due to the massive data generated by IoT devices on FC, Dynamic Weighted Round Robin (DWRR) algorithm was used, which represents a load balancing (LB) algorithm that is applied to schedule and distributes data among fog servers by reading CPU and memory values of these servers in order to improve system performance. The results proved that DWRR algorithm provides high throughput which reaches 3290 req/sec at 919 users. A lot of research is concerned with distribution of workload by using LB techniques without paying much attention to Fault Tolerance (FT), which implies that the system continues to operate even when fault occurs. Therefore, we proposed a replication FT technique called primary-backup replication based on dynamic checkpoint interval on FC. Checkpoint was used to replicate new data from a primary server to a backup server dynamically by monitoring CPU values of primary fog server, so that checkpoint occurs only when the CPU value is larger than 0.2 to reduce overhead. The results showed that the execution time of data filtering process on the FC with a dynamic checkpoint is less than the time spent in the case of the static checkpoint that is independent on the CPU status.

Keywords: Fault tolerance, Data replication, Checkpointing, Reliability, Fog computing, Task scheduling.

تسامح الخطأ الديناميكي مع الجدولة الديناميكية لنظام الرعاية الصحية على الحوسبة الضبابية

هديل طارق رجب*، منال فاضل يونس
قسم الحاسبات، كلية الهندسة، جامعة بغداد، بغداد، العراق.

الخلاصة

يساهم انترنيت الاشياء (IoT)، في تحسين نوعية الحياة حيث يدعم العديد من التطبيقات وخاصة أنظمة الرعاية الصحية. يتم إرسال البيانات الناتجة من أجهزة إنترنت الأشياء إلى الحوسبة السحابية (CC) للمعالجة والتخزين على الرغم من الكمون الناتج عن المسافة. بسبب الثورة في أجهزة إنترنت الأشياء ، زادت البيانات المرسلة إلى CC. نتيجة لذلك ، هناك مشكلة أخرى تضاف إلى زمن الانتقال وهي زيادة الازدحام على الشبكة

*Email: h.tareq1205@coeng.uobaghdad.edu.iq

السحابية. تم استخدام حوسبة الضباب (FC) لحل هذه المشاكل نظرًا لقربها من أجهزة إنترنت الأشياء ، وتصفية البيانات المرسلة إلى CC. FC هي طبقة متوسطة تقع بين أجهزة إنترنت الأشياء وطبقة CC. نظرًا لاستقبال FC للبيانات المرسلة من أجهزة IoT قبل CC ، تم استخدام خوارزمية Dynamic Weighted Round Robin (DWRR) والتي تمثل خوارزمية موازنة التحميل (LB) التي تستخدم لجدولة وتوزيع البيانات بين خوادم الضباب عن طريق قراءة قيم وحدة المعالجة المركزية والذاكرة الخاصة بهذه الخوادم من أجل تحسين أداء النظام. أثبتت النتائج أن خوارزمية DWRR توفر إنتاجية عالية تصل إلى / req 3290sec عند 919 مستخدمًا. تهتم الكثير من الأبحاث بتوزيع عبء العمل باستخدام تقنيات LB دون إيلاء الكثير من الاهتمام الى التسامح مع الخطأ (FT) الذي يشير إلى استمرار النظام في العمل حتى عند حدوث الخطأ. لذلك ، اقترحنا تقنية FT للنسخ المتماثل تسمى النسخ الاحتياطي- الأساسي للنسخ المتماثل استنادًا إلى الفاصل الزمني لنقطة التفتيش الديناميكية على FC. تم استخدام نقطة التفتيش لتكرار البيانات الجديدة من الخادم الأساسي إلى خادم النسخ الاحتياطي ديناميكيًا من خلال مراقبة قيم وحدة المعالجة المركزية لخادم الضباب الأساسي بحيث تحدث نقطة التفتيش هذه فقط عندما تكون قيمة وحدة المعالجة المركزية أكبر من 0.2 لتقليل الحمل. أظهرت النتائج أن وقت تنفيذ عملية تصفية البيانات على FC مع نقطة تفتيش ديناميكية أقل من الوقت الذي يقضيه في حالة نقطة التفتيش الثابتة التي لا تهتم لحالة وحدة المعالجة المركزية.

Introduction

IoT allows the connection of different things such as sensors and cellular phones via the Internet [1]. There are numerous transmission protocols used for Machine to Machine (M2M) communication, such as Message Queuing Telemetry Transport Protocol (MQTT) and Hypertext Transport Protocol (HTTP) [2]. Cloud Computing (CC) is the easiest way to gather and process data generated from IoT devices by connecting these devices to cloud servers [3]. According to Cisco, the humans will use more than 50 billion IoT devices/sensors that were planned to be linked to the Internet by 2020, as shown in Figure-1. Also, researchers estimated the number of these things to reach 1 trillion by 2025 [4, 5]. Massive amount of data will be produced by the exponential growth of IoT (edge) devices located nearby from users. As a result of that and the remote location of cloud servers from edge devices, several challenges appear in IoT systems, including network congestion, data loss, and higher latency [6]. Thus, a computing paradigm called Fog Computing (FC) was proposed to process data generated from IoT (edge) devices in real-time [5].

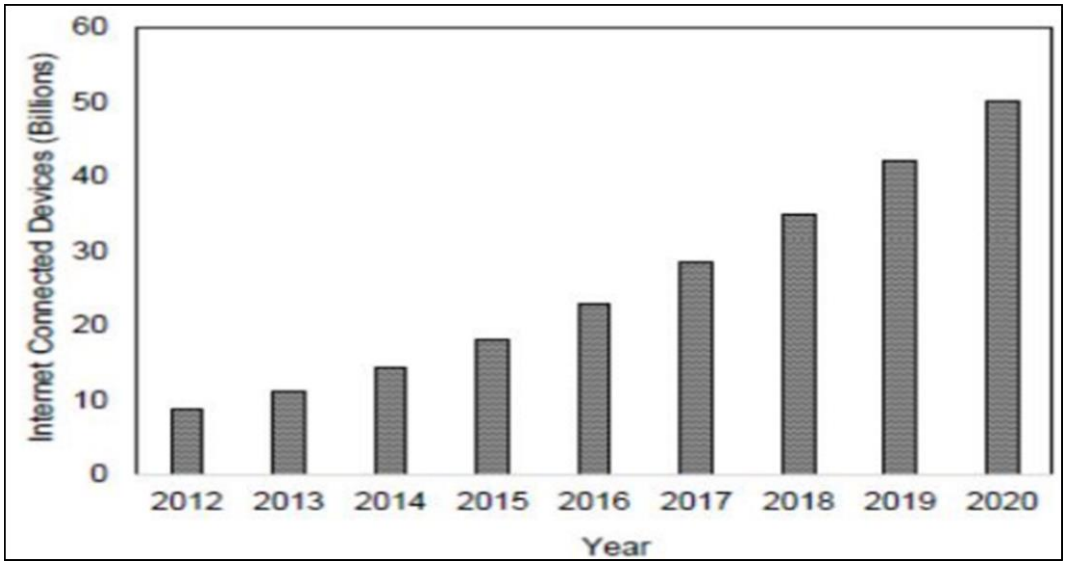

Figure 1- Global change **of** number of Internet connected devices [7].

A server may run slowly during data processing due to the massive amount of data. Therefore, the Load Balancing (LB) technique was used, which is a procedure to distribute the workload statically and dynamically across available servers in the cloud environment. It contributes to increase the throughput of the system and reduce energy consumption. The load balancer is a server used to

implement the LB technique by redirecting requests or workload among available servers to enhance system performance [8]. LB technique in the FC environment is the same operation as in the CC environment, except that Fog Load Balancing (FLB) is closer to IoT (edge) devices and has a better response time as compared to the Cloud Load Balancing (CLB). However, a remarkable problem arises by the possibility of the occurrence of fault. Although LB techniques are used, there is not much attention and alertness to Fault Tolerance (FT) in recent times [9]. FT means the process of executing system tasks which continues even when a fault occurs. A fault is an abnormal state of the component or system that leads to failure [10]. Due to large amount of data that must be processed in the fog servers before it goes to cloud, it is necessary to provide reliability to fog servers in case of failure occurrence, as in the failure of fog servers, especially if these data are critical. Therefore, this paper proposes an FT architecture on fog servers for healthcare data generated from edge devices (sensors).

Related Works

There are many techniques that have been used to achieve FT and LB, some of which are discussed as follows.

Ryuji *et al.* [11] presented results for the non-replication fault tolerance which was applied to protect only data coming from the sensor to the fog server. Thus, the data is directed to the active server but does not provide protection for the data inside the fog server when a fault occurs during processing.

Berkin and Oznur [12] presented many periodic checkpoint algorithms on primary-backup replication and compared them for improving checkpoint time. Nevertheless, all the algorithms used were static without paying attention to the state of the server during the execution of tasks, which resulted in overhead and increased the execution time.

Al-Joboury and Al-Hemiary [13] provided a mechanism to monitor healthcare data in real-time and reduce the congestion on the cloud network. This was achieved by sending the pulse (heartbeat) sensor messages by MQTT protocol to the fog server. These data were then filtered on fog server every 30 minutes by selecting max., min., and avg. values. Then, the data were sent directly to the cloud. However, this paper did not apply a dynamic LB technique to distribute data to more than one fog server, taking into account the current state of servers.

The remainder of the present paper is arranged as an illustration of the system architecture of FC that consists of three layers: IoT (edge) devices, fog, and cloud layer. Then, we present an overview of the importance of using the LB technique and clarify the differences among its types. After that, the importance of the technique used in this paper to achieve fault tolerance is illustrated. Then, we discuss the proposed system. Finally, we present the results and conclude the paper.

The system architecture consists of three layers: IoT (edge) devices, fog, and cloud layer, as shown in Figure-2.

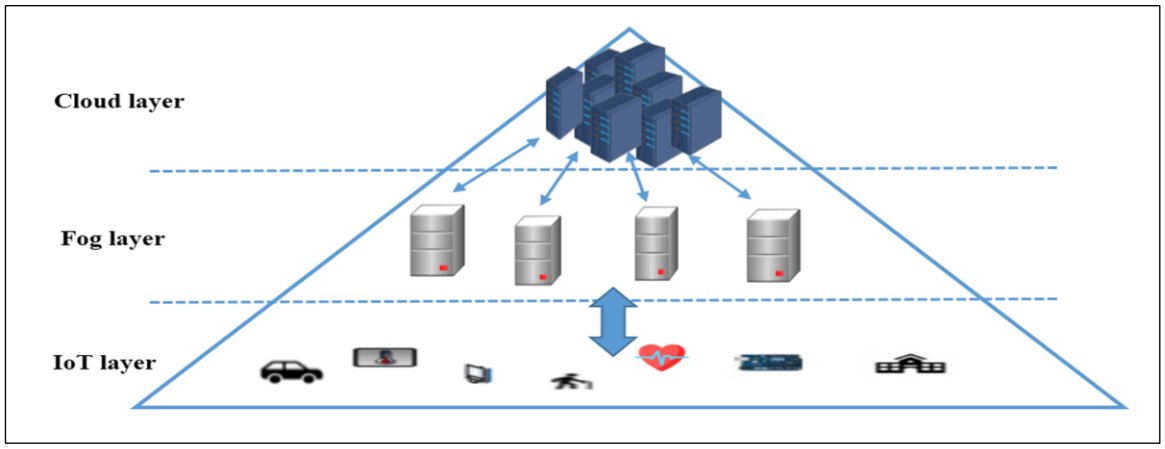

Figure 2- Integrated IoT, fog, and cloud layered architecture.

1. IoT (edge) Devices Layer for Healthcare System

IoT is a trendy technology used in the field of wireless telecommunications that was invented by Kevin Ashton in 1999 to create a bridge connection between physical world and digital world through the Internet. IoT technology is dealing with physical objects that refer to 'things' linked to the Internet such as wireless sensors, actuators, and microcontrollers (MCUs), which enables these things to gather

and exchange data via Internet connection [14]. IoT is a commonly used technology that supports many applications that contribute to improve various areas of life, especially the medical field through healthcare applications. IoT-medical devices, such as medical sensors, are wearable and attached to patient's body. With the usage of IoT-medical devices, specialists can monitor medical parameters such as heartbeat, body temperature, etc. Thus, the number of patient's visits to the hospital will be reduced [15]. In this paper, a heartbeat sensor was used.

2. Fog Layer

FC represents an intermediate layer which functions as a bridge located between IoT (edge) devices and cloud servers. FC was proposed by Cisco in 2012. It presents computing, processing, and temporary storage, where FC is slightly similar to an CC so that it works to bring the cloud services close to IoT (edge) devices [16]. In this paper, FC was proposed to reduce the latency because the fog server is closer to IoT (edge) devices and, thus, it provides monitoring for heartbeat messages on the fog server using data monitoring applications. In addition, FC minimizes congestion on cloud network by sending only the necessary messages from fog-to-cloud (F2C).

3. Cloud Layer

CC is a technology used to store and process data. The access to it is via the Internet rather than the computer's hard drive. CC is a shared pool of computing and permanent storage resources that can be obtained on-demand and is dynamically present to the users. A cloud server is also called a virtual server because one of the advantages of the cloud is virtualization. CC increases data reliability, unlike desktop computing. When using CC, if the personal computer crashes, all data are still present in the cloud, but when using desktop computing, if a hard disk crashes, all valuable data are destroyed [17].

Load Balancing Techniques

Load balancing is a technique used to enhance performance by distributing the workload across various nodes [18]. It is applied to improve throughput and response time, realize efficient resource utilization, avert bottlenecks, and reduce energy consumption [18, 19].

LB techniques are classified, according to system state, into two categories:

1. **Static load balancing [19]:** Static LB technique does not depend on the current system state; therefore, it works well only when the load fluctuation in the servers is low. The main drawback of the static LB technique is that the current system state is not taken into account during decision making and, thus, the workload is distributed equally between servers.
2. **Dynamic load balancing [19, 20]:** In a dynamic LB technique, decision making is taken based on the current system state. As a result, the workload is not always equally distributed between servers. In this technique, the state information is exchanged between servers and consequently the workload is distributed between servers. Therefore, a dynamic LB technique always provides a better load balancing solution. Hence, in this paper, the dynamic LB technique is used.

Fault Tolerance

FT is the ability of the system to remain operating even when the fault occurs. It is primarily used to improve system uptime and ameliorate its reliability and availability. FT is a very important and desirable property that should be available in all applications, especially in critical applications [21]. When the reliability of service is low, it implies that its efficiency will also decrease and the customers will be waiting for a long time for service [22]. Because FC represents an intermediate layer between IoT (edge) devices and cloud servers, it is very important to provide FT and reliability on the fog servers [23]. Therefore, in this paper, the FT technique, based on replication, is used on fog servers. Data replication has many benefits, including FT, and improves data availability at the same time, which increases system strength [24]. FT technique based on replication implies that data are replicated to numerous servers and, if one of the host servers fails, the data are processed successfully as long as there are other copies of the data on other servers [25]. Active replication indicates that the same data are sent to more than one server at the same time. Nevertheless, this method has many disadvantages that are affecting the network performance by increasing traffic in-network and may obstruct the connection in real-time within the network. In addition, active replication leads to overwork of all servers [26]. Therefore, the replication technique used to achieve fault tolerance in this paper is called primary-backup replication or active/passive (A/P) replication. This implies that all data will be transferred first to one server, called the primary server. Then, a checkpoint receives data coming to the primary server every period of time and sends them to another server, called the backup server. By using this technique, the traffic during sending data to the fog server (primary server) will

be reduced and, thus, data (heartbeat messages) can be monitored in real-time on fog server. An additional importance of using checkpoint for each period of time is to avoid the repetition of sending all data of primary server to the backup server. As a result, the amount of data sent is reduced and, thus, the speed of sending data to the backup server will increase.

The main contribution of this paper is to implement a checkpoint to send only new data to a backup server, based on a dynamic interval by monitoring CPU values of the primary fog server. This method contributes to the reduction of the impact of the checkpoint process on increasing the time of the process of filtering the sensor data. Figure-7 illustrates the results. In addition, the Dynamic Weighted Round Robin (DWRR) algorithm will be employed on fog servers in this paper and, consequently, it will increase throughput and reduce the amount of data to be checked and filtered on fog servers.

The Proposed System

The proposed system consists of three layers: the IoT (edge) device layer represented by heartbeat sensor, the fog layer to process data, and the cloud layer to permanently store data sent from the fog server.

From Sensors to Fog Server

A real pulse sensor is placed on the patient's thumb to sense the heartbeat. The pulse sensor will measure the patient's pulses through a given time period. In this paper, patient's pulse is measured every 30 s (time is set during programming) and all values generated will be transferred by the MQTT protocol to the fog server. Only necessary messages will be immediately sent to the cloud.

Fog Environment

The proposed system is represented by the presence of FC within a particular building, like Baghdad University, Department of Computer Engineering.

The fog server temporarily stores data of the heartbeat sensor. These data live within the fog server for 30 minutes. Hence, every 30 minutes, the messages sent to the fog server are filtered by extracting the maximum, minimum, and average values. These values are sent to the cloud in real-time, while the remaining data are deleted from it, as shown in the Figure-3. Thus, the congestion on cloud network was reduced because instead of sending all messages generated from the sensor to cloud, only three messages will be sent to the cloud for the purpose of permanent storage.

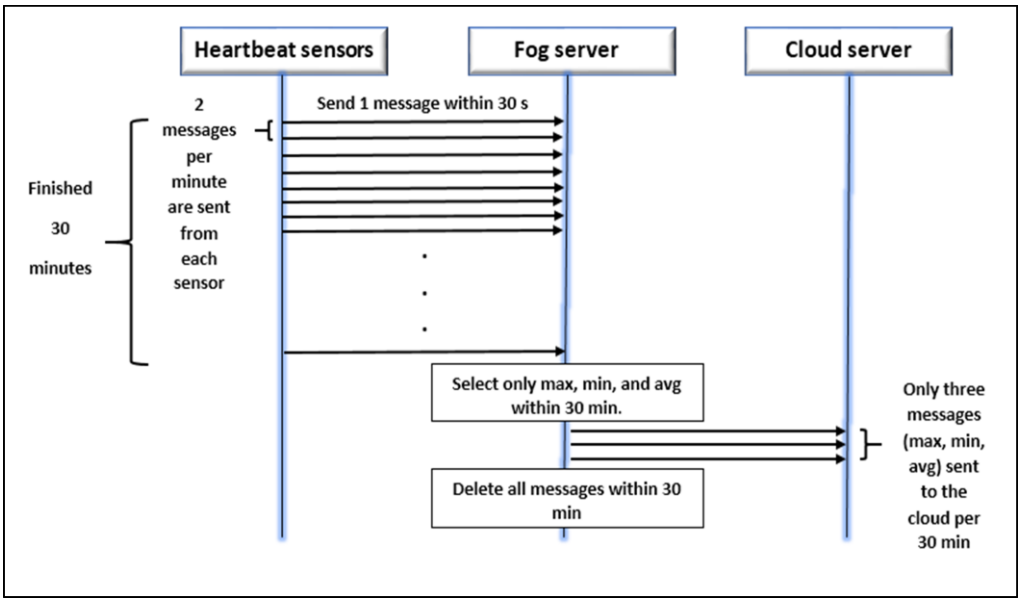

Figure 3- Sequence diagram of data filtering in fog server

Because of the usage of a very large number of sensors that send data to the fog server, a burden on the server is caused. Therefore, it is necessary to take into account the improvement of system performance by distributing the load through more than one server. In addition, the approach used to provide protection for data when it arrives at servers is very important. Therefore, in this section, the

structure of the proposed healthcare system to distribute and protect the data on the fog servers will be illustrated, as shown in Figure-4.

In this paper, VirtualBox is used to create Virtual Machines (VMs) that represent virtual fog servers so as to create the desired number of servers in the fog environment.

Two types of NoSQL databases are used; Redis database on the proxy load balancer, because it is lightweight and very fast for messages broker, and CouchDB on the fog servers because it has an FT storage engine for the safety of data.

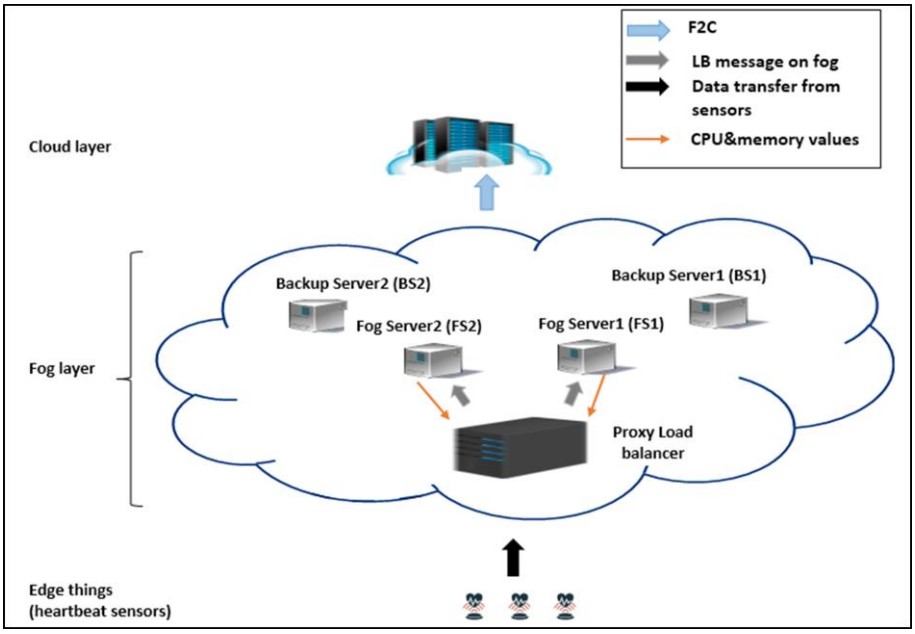

Figure 4- The proposed healthcare system for distribution and protection of the data on the fog servers.

Two fog servers are used in this proposed system to distribute the load transmitted from the sensors. To distribute these data, a proxy load balancer is used.

Proxy Load Balancer: The data generated by a real pulse sensor are transmitted through the MQTT protocol, which is a lightweight protocol. MQTT protocol is based on the publish/subscribe (Pub/Sub) paradigm where publishers represent data generated from sensors or any IoT device, while subscribers represent data of consumers, so that the Pub/Sub paradigm is met by a central node called broker [27]. The topology of the MQTT protocol consists of three parts: Publisher(s), Broker, and Subscriber(s). First, the Publisher(s) sends data to the MQTT Broker for publishing to an address that is called "topic". Then, the Subscriber(s) subscribes to the MQTT broker for this topic [28]. The broker intercedes the exchange of messages between the publishers and subscribers, as performed by Mosquitto which is an open-source MQTT broker. Mosquitto provides a lightweight manner to execute messages using the Pub/Sub paradigm. It is appropriate for IoT messaging uses, such as sensors that have low power, microcontrollers, mobile devices, and embedded devices [29, 30]. Mosquitto broker is installed on the proxy load balancer server. In this proxy server, after subscribing messages by Mosquitto broker, the load is scheduled among Fog Server1 (FS1) and Fog Server2 (FS2) by using the DWRR algorithm. DWRR algorithm depends on the CPU and memory values of fog servers (FS1 and FS2). Fog servers connect to Redis DB that is located on the proxy load balancer server. Proxy uses the Redis database to receive the values of resources (CPU and memory) which are sent from FS1 and FS2. The sorting stage begins with the retrieval of CPU and memory values of FS1 and FS2 from the Redis DB and the comparison of these values. Based on these values, the proxy load balancer server distributes the load on FS1 and FS2, as shown in Figure-4. Thus, if FS1 has CPU and memory values larger than those of FS2, then the weight of messages sent from the proxy load balancer to FS1 is larger than the weight of those of the FS2. DWRR algorithm is illustrated as a flowchart in Figure-5.

After data are distributed based on the DWRR algorithm, data in the fog servers will be preserved by using the primary-backup replication, also called the active/passive (A/P) replication technique.

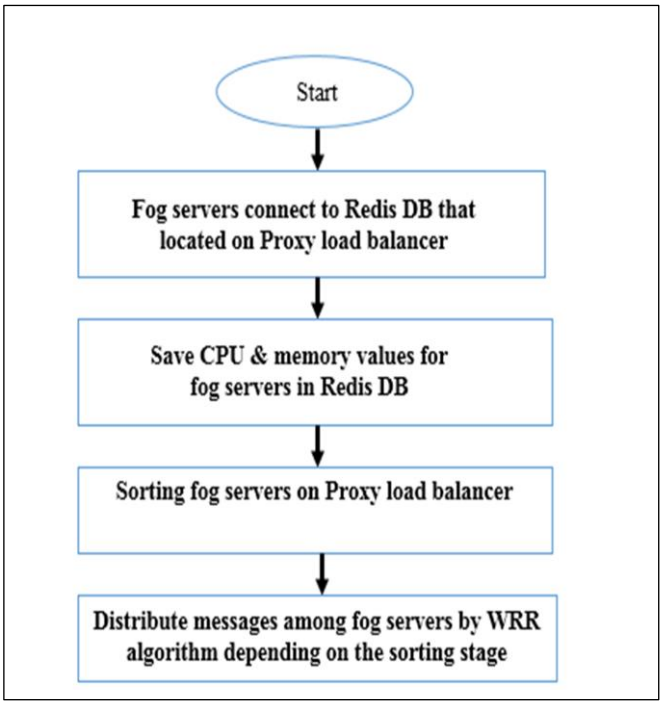

Figure 5- DWRR algorithm flowchart

Fault Tolerance on Fog Servers: Active or primary servers are FS1 and FS2, whereas passive or backup servers are Backup Server1 (BS1) and Backup Server2 (BS2), as shown in Figure-4. BS1 is a backup server for FS1, whereas BS2 is a backup server for FS2. If FS1 and FS2 servers are fault, then BS1 and BS2 servers are activated to do the work of the active servers. The primary server sends data to the backup server continuously, but to avoid repeating sending all the data previously sent to the backup server, the checkpoint is used to distinguish the new data sent to the primary server and that only new data is sent to the backup server. This is caused by periodically comparing data access time to the primary server. If the time of incoming data is greater than the time of the last previous checkpoint occurred, it implies that the data are new and will be sent to the backup server without re-sending all the data previously sent to this server. In the FS1 and FS2, the data checkpoint continues with a periodic interval (static checkpoint interval) even during the time of the data filtering process, which may affect the execution time of the filtering process of data to be sent to the cloud in real-time if the fog server is weak. It was observed by experiments that when the data filtering (every 30 min) occurs with a continuation of the checkpoint of data coming to the server, the data filtering time is greatly affected when the remaining CPU value is 0.2 or less. Therefore, dynamic checkpoint is used by reading the CPU amount of fog servers (FS1 and FS2). If the value of the remaining CPU value is 0.2 or less, the checkpoint is stopped dynamically and returns as soon as the remaining CPU value is greater than 0.2, as shown in the dynamic checkpoint algorithm of the primary server. The dynamic checkpoint interval will contribute for reducing the execution time of the data filtering process in the fog servers.

Dynamic Checkpoint Algorithm of the primary Server

```
Input: Data; Checkpoint
Static initial time
Read CPU values
For incoming data do
    If CPU > 0.2 then
      Start checkpoint every interval time
        If (time of current data > time of previous data) then
          Checkpoint ← current data
          Send Checkpoint to backup server
        End
      End
    Else if CPU ≤ 0.2 then
      Stopped checkpoint process
    End
    Data filtering every 30 min.
End
```

Each backup server sends a request to its primary server every interval time in order to detect the failure when it occurs. Hence, when the primary server does not send a reply to the backup server, it implies a primary server failure. At that time, the backup server is activated to filter data that were sent from the primary server and store the IP address of the backup server in Redis DB, in the place of the server's IP address that stopped working for a purpose to receive data coming from the MQTT proxy.

Results

In the suggested system, two objectives are achieved. First, increasing the throughput by balancing a workload on fog servers. The workload generated by IoT sensors and transmitted based on MQTT protocol was distributed on fog servers. The locust tool was used to generate additional load on the fog environment to measure the throughput. Throughput is the number of requests a server responds to per second. Figure-6 illustrates throughput comparison between DWRR algorithm and a static algorithm, which is called Round Robin (RR) algorithm that sequentially distributes workload by haproxy on fog servers. The throughput results for both algorithms were obtained in Figure-6 by using the locust tool. In the DWRR algorithm, the workload generated from the locust tool is sent to the proxy load balancer, where this proxy distributes the workload among fog servers dynamically based on CPU and memory values of these servers. Whereas in the RR algorithm, the workload is sent to the haproxy to distribute it statically among fog servers without taking into account fog servers' state during distribution. It is noticeable that the throughput of DWRR from the beginning is higher than that of the RR. This increase is more prominent at 567 users where the throughput of DWRR reaches 220 requests per second, while the throughput of RR reaches only 70 req/sec. DWRR throughput continues to increase so that the throughput at 594 users reaches to 780 req/sec, while in RR it is up to 400 req/sec. After that, throughput for RR is increased slightly, reaching 1680 req/sec, while in DWRR it reaches 1410 req/sec at 651 users. Then, DWRR again outperforms RR; at 794 users the DWRR reaches 2755 req/sec while RR reaches 2390 req/sec. RR reaches 2382.4 req/sec at 807 users and stops because of the limitation of haproxy which cannot handle more users. DWRR throughput at the same number of users (807) reaches to 2700 req/sec and continues to increase to a maximum of 3290 req/sec at 919 users.

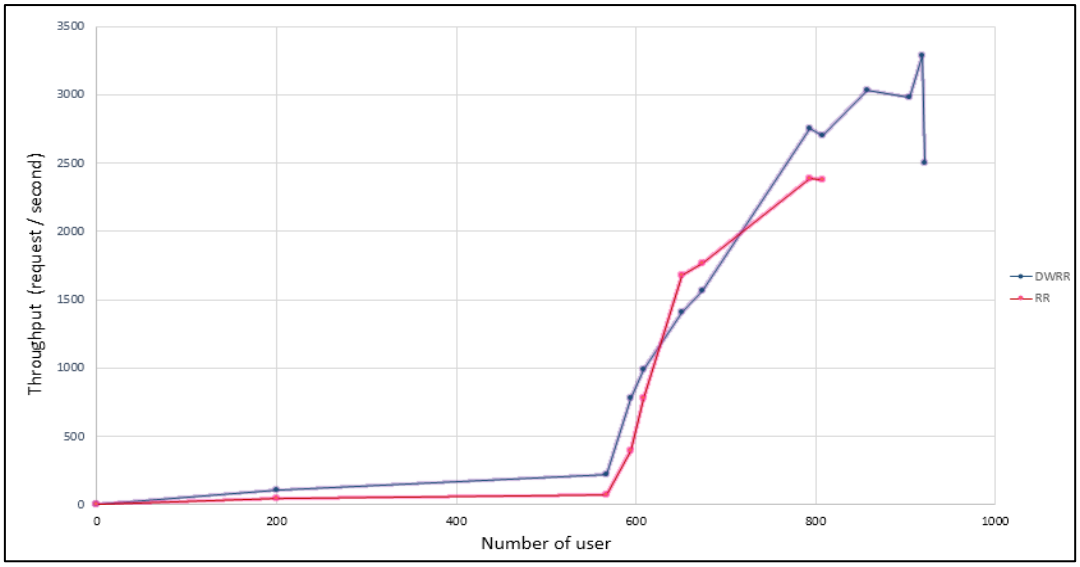

Figure 6- Throughput of the MQTT proxy on fog computing

The second goal is to ensure that data is filtered and sent to the cloud in real time by reducing an overhead that may result from synchronization of the filter and checkpoint, so that increased workload will increment the implementation time of an application.

As shown in Figure-7, time of execution of data filtering is checked in the case of the static and dynamic checkpoints. It is observed that when a checkpoint is made during fixed intervals (static), a very large overhead will be generated, especially when the CPU value is less than or equal to 0.2, then checkpoint with dynamic intervals is used. The results of the static checkpoint were measured by checkpointing the data located on CouchDB every fixed interval, even when the data filtering time occurs, which in turn often causes fatigue to the fog server. Whereas the results of the dynamic checkpoint are measured by check pointing the data only when the CPU value is larger than 0.2. When the number of data in the database is equal to 200, the time taken to filter data and extract max., min., and avg. values only in the dynamic checkpoint is equal to 1515 ms, while the time taken in the static checkpoint is 2145 ms. When the number of data is 350, the time taken to filter data at a dynamic checkpoint is 1781 ms, while the time taken at a static checkpoint is 2222 ms. In case that the number of data is 500, the time at the dynamic checkpoint is 1816 ms, while the that at the static checkpoint is 2229 ms. When the data number reaches 650, the time at the dynamic checkpoint is equal to 1876 ms, while that at the static checkpoint is 2319 ms.

It is noted that there is a clear difference in the time spent for data filtering process in the fog servers between dynamic and static checkpoints.

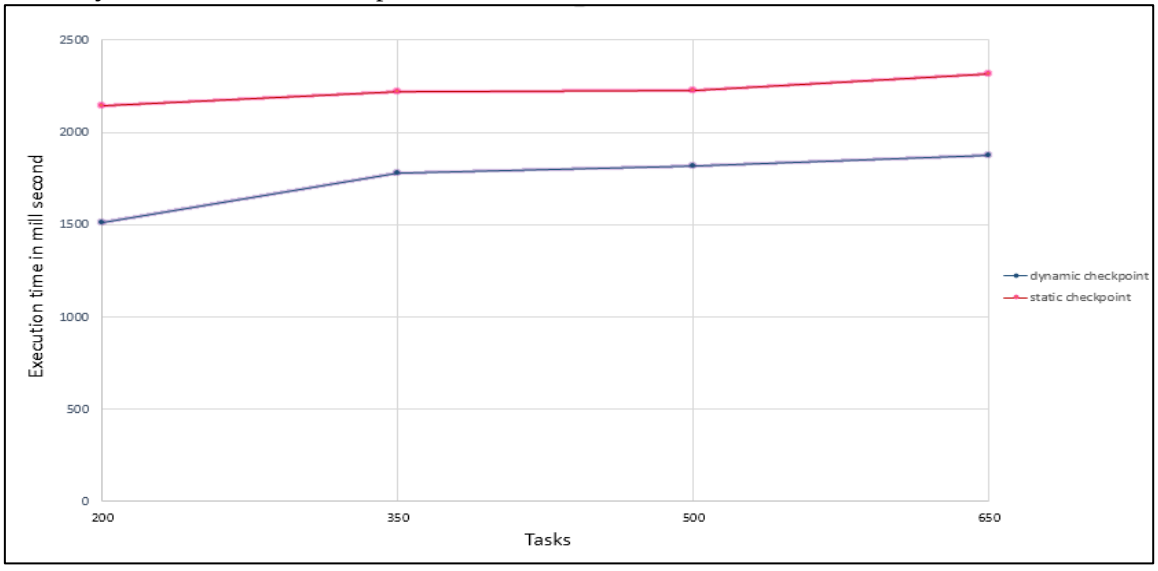

Figure 7- Execution time of data filtering

Conclusions

Major challenges arise when vary large amount of data is generated, resulting from the frequent use of IoT devices, increased latency due to the distance of the cloud, in addition to the increment congestion on the cloud network. Therefore, the proposed system in the present study suggests a new edge-fog-cloud technique with features of low time execution and high throughput. Then data protection in the fog environment was ensured, which is the most important advantage in order to protect the data in case of a fault in the server, especially for the healthcare data. It is concluded that the throughput of the DWRR algorithm is greater than that of the RR algorithm. However, despite the increased throughput in the fog environment, this feature does not work for data protection in the event of a failure in the fog server. Thus, the technique called primary-backup or active/passive (A/P) replication is used based on dynamic checkpoint interval. The dynamic checkpoint interval contributes to reducing the time spent for the data filtering process in the fog servers by stopping the checkpoint when the CPU value is equal or less than 0.2. In the future, the proposed system can be developed by increasing the number of backup servers so that the data will be replicated in more than one server, which leads to increased reliability.

References

1. Bibani O., Mouradian C.**,** Yangui S., Glitho R.H., Gaaloul w., H-Alouane N.B., Morrow M. and Polakos P. **2016**. A Demo of IoT Healthcare Application Provisioning in Hybrid Cloud/Fog Environment. *IEEE*. DOI: 10.1109/CloudCom.2016.0081.
2. Çorak B.H**.,** Okay F.Y**.,** Güzel M., Murt Ş. and Ozdemir S. **2018**. Comparative Analysis of IoT Communication Protocols. *IEEE*. DOI: 10.1109/ISNCC.2018.8530963.
3. Gusev M. and Dustdar S. **2018**. Going Back to the Roots—The Evolution of Edge Computing, An IoT Perspective. *IEEE*, **22**(2)**.** DOI: 10.1109/MIC.2018.022021657.
4. Zhang P., Liu J. K., Richard Yu F., Sookhak M., Ho Au M. and Luo X. **2018**. A Survey on Access Control in Fog Computing. *IEEE*, **56**(2). DOI: 10.1109/MCOM.2018.1700333.
5. Dastjerdi A. V. and Buyya R. **2016**. Fog Computing: Helping the Internet of Things Realize Its Potential. *IEEE***, 49**(8). DOI: 10.1109/MC.2016.245.
6. Cai J., Luo Y., Zheng F., Zhang J., and Luo Q. **2019**. Research and Application of Intelligent Internet of Vehicles Model Based on Fog Computing. *IEEE*. DOI: 10.1109/ITNEC.2019.8729045.
7. Ahmed E., Yaqoob I., Hashem I. A. T., Khan I., Ahmed A. I. A., Imran M. and Vasilakos A. V. **2017**. The role of big data analytics in Internet of Things. *Computer Networks.* doi*:* 10.1016/j.comnet.2017.06.013.
8. Duggal A.K. and Dave M. **2020**. A Comparative Study of Load Balancing Algorithms in a Cloud Environment. *Springer*. doi.org/10.1007/978-981-15-0222-4_10.
9. Abdulhamid S. M., Abd Latiff M. S., Madni S. H. H. and Abdullahi M. **2016**. Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. *Springer*, **29**: 279. doi.org/10.1007/s00521-016-2448-8.
10. Gupta R., Kamal R. and Suman U. **2017**. A QoS-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services. *Springer*. doi.org/10.1007/s41870-017-0066-z.
11. Oma R., Nakamura S., Duolikun D., Enokido T. and Takizawa M. **2019**. Fault-Tolerant Fog Computing Models in the IoT. *Springer*. doi.org/10.1007/978-3-030-02607-3_2.
12. Guler B. and Ozkasap O. **2017**. Analysis of Checkpointing Algorithms for Primary-Backup Replication. *IEEE*.
13. Al-Joboury I. M. and Al-Hemiary E. H. **2017**. F2CDM: Internet of Things for Healthcare Network Based Fog-to-Cloud and Data-in-Motion Using MQTT Protocol. *Springer*. doi.org/10.1007/978-3-319-68179-5_32.
14. Stergiou C. and Psannis K. E. **2016**. Recent advances delivered by Mobile Cloud Computing and Internet of Things for Big Data applications: a survey. doi.org/10.1002/nem.1930.
15. Sony P. and Sureshkumar N. **2019**. Concept-Based Electronic Health Record Retrieval System in Healthcare IOT. *Springer*. doi.org/10.1007/978-981-13-0617-4_17.
16. Mahmud R., Kotagiri R. and Buyya R. **2018**. Fog Computing: A Taxonomy, Survey and Future Directions. *Springer*. doi.org/10.1007/978-981-10-5861-5_5.

17. Khaled Salah Mohamed. **2019**. IoT Cloud Computing, Storage, and Data Analytics. *Springer*. doi.org/10.1007/978-3-030-18133-8_4.
18. Chawla A. and N. S. Ghumman N. S. **2018**. Package-Based Approach for Load Balancing in Cloud Computing. *Springer*. doi.org/10.1007/978-981-10-6620-7_9.
19. Kumar P. and Kumar R. **2019**. Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey. *ACM*, **51**(6). doi>10.1145/3281010.
20. Puthal D., Obaidat M. S., Nanda P.**,** Prasad M.**,** Mohanty S. P. and Zomaya A. Y. **2018**. Secure and Sustainable Load Balancing of Edge Data Centers in Fog Computing. *IEEE*, **56**(5). DOI: 10.1109/MCOM.2018.1700795.
21. **21.** Souza A., Papadopoulos A. V., Bolivar L. T., Gilbert D. and Tordsson J. **2018**. Hybrid Adaptive Checkpointing for Virtual Machine Fault Tolerance. *IEEE*. DOI: 10.1109 /IC2E. 2018.00023.
22. Madani S. S. and Jamali S. **2018**. A Comparative Study Of Fault Tolerance Techniques In Cloud Computing. *International Journal Of Research In Computer Applications And Robotics*.
23. Mohamed, N., Al-Jaroodi J. and Jawhar I. **2019**. Towards Fault Tolerant Fog Computing for IoT-Based Smart City Applications. *IEEE*. DOI: 10.1109/CCWC.2019.8666447.
24. Qaim W. B. and Özkasap Ö. **2019**. State-of-the-Art Data Replication Techniques in IoT-Based Sensor Systems. *IEEE*. DOI: 10.1109/GLOCOMW.2018.8644438.
25. Oma R., Nakamura S., Enokido T. and Takizawa M. **2018**. Hybrid Replication Schemes of Processes in Energy-Efficient Server Clusters. *Springer*. DOI 10.1007/978-3-319-65521-562.
26. Eckhart M. and Ekelhart A. **2018**. A Specification-based State Replication Approach for Digital Twins. *ACM*. doi.org/10.1145/3264888.3264892.
27. Bacco M., Boero L., Cassará P., Colucci M., Gotta A., Marchese M. and Patrone F. **2019**. IoT Applications and Services in Space Information Networks. *IEEE*.
28. La Marra A., Martinelli F., Mori P., Rizos A. and Saracino A. **2017**. Improving MQTT by Inclusion of Usage Control. *Springer*. doi.org/10.1007/978-3-319-72389-1_43.
29. Jutadhamakorn P., Pillavas T., Visoottiviseth V., Takano R., Haga J. and Kobayashi D. **2017**. A Scalable and Low-Cost MQTT Broker Clustering System. *IEEE*. DOI: 10.1109 /INCIT. 2017.8257870.
30. Tantitharanukul N., Osathanunkul K., Hantrakul K., Pramokchon P. and Khoenkaw P. **2017**. MQTT-Topics Management System for Sharing of Open Data. *IEEE*. DOI: 10.1109/ ICDA MT .2017.7904935.