Enabling IoT/M2M System Scalability with Fog Computing

Yuan-Han Lee and Fuchun Joseph Lin(✉)

Department of Computer Science, College of Computer Science,
National Chiao Tung University, Hsinchu, Taiwan
{henry19950709.cs02g,fjlin}@nctu.edu.tw

Abstract. As increasingly more IoT/M2M devices are connected to Internet, they will cause serious congestion to IoT/M2M systems normally deployed in the cloud. Although Cloud can scale out to support more data requests, it may not be able to satisfy the low latency demanded by certain IoT/M2M applications. Fog, as an edge of Cloud, can alleviate the congested problem in the cloud and provide low latency for critical IoT/M2M applications due to its proximity to IoT/M2M devices. In this research, we propose (1) utilizing oneM2M, a global IoT/M2M standard, as the middleware to connect the cloud and the fog, (2) using Traffic Classifiers to intercept and divert IoT/M2M traffic requiring low latency to Fog and (3) deploying independent scalability mechanisms for Cloud and Fog. We demonstrate and verify our scalability design using a smart hospital use case and show that our proposed system can achieve better scalability results in terms of latency, CPU usage and power consumption compared to those with only Fog or Cloud.

Keywords: Scalability · Cloud computing · Fog computing · oneM2M · IoT · M2M · OpenStack · Kubernetes

1 Introduction

With an estimate of 20 billion IoT/M2M devices connected to the Internet in 2020 by Gartner, it is foreseen that the cloud-based IoT/M2M systems will soon face the issues of network congestion. Traditionally, an IoT/M2M system can achieve scalability solely in the cloud. Nevertheless, it may still fail to deliver the low latency demanded by some IoT/M2M applications such as those for eHealth and video streaming. In this research, we propose the utilization of "Fog Computing" [1] to help Cloud handle the data traffic that requires low latency. Also, to increase the overall system capacity, scalability mechanisms for Cloud and Fog are independently designed while assuring the demands from low latency applications are met by Fog.

Fog, as an extension of Cloud, attempts to move the cloud capacity such as compute, network and storage to the edge [2]. As such, Fog is much closer to end users and IoT/M2M devices than Cloud. The Fog node is the key component in the Fog architecture. Before data is sent to Cloud, it will be sent to Fog first for filtering and preprocessing.

B. Li et al. (Eds.): IoTaaS 2019, LNICST 316, pp. 489–502, 2020.
https://doi.org/10.1007/978-3-030-44751-9_41

Fog then can decide whether it should send data to Cloud for further processing or just finish the processing locally. In this way, Fog not only can reduce the load of the cloud but also can reduce processing latency of IoT/M2M applications.

In our research, in order to achieve overall system scalability while accommodating the demands from low latency applications, we propose adding a Traffic Classifier in the Fog architecture to identify IoT/M2M traffic requiring low latency. For IoT/M2M systems in Cloud and Fog, we use oneM2M, a global IoT/M2M standard, as the middleware to integrate both Cloud and Fog. Furthermore, independent scalability mechanisms are designed for them in order to dynamically scale out/in the respective oneM2M MN-CSE (Middle Node – Common Service Entity) and IN-CSE (Infrastructure Node – Common Service Entity) instances according to the IoT/M2M traffic load.

Smart Hospital [3] is an ideal use case for applying system scalability of Cloud and Fog. There are several IoT/M2M applications deployed in such a hospital and each one has different demands for latency. As the hospital grows and the patient needs evolve, the smart hospital should keep up with these demands with its system scalability design.

The rest of this paper is organized as follows. Section 2 gives a survey of related work and explains our motivation and unique contribution. Section 3 describes the high-level architecture of our proposed system. Section 4 presents our implementation details. Section 5 compares our proposed system to that with only Fog or Cloud and explains our experimental results for the smart hospital use case. Finally, Section 6 presents our conclusion and future work. The main abbreviations used throughout the paper are summarized in Table 1.

2 Related Work

Scalability research is done based on either Cloud or Fog computing. In addition, there are also research efforts to explore the collaboration between Cloud and Fog. As such, we categorize our survey into three areas in the following.

2.1 Scalability Research Based on Cloud Computing

Cerritos et al. [4] designed a Master Node in the cloud that is aware of system resources and traffic load so that it not only can decide load balancing policies but also proactively

Table 1. Summary of abbreviations

Abbreviation	Expansion
MN	Middle Node
IN	Infrastructure Node
CSE	Common Service Entity
AE	Application Entity
TC	Traffic Classifier

react to scalability needs. Bastida et al. [5] proposed a highly scalable OpenStack-based architecture for IoT/M2M platforms by taking advantage of functionalities of OpenStack and introducing a master node cooperating with a load balancing queue for fair distribution of incoming traffic among platform nodes.

2.2 Scalability Research Based on Fog Computing

As the latency demand of IoT/M2M applications becomes increasingly more important, Fog is regarded as an alternative to Cloud in this issue. Sen et al. [6] implemented an auto-scaled IoT broker Nucleus with MQTT in the fog architecture that can scale as the number of IoT devices increases. Tseng et al. [7] integrated Middle Nodes of oneM2M with highly scalable container-based Fog nodes to develop a scalable fog network that can dynamically scale in/out oneM2M instances along with Fog nodes.

2.3 Collaboration Between Cloud and Fog

By collaborating with Fog, both latency and congestion in Cloud can be greatly improved. Hence there has been active research on the collaboration between Cloud and Fog. Zhao et al. [8] proposed an approach for edge-node-assisted data transmission in the cloud-centric IoT architecture to overcome the problem of overwhelming bandwidth consumption in the cloud. Chen et al. [9] took advantage of Fog to reduce the workload of Cloud by designing an innovative scheduling mechanism to optimize the dispatch of cloud and fog computing resources.

With the heterogeneous nature of IoT applications, processing them solely in the cloud or in the fog is not sufficient to meet all their QoS requirements. Although some collaboration models between Cloud and Fog have been proposed, these models focus on load sharing between Cloud and Fog than the overall system scalability. In our research, we adopt a different approach: First, instead of collaboration we just let Cloud and Fog each handle the IoT traffic they are good at. Second, we enhance each with independent scalability mechanism to support the huge amount of IoT/M2M traffic.

To accomplish our scalability objective: (1) we design the Traffic Classifiers in the fog architecture to identify the IoT/M2M traffic that requires low latency, (2) we utilize oneM2M as a middleware to connect Cloud and Fog, and (3) we enable both Cloud and Fog to dynamically scale out/in their respective oneM2M instances according to the incoming traffic load.

3 Proposed System Architecture

In this section, we explain the high level architecture and communication interfaces of our system. As depicted in Fig. 1, starting from the bottom is Sensor Network that collects and sends the IoT traffic upward. Next above is Fog Network where each fog local area network is equipped with an embedded Traffic Classifier for identifying the IoT traffic to be processed in Fog. Finally, on the top is Cloud Network with multiple cloud nodes; each has its own IoT platform to handle the IoT Traffic.

Though not explicit in Fig. 1, oneM2M is adopted as the communication middleware of our system that will be explained below.

3.1 Sensor Network

Sensor Network consists of multiple types of sensor devices. They can be regarded as representing IoT applications and each application requires different QoS for latency. For example, the ehealth application demands low latency processing to ensure quick report of a patient's medical status, while clinic historical information can endure high latency due to its nature of information retrieval and storage.

3.2 Fog Network

Fog Network is an LAN close to Sensor Network. Traffic Classifiers, as the first points of receiving the IoT traffic, are capable of distinguishing whether an IoT application requires low latency processing or not. Then, it will decide whether the traffic should be processed immediately locally or go to the Cloud Network according to the application nature. Fog Network consists of a hierarchy of Fog Nodes as illustrated in Fig. 1. In order to build a highly scalable Fog architecture, Fog Network must not only balance the load of each Fog Node but also have the scalability mechanism to scale out/in these nodes according to the overall workload.

3.3 Cloud Network

On the other hand, Cloud Network is responsible for handling all other IoT traffic which have no low latency demand. It consists of only a single level of Cloud nodes as illustrated in Fig. 1. Similar to Fog Network, Cloud Network can distribute the load to different Cloud Nodes and scale them up/down according to the workload.

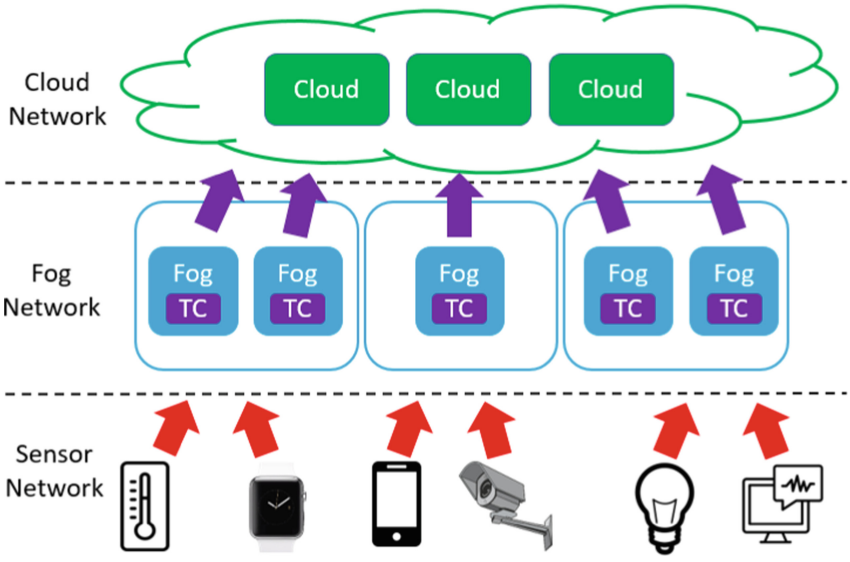

Fig. 1. Proposed system architecture

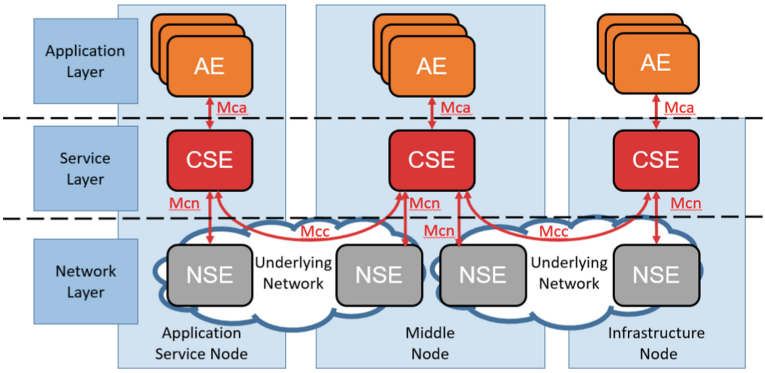

Fig. 2. oneM2M functional architecture

3.4 Communication Interface

In order to construct the above components, a middleware is required to support the communication and connection among them. Our research utilizes the standardized IoT platform, oneM2M [10], as the middleware. Figure 2 depicts the oneM2M architecture in our system which consists of a infrastructure domain and a field domain. Besides IoT/M2M devices represented by Application Service Node (ASN) and Application Dedicated Node (ADN), we focus on the other two types of nodes: Infrastructure Node (IN) and Middle Node (MN). IN is an IoT/M2M server that is normally deployed in the cloud while MN resides between IoT/M2M devices and Cloud. Both of them consist of three functional entities: Application Entity (AE), Common Service Entity (CSE) and Network Service Entity (NSE). AE is an application service. CSE provides common service functions (CSFs) which maintain communication interfaces in oneM2M. NSE is responsible for providing the interface to the underlying transport network. The oneM2M supports HTTP binding and provides Restful APIs over each layer. The oneM2M also defines several reference points among its entities: Mca between AE and CSE; Mcc between CSEs; and Mcn between CSE and NSE.

IN is related to MN as how cloud is related to Fog as there are many common features between them. In our research, MNs are thus deployed in Fog nodes and used to offload the congested IN normally deployed in the cloud. These MNs on Fog nodes support low latency required for critical IoT/M2M applications.

4 Detailed Design and Implementation

This section illustrates the implementation details of our system architecture introduced in Sect. 3, including its applications to a smart hospital use case.

4.1 Sensor Network

We assume three kinds of IoT/M2M devices deployed in the hospital. Each one produces a specific type of data traffic:

- *Heartbeat Data* - produced by the heart monitoring devices to ensure the wellness of the patients with heart disease.
- *Video Stream* - produced by video cameras installed in the hospital to monitor patients and detect any anomaly.
- *Personal and Clinic Historical Information* - produced by the hospital information system terminals to track patients' personal data and clinic historic record for reference and analytics.

To simulate the three kinds of devices above we design a multi-thread Traffic Generator configurable in terms of number of thread, data size and data frequency. Each simulated device can send IoT data to Fog Network via HTTP Post by specifying the URI address of a oneM2M AE (Application Entity) such as m2m/video_app/video_container.

4.2 Fog Network

Our design of Fog Network, as depicted in the lower part of Fig. 3, is based on the open-source container orchestration system, Kubernetes [11]. Fog Nodes are constructed as Kubernetes pods managed by kubelet and communicate with each other using Flannel Container Network Interface (CNI).

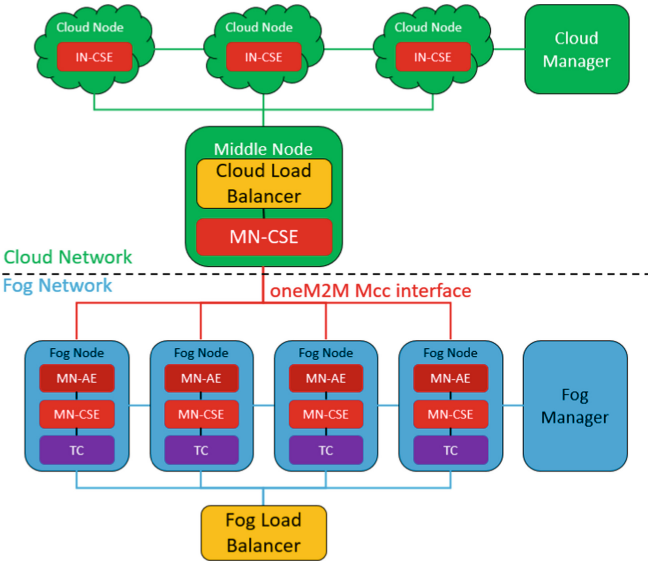

Fig. 3. Detailed system architecture

The components of Fog Network and its associated scalability mechanism are described below.

1. *Fog Manager:* Fog Manager is implemented based on Kubernetes Metrics Server [12] that can monitor the CPU utilization of Fog nodes periodically. Through API, the resource usage metrics for pods and nodes can be monitored. Fog Manager then can scale Fog nodes out and in according to the workload discovered.

2. *Fog Node:* We construct Fog nodes as a single-level architecture and each one consists of three entities: Traffic Classifier (TC), oneM2M MN-CSE and oneM2M MN-AE.

TC is the first point of Fog Network for receiving IoT traffic from Sensor Network. It decides whether to relay the traffic to remote Cloud or keeps it locally. Assuming the kinds of IoT applications coming to the IoT/M2M system are known in advance, a mapping table is pre-provisioned in Traffic Classifier to specify the latency demand of each type of IoT applications. When Traffic Classifier receives an HTTP request, it will parse the HTTP post to get the URI of oneM2M AE and use it to determine its latency demand from the mapping table.

Among the three IoT devices in our hospital use case, heart monitoring devices require ultra-low latency response to ensure the quick report of a patient's medical status. Video cameras also demand low latency for anomaly detection. On the other hand, personal and historical clinic information has no need of low latency because it's for information retrieval and storage. Consequently, Traffic Classifier would divert the first two types of traffic to local Fog nodes for fast processing and the last one to Cloud Network for regular processing.

Each MN-CSE in Fog Network registers to the upper MN-CSE that acts as the load balancer in the Cloud Network. As a result, when TCs want to forward the traffic to Cloud, MN-CSEs in Fog nodes can retarget these requests to the MN-CSE Load Balancer in the cloud via the oneM2M Mcc interface. MN-AE subscribes to the MN-CSE in the same Fog node so that it can receive notification from MN-CSE when there's any new IoT traffic coming in.

When TC forwards the IoT traffic requiring low latency to the MN-CSE in Fog node, the MN-CSE stores the incoming data and sends notification to the MN-AE. MN-AE will either analyze the heartbeat data or process the video stream to determine whether there is any anomaly. If any anomaly is detected, MN-AE will trigger an alarm for special actions.

3. *Fog Load Balancer:* Fog Load Balancer is used to distribute the incoming data traffic to each Fog node fairly. It is designed based on RabbitMQ [13], an open-source message-broker software supporting Advanced Message Queuing Protocol (AMQP). Load Balancer is implemented in Remote Procedure Call (RPC) consisting of the request channel and the response channel. The RPC client in Load Balancer forwards the HTTP requests to the RPC servers in each Fog node via the request channel. Next, the RPC server in each Fog Node translates the messages to a specific format for inserting IoT data to oneM2M; it then sends a response back to the RPC client via the response channel. The response would be forwarded through Load Balancer to IoT/M2M devices in Sensor Network.
4. *Scalability Mechanism:* We define two thresholds as depicted in Fig. 4: Scale In (10%) and Scale Out (50%). Whenever Fog Manager detects that the average CPU utilization of Fog nodes is higher than the Scale Out threshold, it will scale out one more Fog node. On the other hand, if the average CPU utilization of Fog nodes is lower than the Scale In threshold, Fog Manager will scale in one Fog node.

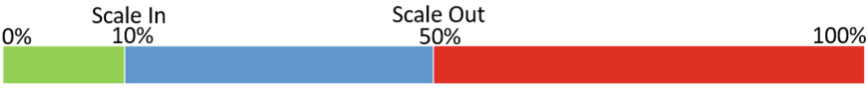

Fig. 4. The threshold of scalability mechanism

4.3 Cloud Network

We depict the architecture of our scalable Cloud Network in the upper part of Fig. 3. We use the open-source cloud operating system, OpenStack [14], to construct our Cloud Network. All the components as discussed below are virtual machines managed by OpenStack Nova and communicate with each other using OpenStack Neutron.

1. *Cloud Manager:* Cloud Manager manages the scale-out and scale-in of Cloud nodes with the following operations: (1) Cloud nodes send their CPU status to Cloud Manager periodically, (2) At CPU overload, Cloud Manager would scale Cloud nodes up while at CPU underload, it would scale Cloud nodes down.
2. *Cloud Node:* We provision a oneM2M IN-CSE into each Cloud node so that each Cloud node becomes a oneM2M server capable of processing the personal and clinic historical information. In addition, a plugin is embedded in each Cloud node for reporting its CPU utilization to Cloud Manager.
3. *Cloud Load Balancer:* Cloud Load Balancer is integrated in oneM2M Middle Node. When MN-CSE receives the data from TCs in Fog Network, it will redirect the data to Cloud Load Balancer immediately instead of processing it. Similar to Fog Network, Cloud Load Balancer for Cloud Nodes is also implemented based on RabbitMQ with RPC. The RPC client in Load Balancer forwards the HTTP requests to the RPC servers in each Cloud node via the request channel. After RPC servers send the data to IN-CSE, the response from the Cloud node is then sent back via the response channel to TCs, then to Sensor Network.
4. *Scalability Mechanism:* We define the same two thresholds: Scale In (10%) and Scale Out (50%) as depicted in Fig. 4, Once the average CPU utilization of Cloud nodes calculated by Cloud Manager exceeds Scale Out threshold, Cloud Manager will create and add one more Cloud node. Conversely, if the average CPU utilization of Cloud nodes is lower than Scale In threshold, Cloud Manager will stop and remove one Cloud node.

5 Experiment and Evaluation

In this section, we will show the experimental testing of the proposed architecture and the evaluation of testing results.

5.1 Testbed Environment

Our testbed environment is shown in Table 2. In Cloud Network, OpenStack Newton is deployed while in Fog Network, Kubernetes v1.12.1 and Docker [15] 18.06.1-ce are

installed. We use OM2M [16], an open source implementation of oneM2M from LAAS-CNRS, to construct oneM2M MN and IN. Cloud Network consists of two physical machines: one for Controller and one for Compute Node. Fog Network consists of four physical machines: one for K8S Master and three for K8S Nodes. Besides, we set the minimum number of Cloud nodes to 2, the maximum number to 5 and each one runs on a virtual machine with 8CPU, 4 GB of RAM and 30 GB of HDD. On the other hand, we set the minimum number of Fog nodes to 2 and the maximum number to 7, each one running on a Kubernetes pod with 1 CPU, 2 GB of RAM and 15 GB of HDD.

Table 2. Testbed hardware information

Component	Operating system	CPU	RAM	Machine	Disk
Sensor network	Ubuntu 16.04	Inter Core i5-8400 CPU @2.80 GHz (6cores)	8 GB	1 Desktop	256 GB
Fog network		Intel Core i5-4200H CPU @ 2.80 GHz (4cores)	8 GB	4 Laptops	512 GB
Cloud network		Intel Core i7-8700 CPU @ 3.20 Ghz (12cores)	64 GB	2 Desktops	1 TB

To simulate the proximity of Fog Network and the Cloud Network to Sensor Network, we configure the network in such a way that the average response time from Fog Network is 6.7 ms while the one from Cloud Network is 210 ms with 1000 simple HTTP requests sent to each site. This simulates the distance from Sensor Network to Fog Network and Cloud Network, respectively.

5.2 Experiment Setup

We design a Traffic Generator that is a multi-thread program, to simulate the IoT traffic of heartbeat data [17], video stream [18] and clinic information [19] as described in Sect. 4.1. Traffic Generator can configure any number of devices with different data frequency and data size such as those in Table 3. used in our experiments. The settings

Table 3. Traffic generator configuration

IoT traffic	Thread number (low/high)	Data frequency	Data size
Heartbeat data	2/10	10 records/s	170 B
Video stream	2/10	10 records/s	20 KB
Clinic information	4/20	2 records/s	950 B

are chosen to reflect the relatively small size but high frequency of heartbeat data, the large data size and high frequency of video stream though our IoT/M2M system only stores the metadata of video instead of the whole media. Furthermore, we assume Personal and Clinic Historical Information is the major application data traffic in the smart hospital so its number of devices is set to be the largest. We intend to show that by letting Cloud and Fog handle the type of traffic they are good at, we can achieve better overall system scalability than sending all the traffic to either Fog or Cloud alone.

We test our system with two scenarios: high and low traffic loads. The amount of high traffic is five times than that of low traffic as indicated in Table 3. In our experiment, we send both high and low traffic loads for 5 min respectively to our proposed system and compare the scalability of our system versus those with only Fog or Cloud in terms of *latency*, *CPU usage* and *power consumption* per each IoT request. These three features can represent how well the scalability of a system has been designed. Ideally, for the system with good scalability design, when the amount of processed data increases, it would cause lower latency, lower CPU resource and lower power consumption than the ones with worse scalability.

5.3 Evaluation Result

We expect our system can leverage the benefits of dual scalability mechanisms of Fog and Cloud to outperform the other two compared systems in majority of our test cases.

- Latency

In order to explain the benefits of our proposed system, we define maximum tolerable latency of each application. For the application which would be forwarded to Fog Network, the value of heartbeat data is set at 70 ms and the one of video stream is set at 250 ms. On the other hand, the value of Personal and Clinic Historical Information is set at 4 s because it doesn't require low latency response.

The evaluation results of latency are shown in Fig. 5. Under the condition of low traffic, our proposed system and the one with only Fog can meet all the latency demands

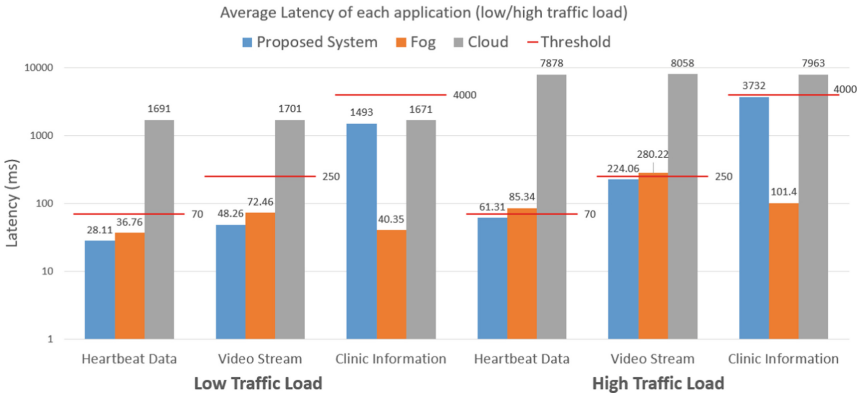

Fig. 5. Latency results for three applications in low/high traffic load

of three IoT applications. However, the one with only Cloud cannot provide low latency to heartbeat data and video stream due to the long haul communication. On the other hand, only our proposed system satisfies all latency requirements in the case of high traffic thanks to the effort of Traffic Classifier. Similar to the situation in low traffic, the system with only Cloud cannot even meet one of them. In addition, the system with only Fog has to allocate part of computing resources for Personal and Clinic Historical Information, so it cannot fully focus on handling the other traffic requiring low latency.

- CPU Usage

Figure 6 depicts the result of Average CPU usage. These results measured average CPU utilization of physical machines for computing. To calculate the overall CPU usage of our proposed Cloud-Fog system, we survey a CPU benchmark website [20] to compare the CPU of both sites so that we can get the CPU usage of our hybrid system by using the weighted average method. For the low traffic load, the system with only Fog outperforms the others due to its lightweight characteristics. On the other hand, Cloud has the highest CPU usage due to the needs to maintain heavyweight Virtual Machines even in the light traffic load.

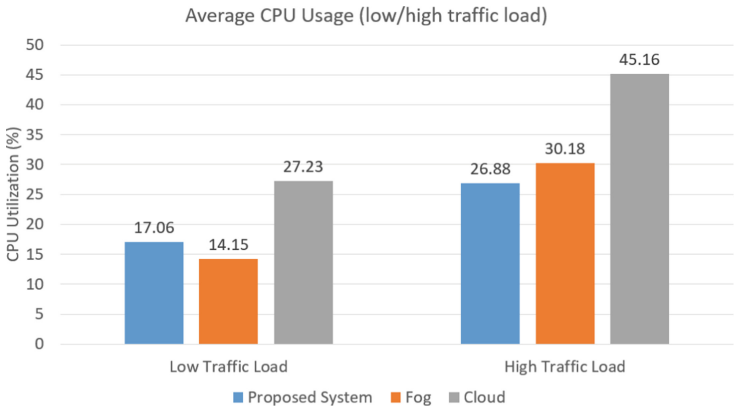

Fig. 6. Result for CPU usage in low/high traffic load

However, our proposed system shows the best result in the condition of high traffic load. That's because when the amount of data traffic increases, the value of Traffic Classifier and dual scalability mechanisms stands out. Both Fog and Cloud can fully utilize their computing resources to handle the traffic that they are good at instead of spending extra resource to process all three types of traffic.

- Power Consumption

Figure 7 shows the results of average power consumption per request for three applications. We calculate the total power consumption in the period of our experiment and

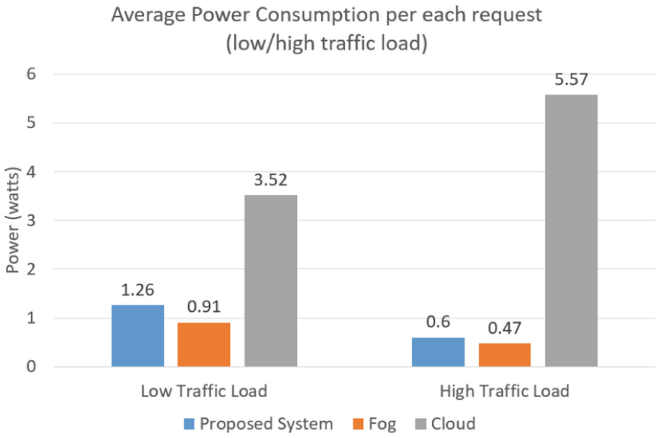

Fig. 7. Result for average power consumption per request in low/high traffic load

derive the results by dividing it by the total number of requests. We use the following formula (1) to calculate the power consumption:

$$Power\ Consumption = TDP * CPU\% + K * Memory\% \tag{1}$$

where TDP is the microprocessor's Thermal Design Power, a reference measurement of CPU running in normal conditions and given by the manufacturer. For our particular testbed, TDP for the machines in Cloud is 65 W and for the ones in Fog is 47 W. K is the common power consumption of memory modules [21], and it's 6 W for the memory in Cloud and 4 W for the one in Fog. CPU% and Memory% are the average CPU utilization and memory usage of the machines.

The results show that our proposed system and the one with only Fog consume less power consumption *per request* in the high traffic load than in the low traffic load, because both sites leverage Fog Computing to handle the traffic. Although they may consume more total power in high traffic than in low traffic, the amount of requests they can handle also largely increases due to low latency communications and scalability mechanisms. As the impact of the increase in the amount of requests is greater than the increase of total power usage, the average power consumption per request becomes dramatically lower. On the other hand, the system with only Cloud performs worse in the high traffic load than in the low traffic load because its high latency characteristics produces more impact to the average power consumption per request.

In the two conditions of traffic load, our proposed system performs better than the one with only Cloud but it doesn't perform better than the one with only Fog because we leverage both Cloud and Fog to handle the IoT traffic so it's reasonable that our proposed system consume more power than the system with only Fog.

In summary, only our proposed system can meet all the latency requirements of three applications even though it doesn't always perform the best in terms of the average power consumption per request. However, for smart hospital latency should be the first priority consideration. This verifies that our proposed architecture is indeed the best choice in the case of smart hospital applications.

6 Conclusion and Future Work

In this research, we propose the use of Fog Computing to help Cloud handle the IoT/M2M applications requiring low latency. Accordingly, we introduce a Traffic Classifier which is deployed near the Sensor Network, to divert different IoT traffic to Fog Network or Cloud Network according to their latency requirements. We also propose the use of standardized IoT/M2M platform, oneM2M, to integrate Kubernetes-based Fog Network and OpenStack-based Cloud Network. Moreover, we adopt dual scalability mechanisms in Fog and Cloud to achieve overall system scalability while accommodating the demands from low latency applications.

We compared our proposed system to the ones with only Fog or Cloud by utilizing a smart hospital use case. Three architectures are compared in terms of their latency, CPU usage and power consumption. We verify that the proposed system is the best choice for the specific use case.

In the future, we plan to apply SDN/NFV technology to our proposed system. The goal is to bring the benefits of softwarization and virtualization to system scalability such as to manage Cloud Node as VM-based VNFs and Fog Node as container-based VNFs in the NFV architecture.

Another extension in our plan is to add more layers of Fog Nodes. As more complicated IoT traffic comes in, it's necessary to distribute the task to more levels of Fog nodes and each level of Fog nodes can be equipped with scalability mechanisms to support the processing of the huge amount of incoming data requests.

Acknowledgement. This work was financially supported by the Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, pp. 13–16 (2012)
2. OpenFog Consortium: OpenFog Reference Architecture for Fog Computing (2017)
3. Smart Hospital. https://www.sdglobaltech.com/blog/how-developers-must-prepare-to-create-smart-hospital-solutions. Accessed 9 Apr 2019
4. Cerritos, E., Lin, F.J., De la Bastida, D.: High scalability for cloud-based IoT/M2M systems. In: IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia (2016)
5. De la Bastida, D., Lin, F.J.: OpenStack-based highly scalable IoT/M2M platforms. In: iThings, Exeter, England, UK, June 2017
6. Sen, S., Balasubramanian, A.: A highly resilient and scalable broker architecture for IoT applications. In: 10th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India (2018)
7. Tseng, C.L., Lin F.J.: Extending scalability of IoT/M2M platforms with fog computing. In: IEEE World Forum on IoT, Singapore (2018)
8. Zhao, W., Liu, J., Guo, H., Hara, T.: ETC-IoT: edge-node-assisted transmitting for the cloud-centric internet of things. IEEE Netw. **32**, 101–107 (2018)

9. Chen, Y.C., Chang, Y.C., Chen, C.H., Lin, Y.S., Chen, J.L., Chang, Y.Y.: Cloud-fog computing for information-centric Internet-of-Things applications. In: 2017 International Conference on Applied System Innovation (ICASI) (2017)
10. oneM2M: Functional Architecture, oneM2M Technical SpecificationTS-0001-V.3.12.0
11. Kubernetes. https://kubernetes.io/. Accessed 9 Apr 2019
12. Metrics Server. https://github.com/kubernetes-incubator/metrics-server. Accessed 9 Apr 2019
13. RabbitMQ. https://www.rabbitmq.com/. Accessed 9 Apr 2019
14. Openstack. https://www.openstack.org. Accessed 9 Apr 2019
15. Docker. https://www.docker.com/. Accessed 9 Apr 2019
16. OM2M. https://www.eclipse.org/om2m/. Accessed 9 Apr 2019
17. Heartbeat Data. https://archive.ics.uci.edu/ml/datasets/Heart+Disease. Accessed 24 Apr 2019
18. Video Stream. https://schema.org/VideoObject. Accessed 25 Apr 2019
19. Personal and Clinic Historical Information. https://schema.org/Patient. Accessed 25 Apr 2019
20. UserBenchmark. https://cpu.userbenchmark.com/. Accessed 9 Apr 2019
21. Power Supply Calculator: https://outervision.com/power-supply-calculator. Accessed 9 Apr 2019