International Journal on Advanced Science Engineering Information Technology

Vol.11 (2021) No. 6
ISSN: 2088-5334

Segmented Network Architecture for Promoting High Availability in Fog Computing through Middleware

Mohd Hariz Naim [a,*], Jasni Mohamad Zain [b], Kamarularifin Abd Jalil [b], Lizawati Salahuddin [a]

[a] *Centre of Advanced Computing Technology C-ACT, Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka, 761000, Durian Tunggal, Melaka, Malaysia*
[b] *Department of Computer Technology and Network (CTN), Faculty of Computer Science and Mathematical, Universiti Teknologi MARA, 40450, Shah Alam, Selangor, Malaysia*

*Corresponding author: *mohdhariz@utem.edu.my*

***Abstract*— This paper proposes an architecture for deploying applications on a fog computing environment by adding another layer of fog nodes in a network segment that gains high software application availability. The conventional fog computing architecture would permanently shift the storage, applications, and data from cloud servers to fog nodes, thus reducing the dependency on the cloud. As a result, fog nodes are burdened with the task previously done by cloud servers and have become "mini cloud servers." Instead of permanently shifting the tasks from cloud servers to fog nodes, the proposed architecture would only do the shifting, when necessary, like if an internet outage. Additionally, this research also introduced the middleware application that acts as a detector and replacement if service outage so that the availability of the services is not interrupted, especially during the internet outage, by adding another layer of fog node in a network segment. The computational process occurs between end-users and the fog nodes without having to rely on cloud servers. An experiment was conducted to test the performance of the proposed architecture. From the experiment, it can be concluded that the deployment of fog nodes in a segmented network is possible and able to increase the availability of data and services if an internet outage.**

***Keywords*— Fog computing; high availability; deployment architecture; middleware.**

Manuscript received 23 Oct. 2020; revised 11 Mar. 2021; accepted 18 Apr. 2021. Date of publication 31 Dec. 2021.

I. INTRODUCTION

The birth of the internet and its wide usage has given birth to the cloud and mobile computing, where applications reside on cloud providers such as Google Cloud Platform, Microsoft Azure, and Amazon Web Service. These cloud service providers have upgraded their infrastructure to produce more scalable and robust cloud computing to meet the demand with better availability and the increasing number of data centers [1]. The resources constraint faced by end-user devices ranging from IoT devices, computers, and physical servers has made cloud computing the providers of a preferred and flexible resource in deploying the solution [2], giving them an advantage on resources to focus on developing quality and fewer bugs software applications [3] before delivering to the end-users. Previously, in the early days of computing, software was installed and run separately and independently [4] in every machine. This type of application is commonly known as desktop or terminal applications such as Microsoft Office or Telnet, requiring individual installations inside a machine. As data is kept separately, it leads to unsynchronized and inaccurate information [4]. For example, in the banking sector, a customer's information may reside on many computers as the customer could be dealing with multiple times by different counters. As one solution, the unsynchronized information problem has led to the birth of a centralized database where information could be stored in one single machine, and all other machines could retrieve or send the data via socket connections implemented in the applications [5]. However, each application would need to be updated individually as a different version to implement this method. Some of these applications also become obsolete, and new applications must be created again from scratch as updating the applications seems futile and costly [4], [6].

As modern technology emerges, so make software applications. In order to keep up with the pace of technology, existing software would need to be revamped from a desktop application to web-based applications that can promote centralized information [7] and faster application update. The

update would just need to be done on the server instead of each machine that would take more time and human resources. The server would be installed with the application, and it only requires internet for connecting between machines. The server would live either on the cloud or at the edge of network architecture within a premise.

According to Mesbahi *et al.* [8], cloud computing is a type of computer paradigm consisting of massive physical machines working together that use computing utility for providing high reliability of software solutions. Cloud computing is categorized into three services, namely software as a Service(SaaS), Platform as a Service(PaaS), and Infrastructure as a Service(IaaS) [9]. With the advancement of the internet and the increasing number of applications and users, cloud computing as a Platform as a Service and Infrastructure as a Service has become popular deployment technique. With cloud infrastructure, users of all levels may benefit from the scalability and high availability, assuring less degradation in terms of performance [10]. However, it does not cover the availability on the edge of the network, and end-users may still be prone to service interruption whenever the internet is unavailable.

On the other hand, fog computing or edge computing is another computer paradigm concept where it does not rely on the cloud for computational processing tasks [11]–[13]. Instead, it runs the logical and computational process on the lower layer of the network closer to the end users' machine. One of the prominent usages of fog computing is the health sector [14], [15] and home automation [16], where there are devices such as heartbeat sensors to read the heartbeat of a patient and infrared sensors to detect movement. The sensors may have their computational process from the data obtained to send alerts or notifications to the corresponding person in charge, such as medical practitioners or law enforcers. To increase high availability, this study proposes using IoT device from a Raspberry Pi that acts as a middleware residing in a network segment as part of a fog computing environment.

In this paper, section 1 discusses the related works, including the architecture of cloud computing related to infrastructure as a service and platform as a service. Some issues are also discussed with cloud computing, followed by a fog computing architecture and how the computer nodes are segregated according to the certain subnet. In addition, the review of related deployment architecture is explained in this section. Section 2 then explains the material and method that describes the proposed software topology that is ideal and suitable to be deployed in a fog computing environment, followed by experimentation to explain how the simulation is run. The discussions on the result are presented in section 3. Lastly, section 4 discusses conclusions that provide motivations for providing high availability in data and software services.

A. Cloud Computing Architecture

Due to the increasing number of software applications, including back end services such as web services and socket connections, many software providers are looking for solutions that can deploy their products without having to consider specifications of physical servers and in addition to flexible resources that can expand when required [8]. This has promoted cloud computing as one of the solutions in favor as the cloud as a service may give software providers a cost-effective [17] and more peace of mind when deploying their products. Cloud environments may differ according to how the software providers would choose to deploy their solution to which cloud providers. The architecture, in general, can be visualized in Figure 1.

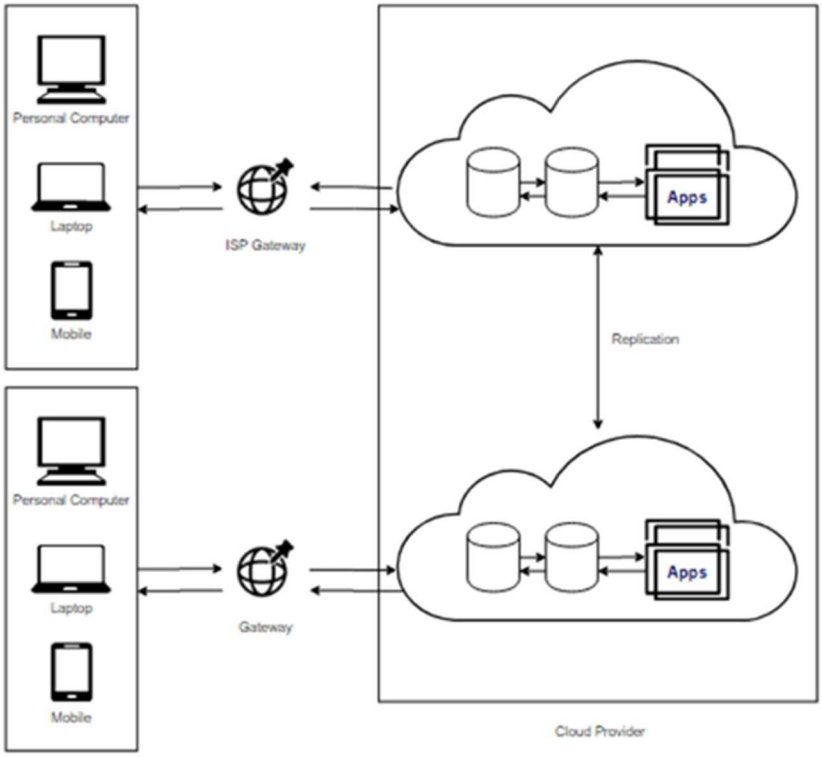

Fig. 1 The General Architecture of Cloud Architecture via Internet Backbone

The architecture in Figure 1 requires the internet as the backbone of the connection. End nodes or users may connect to the cloud by any connection such as High-Speed Downlink Packet Access (HSDPA) or common Wi-Fi or Ethernet that is linked to the Internet Service Provider. In order to prevent connection failures and provide high availability, most cloud providers have implemented a failure-recovery plan through mirroring and replication [18], where the replicas are geographically located far away from each other. Thus, in case of mishaps in the server farm, such as earthquakes or floods, the services and data may still be safe in the other replicas.

Zooming into the deployment layer may differ according to how software providers may deploy their applications. The typical application layer may interact directly with the cloud, as depicted in Figure 2. On the end nodes, the users' applications and sensors provide data and send it to the central cloud server [8], and there is no computational process at this layer. Only necessary logical processes such as authentication and validation, which is called front-end processing, may happen at this layer [7], [19] depending on functionalities and the type of the applications. This proves that it could reduce the burden at the client side [20] as most processes are done at the cloud layer. The database storages reside in the cloud layer, and the computational process is called the back-end process.

The research done by Mesbahi *et al.* [8] has proposed using OpenStack architecture to promote a high and reliable cloud data center by implementing an 80/20 rule. The OpenStack is an open-source software platform for cloud deployment which provides infrastructure-as-a-service (IaaS) where resources are placed on virtual servers and made available to the end-users. The OpenStack is also considered a cloud management system, having the capabilities of dynamically scaling the resources according to many tasks assigned. Different computers, including physical and virtual machines, maybe clustered to form a single cloud datacenter through OpenStack. The authors also combined several components involving the Nova tool, Failure Analyzer, Heat and Glance tool to compose the proposed architecture. Those afore mention tools are available as part of components in the OpenStack controller.

The authors introduced a hypothesis that 80% of failure tasks are derived from 20% of machine failure. Thus, the Nova can divide each cluster into less risky sub-cluster according to the machine failure record. These records are generated by the failure analyzer that monitors each machine's behavior and event seamlessly. Although the authors claim that the subclustered machine has a lesser failure rate and can provide 99.999% of availability, the proposed architecture is too dependent on the failure records, which could be incorrect to a certain machine that has received modification or part replacement. This would also lead to unutilized machine resources as only the less risky subclustered is assigned with the tasks while another machine could be idle or less assigned. The research done by Acquaviva *et al.* [21] integrates three different cloud providers to ensure the high availability and performance of the cloud by implementing replication mechanisms to replicate services among cloud providers. The middleware that is the main component for managing virtual machine (VM) replicas has implemented an algorithm to optimize the resource optimization among VM. However, the proposed solution is to a constraint on middleware that is too app-centric where any application that does not follow the framework nor implement the algorithm could not improve the availability of the application. Furthermore, the middleware residing on the cloud would make the technique pointless whenever the connection is interrupted or unavailable.

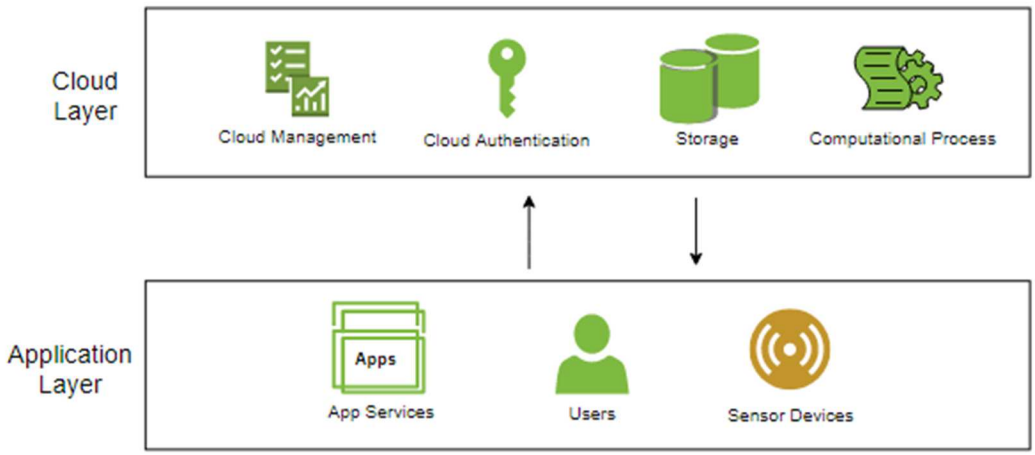

Fig. 2 Applications and Sensors devices at the same layer with Users

B. Fog Computing Architecture

In contrast to cloud computing, additional nodes known as fog nodes may be required where the back-end process is done within the network segments, as shown in Figure 3. Instead of sending data and depending on the cloud for a computational process that proves to be more latency and consuming resources [11], [22], the computational process is done at the end of the network done by fog nodes. Here, other nodes would send the data to the fog nodes for the computational process which would not require the internet backbone [12]. The fog nodes would be able to respond to the pre-decision-making back to the nodes [16]. For example, whenever a smoke sensor detects smoke and sends the data to the fog node, the fog node would straight away send a notification to the authorities. Only important information after the data has

been processed, such as reporting information to be saved, would be sent to the cloud's central server for synchronization and further decision making.

In terms of database storage, the fog nodes would be installed with the same database as the cloud server has, and it has performed almost at par as usual central server [5] though it could be upgraded or downgraded to meet its functional purposes. In addition, the fog nodes can be distributed and scattered around the intranet for better accessibility. The cost for fog nodes may vary according to the types of hardware specifications that it has and but any existing nodes could be turned to become fog nodes [22], [23] which give advantage in term of reusability of existing devices available on the network.

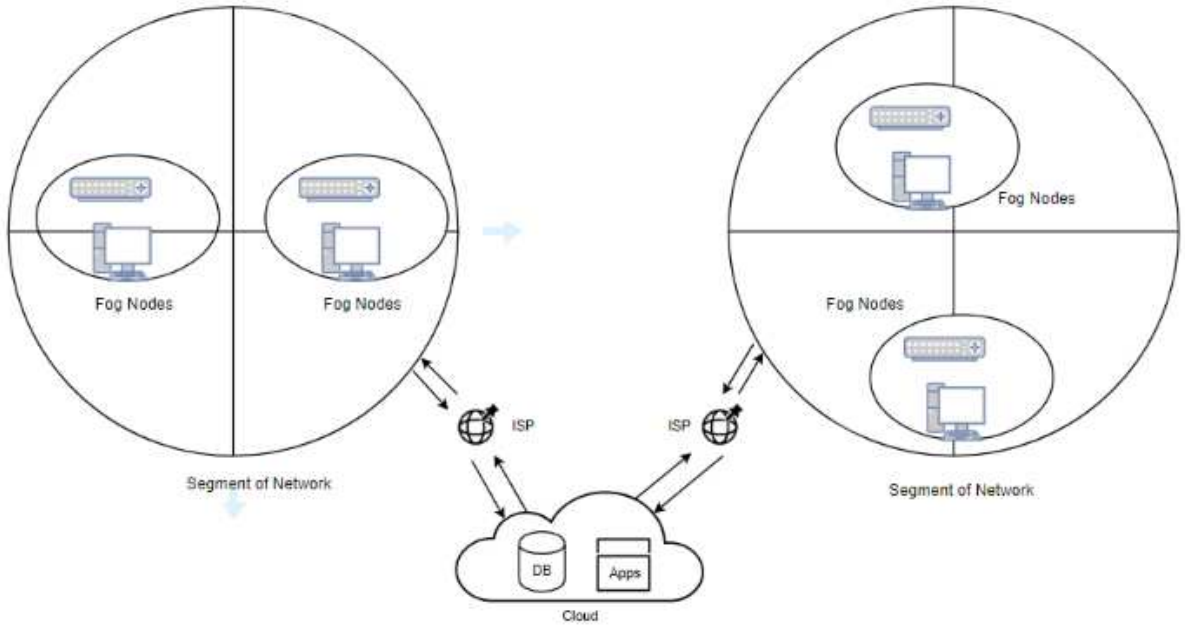

Fig. 3 The architecture of fog computing connected to the cloud via Internet Service Provider (ISP)

According to Hu *et al.* [12], fog computing architecture is composed of 3 layers structure:

- Cloud layer. This layer positions on the topmost that resides on the internet, which consists of many high-performance machines and enormous storage capacity for storing gigantic information. This layer also performs a heavy computational process such as decision-making and data analysis gained from the lower layer.
- Fog layer. The layer comprises gateways, switchers, routers, network access points, and certain computer machines that are converted to become mini servers for achieving specific computational processes.
- Terminal layer. This layer is the closest to the physical environments and end-users, generating the raw data from various devices scattered around the network, such as heat sensors, smoke sensors, mobile devices, and input readers. The data is then transmitted to the fog layers for pre-processing.

As millions of fog devices, including the Internet of Things (IoT) scattered in certain networks, can produce zettabytes of information. Processing the massive information and looking up important data would consume high resources, which would mean a challenge for fog computing. The research done by Tortonesi *et al.* [24] proposed a model for fog computing to filter out massive information called the Sieve, Process, and Forward (SPF). The authors claimed that the model is capable of reusing the processing components to be used by numerous different services and this would certainly optimize resources of both fog layer and cloud layer.

Even though having the fog computing layer running on an edge network seems promising, it still has some flaws, especially in power usage [25] where it must depend on battery capacity to continue operating. Depending on the battery's operation has caused the fog devices to have limited operational life and restricted processing power to conserve energy. Additionally, the challenges with resources capabilities such as memory and storage would also restrain the fog devices from operating at the same scale as a cloud server. Thus, deployment on the fog level would need meticulous consideration and careful set up to ensure the performance is optimized to avoid application service downtime due to software aging and operating system degradation.

C. High Availability of Application Services

According to the definition by Bahn [25], data and service availability can be defined as the level of a system that can operate, accessible when obliged, and the data should be confidentially protected from any attacks or breached [26]. Cloud providers have to incorporate high availability as part of the Service Level Agreement(SLA) towards their client [27] so that in the event of any interruptions, their client may still be able to access the service. As defined by Mesbahi et al. [8], the SLA is that the availability of services should be 99.9999% or close to 100%. Although this seems theoretically achievable, in real-life situations, the percentage is almost impossible to achieve as there could be unexpected disaster occurrences such as network attacks or power failures.

Nabi *et al.* [28] stated that the availability technique, in general, is categorized as protective redundancy, fault tolerance, and overload protection, as visualized in Figure 4. In order to maintain and balance the performance when a server is straining with tasks and processing requests, the overload protection is implemented where some tasks and requests are delegated to another server that has fewer tasks. The cloud providers may embrace the high availability service, which may be achieved through replication and switchover [28] of the server as part of protective redundancy and fault tolerance where cloud providers usually position the data centers in the geographically distributed location around the world [29]. The replication of the server and the use of middleware as fault tolerance mechanisms have enabled the computer system to become better in terms of availability. The middleware is considered part of the components that can be removed, replaced, and even modified when failure has occurred [30] to function as normal.

Having replication in multiple server farms would mean that data is cloned in every replicated server, and thus, it would mean the storage may increase [28], causing problems in terms servers' resources consumption, including memory, CPU, and bandwidth. Each time a node is making a request, every replicated server consume their resources and in the event of mass request from many nodes, this would have a disastrous effect on performance.

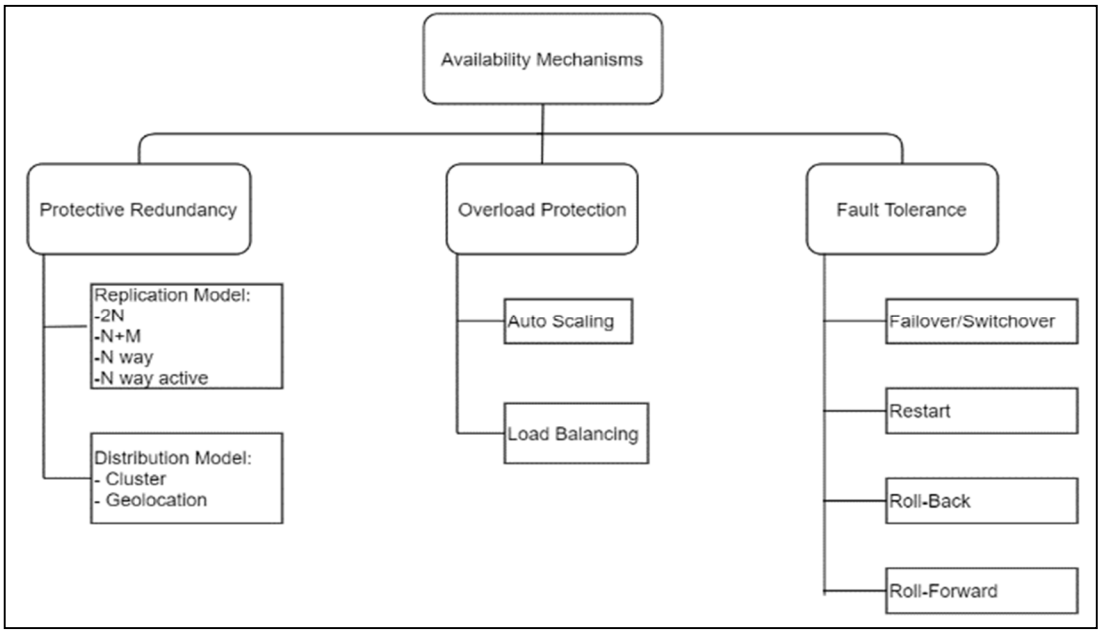

Fig. 4 The availability mechanism, in general, comprising protective redundancy, overload protection, and fault tolerance

As for cloud computing, multiple virtual machines and integrated data centers would benefit from auto scaling and load balancer [31], where the traffic is redirected to which cloud node has less resource load. The load balancer is placed and configured at the entry point of the cloud networks, where it keeps track of the resources consumed by each replicated server and navigates the traffic evenly. Despite the existence of a load balancer, delegated tasks and requests may still suffer from latency [32], [33] as the replicated server could be geographically far from end-users. Another research has optimized task migration by considering user mobility in order to reduce latency[34]. The authors propose an algorithm for evaluating group migration for mobile edge computing before the predicted mobile device is assigned with the task. However, the research does not consider the congestion and applications running in the edge network that eventually contribute to the latency as the network bottleneck has been reached.

II. Material and Method

Taking the concept and advantages of fog computing, an enhancement is proposed by adding another node deployed to the current network. An organization's network was segmented into several sections based on following criteria, the number of nodes, and the location of the designated intranet. The number of nodes would affect the network congestion, and thus, the number of nodes is capped at a maximum of 20 fog devices. The location of the designated intranet would depend on the geographical and physical devices available on the network where the fog devices are grouped according to departments for ensuring better group identification. The additional nodes, which are the fog nodes, were residing in every network segment as depicts in Figure 5.

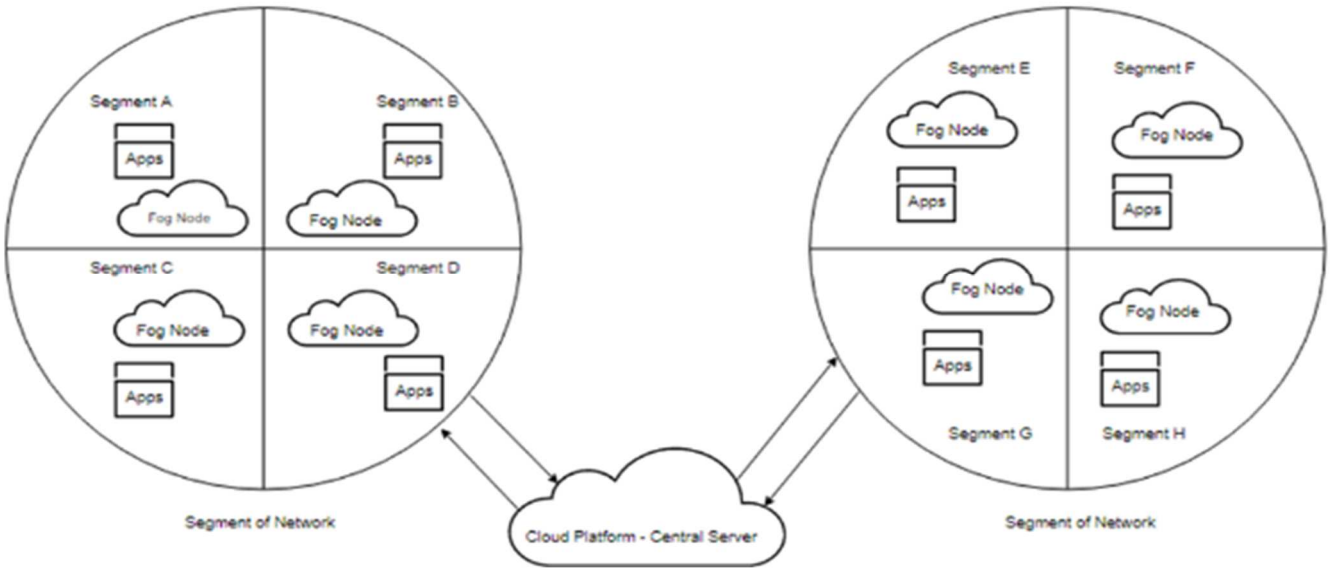

Fig. 5 The proposed deployment of fog nodes where the node lives in every segment of a network

Each of the fog nodes or the middleware performs network checking to the central cloud server and the fog nodes are identical to the central servers residing in the cloud, which explains the purpose of having database and Apache service to mimic the services provided by the central servers.

The experiment is done in an organization having several network segments that are divided by subnet. For the sake of experimentation, we only take one segment of a network that is deployed with the fog node. In the network segment, other nodes function as end users using desktop applications interacting with the central cloud servers via the internet, as visualized in Figure 6, which shows the proposed application layer.

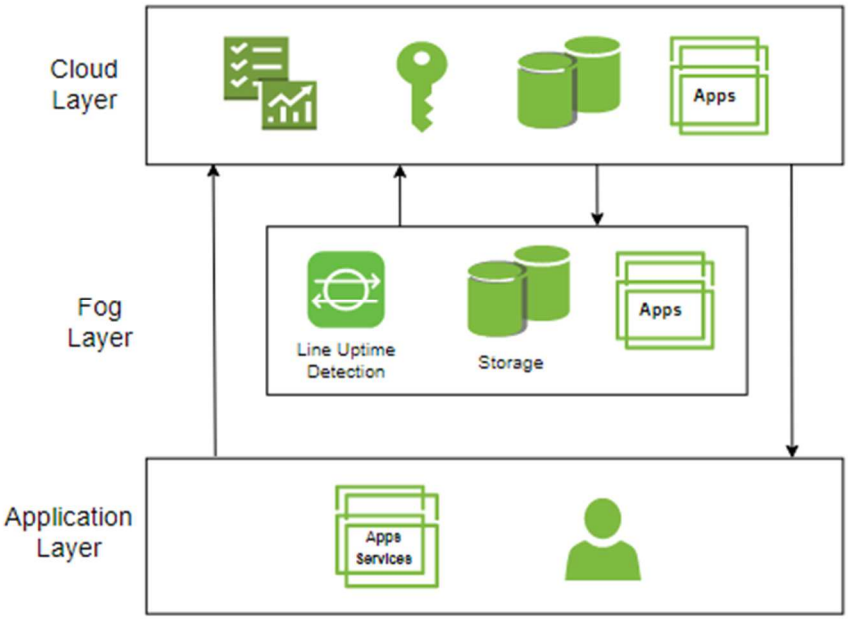

Fig. 6 The proposed fog layer located between application layer and cloud layer

At first, the applications send and receive data from the central cloud as usual cloud-based computing. At the same time, the fog node receives data updates from the central server by the mirroring technique. The Internet Control Message Protocol (ICMP) ping service is developed in Python to detect network interruption between the network segment and the central cloud server. In comparison, the cron script is created to perform the Python script in a timely manner for every certain interval of 3 seconds.

If internet line breakdown, where the end-users cannot access the data and service from the central server, the fog node performs a computation process by detecting it and quickly replacing the network address of the central server. Here, the end-users are automatically redirected to the fog node that acts as a secondary server. The computational process now occurs between end-users and fog nodes without relying on a central cloud server. All the data is saved into the fog nodes database storage, and once the internet connectivity

has been recovered. All the data is synchronized between fog nodes with the central servers.

A. Middleware as Computational Process

Here, a fog device has been appointed as the middleware that can detect and replace the main server outage. The detection follows the heartbeat concept [18], [35], where the fog device keeps listening for the main server through ICMP. The chart visualizes the detection process in Figure 7.

First, the middleware that is the Raspberry Pi is installed with the Cron script scheduler. Similarly, Python script is the main engine for the computation process that performs ICMP ping checks. The scheduler executes the python script, and the process continues by waiting to respond from the main server. If there is a response from the main server, the middleware fetch for any update from the main server and keep on listening for the response from the main server. If an outage, the middleware is not getting any response from the main server, and thus, the middleware performs another process to change its address and swapping it to become the main server. This way, applications refer to the middleware as if it is the main server. Meanwhile, the middleware keeps on listening to the main server by keeping on ping. Once the main server has recovered, the middleware sense it and change it back to the original address. The applications are now redirected back to the main server, and the middleware synchronizes back any update that the middleware has been getting during a server outage.

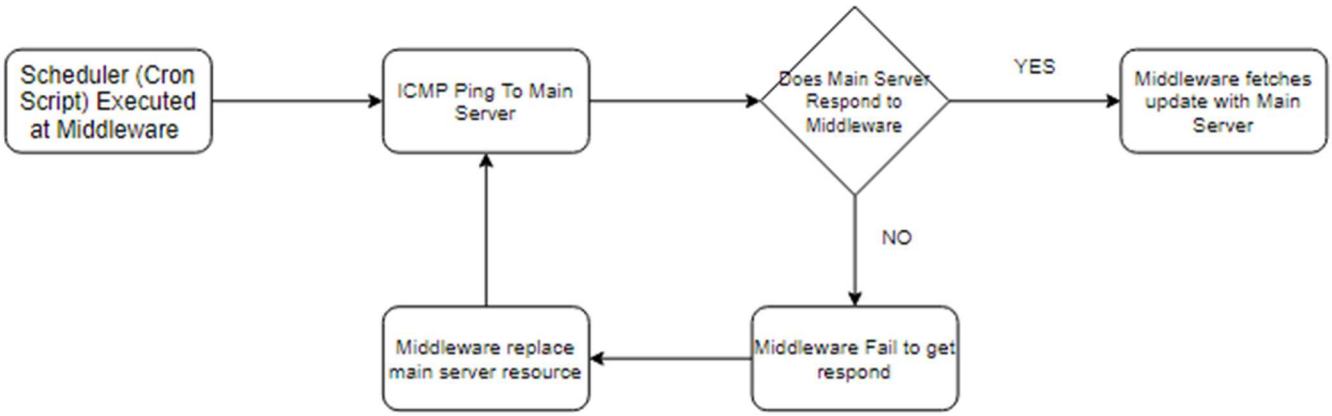

Fig. 7 The detection of failure process executed by the fog node

The detailed experimentation involves two phases involving detection of fog node failure and replication of services between server and fog node. Then, the last process is migrating tasks from the main server to and from the fog node. For simulating the experimentation, only one segment of the network has been deployed with the fog node, where it acts as a middleware for performing all the two phases. The mail server's address (M^S) would need to be pointed by the middleware for listening to the heartbeat by the fog node (F^N).

B. Failure Detection and Replication Algorithm

The logical algorithm for detecting outage occurrences follows Algorithm1, where the input is the main server address that needs to be listened to using ICMP that acts as a heartbeat listener. From the ICMP response, the output produces a response from the main server, which further performs another process of updating the database from the main server to the fog node database according to the binary log file that acts as a flag indicator whenever there are data changes that need to be updated. The synchronization and replication of data would occur when the method fnFetchData is invoked

Algorithm 1:
Input: Address of Selected Main Server (M^S)
Output: Main Server (M^S) Acknowledgement Respond
/* Process for listening to the main server uptime */
respond_receive <- icmp event
while (respond_receive) do
if nodeCurrBinaryLog != main_server_binaryLog then
fnFetchData()
end
if (respond_receive == 0) then
fnChangeAddress()
Function fnFetchData
nodeCurrBinaryLog <- main_server_data_log
nodeDatabase.update(main_server_data)
end

C. Failover and Recovery

The failure and recovery process would perform swapping of address from its original IP Address to become similar to the main server IP address and this hand over the task from the main server to the middleware node.

Algorithm 2:
Input: Local Middleware address
Output: Address Swapping to Main Server
Function fnChangeAddress
nodeNetworkAdapterStat <- down
nodeCurrAddress <- main_server_address
nodeNetworkAdapaterStat <- up
application service startup
listen and lookup for main server second address
end

D. Experimentation Settings

The network address of the fog node and main server are reserved and are assigned permanently to avoid other devices from taking the address, leading to address conflict. The main server is allocated with two static addresses, where the second address is for the fog node to listen for the main server when connectivity has been recovered. Here the fog node and main server have the following address:

Fog Node: 192.168.2.20
Main Server Address 1: 192.168.1.2
Main Server Address 2: 192.168.1.20

The fog node which is a middleware function as a sensor for detecting any network interruption, and here we proposed the use of Raspberry Pi device that have certain service installed as follow:

Database service (MySQL)

- Apache service
- Internet Control Message Protocol (ICMP) service
- Cron script

The MySQL database used is for simulating the real database that resides in the central cloud server whereas the Apache service is used to run web-based applications and listen on the used port 80 as part of the detection technique. The ICMP ping service is used to obtain the status of the targeted host by sending live messages by waiting for a response from the targeted cloud server. By replying to the ping message, the middleware knows that the server is accessible. The cron script, on the other hand, is the scheduler to execute the ICMP ping service based on a certain predefine interval time which we have set to every 5 seconds

Initially, all application requests would point to the main server, and when the middleware detects an outage, the fog node detects and swaps its address to be similar to the main server. All requests automatically point to the fog node by broadcasting the newly changed address instead of looking for the main server. At the same time, the fog node still lookup the main server's second address.

III. Results and Discussion

The result is presented according to the description and steps to simulate real network outage occurrences. The network downtime is simulated by disabling and turning off network adapters at the main switch, the gateway to the main server. The adapters are enabled back to imitate that the network has been recovered normally.

TABLE I
EXPERIMENT BASED ON THE PROPOSED FOG ARCHITECTURE

Experiment Description	Steps	Result
Deploy Raspberry Pi as fog node/middleware to a network segment	Connect it with Wi-Fi within the designated network	Success. The Raspberry Pi now had become one of the nodes
Connect and synchronize fog node to replicate with central cloud server	The Raspberry is installed with the 4 services	Success. The node is now a fog node
Disconnect/Disable internet on the designated network segment	The internet is disabled by disconnecting the main switch from the internet	Success. The designated network no longer has internet connectivity
Detecting downtime by the middleware via ICMP heartbeat technique	ICMP messages are sent continuously to the central server	Success. Downtime detected by the middleware
Switching and masking	Execute python script to change the	Success. The middleware has
middleware to become a service provider	address similar to a central server	become the service provider
Ensure end-user applications are able to run	Open and execute the desktop application on the end user as usual	Success. The desktop application able to retrieve and save information
Connect back internet on the designated network segment	The main switch on the designated network is to reconnect	Success
Ensure end-user application able to run	Continue running the application as usual	Success. The applications able to execute

The experiment on the designated network has been performed as in Table 1 with success which shows that if network interruption, the service and data is still available with the implementation of fog nodes as middleware for performing the computational process at the edge of the network without having to rely on a central cloud server. The fog node can replace the functionality of a server even though there is interval time for detection and failover recovery. It has been recorded that the interval time taken for failover detection is around 5 seconds, whereas the recovery of service by the fog node is around 3 seconds and would take around another 5 seconds to return the service back to the central server. The experiment was run three times, and the result is presented as shown in Table II.

TABLE II
TIME TAKEN FOR THE DETECTION AND RECOVERY

No of Run	Detection Time(s)	Recovery Time(s)	Middleware Respond Time(s)
First	4.5	2.3	2.7
Second	4.7	2.4	3.1
Third	4.4	2.4	3.0

IV. Conclusion

From the experiment that has been done, it can be concluded that deploying a fog node at the edge of the network is possible. Also, it is possible to increase the availability of data and services by creating another layer of fog that resides between the cloud and end users. In the event of network interruption, applications communicate to the fog layer without experiencing network error. However, the scope of this research does not take response time and network bandwidth usage into consideration. In addition, the scope of the application is limited to desktop applications only.

The proposed architecture will be tested on other segments in the future, and the performance in terms of response time and bandwidth utilization is considered. In addition, the technique is enhanced to support the high availability of Domain Name Service (DNS) required for web applications.

Acknowledgment

This research is supported and funded by UTeM through a short-term grant with references PJP/2019/FTMK(B)/S01681. The authors are grateful to the Center of Advanced Computing Technology (C-ACT) Faculty of Information and Communication Technology, Universiti Teknikal Malaysia

Melaka, and Center of Computing Technology and Network (CTN) Faculty of Computer Science and Mathematical Universiti Teknologi Mara, Malaysia, for offering facilities in term of research provision.

REFERENCES

[1] P. Alves, L. Antônio, S. Barreto, and N. Paulo, "Data centers ' services restoration based on the decision-making of distributed agents," *Telecommun. Syst.*, 2020, doi: 10.1007/s11235-020-00660-2.

[2] Y. Jin and H. J. Lee, "On-demand computation offloading architecture in fog networks," *Electron.*, vol. 8, no. 10, 2019, doi: 10.3390/electronics8101076.

[3] E. Hassan, Z. M. Yusof, and K. Ahmad, "Factors Affecting Information Quality in the Malaysian Public Sector," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 9, no. 1, pp. 32–38, 2019, doi: 10.18517/ijaseit.9.1.6385.

[4] J. Eloff and M. Bihina Bella, *Software Failure Investigation A Near-Miss Analysis Approach*, 1st ed. Cham, Switzerland: Springer International Publishing, 2018.

[5] J. Liu, F. Liu, X. Li, K. He, Y. Ma, and J. Wang, "Web Service Clustering Using Relational Database Approach," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 25, no. 8, pp. 1365–1393, 2015, doi: 10.1142/S021819401550028X.

[6] J. Rahme and H. Xu, "A software reliability model for cloud-based software rejuvenation using dynamic fault trees," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 25, no. 9–10, pp. 1491–1513, 2015, doi: 10.1142/S021819401540029X.

[7] M. H. Naim, M. K. A. Ghani, A. S. H. Basari, B. Aboobaider, L. Salahuddin, and W. N. A. Rashid, "Synchronization technique via raspbery Pi as middleware for hospital information system," *Adv. Intell. Syst. Comput.*, vol. 734, pp. 262–271, 2018, doi: 10.1007/978-3-319-76351-4_27.

[8] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh, "Highly reliable architecture using the 80/20 rule in cloud computing datacenters," *Futur. Gener. Comput. Syst.*, vol. 77, pp. 77–86, 2017, doi: 10.1016/j.future.2017.06.011.

[9] K. Syed and K. Vijaya, "Cloud Computing: Review on Recent Research Progress and Issues," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 3, pp. 959–962, 2019, doi: 10.30534/ijatcse/2019/96832019.

[10] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh, "Reliability and high availability in cloud computing environments: a reference roadmap," *Human-centric Comput. Inf. Sci.*, vol. 8, no. 1, 2018, doi: 10.1186/s13673-018-0143-8.

[11] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara, "A review of edge computing reference architectures and a new global edge proposal," *Futur. Gener. Comput. Syst.*, vol. 99, no. 2019, pp. 278–294, 2019, doi: 10.1016/j.future.2019.04.016.

[12] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, no. September, pp. 27–42, 2017, doi: 10.1016/j.jnca.2017.09.002.

[13] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Comput. Networks*, vol. 130, no. 2018, pp. 94–120, 2018, doi: 10.1016/j.comnet.2017.10.002.

[14] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog Computing in Healthcare-A Review and Discussion," *IEEE Access*, vol. 5, no. 2169, pp. 9206–9222, 2017, doi: 10.1109/ACCESS.2017.2704100.

[15] Y. Ren, R. Pazzi, and a Boukerche, "Monitoring patients via a secure and mobile healthcare system," *Wirel. Commun. IEEE*, vol. 17, no. February, pp. 59–65, 2010, doi: 10.1109/MWC.2010.5416351.

[16] A. M. Rahmani *et al.*, "Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach," *Futur. Gener. Comput. Syst.*, vol. 78, pp. 641–658, 2018, doi: 10.1016/j.future.2017.02.014.

[17] B. Snyder, J. Ringenberg, R. Green, V. Devabhaktuni, and M. Alam, "Evaluation and design of highly reliable and highly utilized cloud computing systems," *J. Cloud Comput.*, vol. 4, no. 1, 2015, doi: 10.1186/s13677-015-0036-6.

[18] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Orchestrating the deployment of high availability services on multi-zone and multi-cloud scenarios," *J. Grid Comput.*, vol. 16, no. 1, pp. 39–53, 2017, doi: 10.1007/s10723-017-9417-z.

[19] H. Shahzad, X. Li, and M. Irfan, "Review of data replication techniques for mobile computing environment," *Res. J. Appl. Sci. Eng. Technol.*, vol. 6, no. 9, pp. 1639–1648, 2013, doi: 10.19026/rjaset.6.3883.

[20] R. K. Lomotey, S. Jamal, and R. Deters, "SOPHRA : A Mobile Web Services Hosting Infrastructure in mHealth," *2012 IEEE First Int. Conf. Mob. Serv.*, pp. 88–95, 2012, doi: 10.1109/MobServ.2012.14.

[21] L. Acquaviva *et al.*, "NoMISHAP: A Novel Middleware Support for High Availability in Multicloud PaaS," *IEEE Cloud Comput.*, vol. 4, no. 4, pp. 60–72, Jul. 2017, doi: 10.1109/MCC.2017.3791011.

[22] M. Singh and V. M. Srivastava, "Implementing architecture of fog computing for healthcare systems based on iot," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 4C, pp. 23–27, 2019.

[23] H. Zhang, Y. Xiao, S. Bu, D. Niyato, R. Yu, and Z. Han, "Fog computing in multi-tier data center networks: A hierarchical game approach," *2016 IEEE Int. Conf. Commun. ICC 2016*, pp. 1–6, 2016, doi: 10.1109/ICC.2016.7511146.

[24] M. Tortonesi, M. Govoni, A. Morelli, G. Riberto, C. Stefanelli, and N. Suri, "Taming the IoT data deluge: An innovative information-centric service model for fog computing applications," *Futur. Gener. Comput. Syst.*, vol. 93, pp. 888–902, 2019, doi: 10.1016/j.future.2018.06.009.

[25] C. Bahn, "IEEE Standard Computer Dictionary: IEEE Standard Computer Glossaries." .

[26] L. Cassandra *et al.*, "Adopting an ISO / IEC 27005 : 2011-based Risk Treatment Plan to Prevent Patients from Data Theft," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 3, pp. 914–919, 2020.

[27] Y. Tang, H. Sun, X. Wang, and X. Liu, "Achieving convergent causal consistency and high availability for cloud storage," *Futur. Gener. Comput. Syst.*, vol. 74, pp. 20–31, 2017, doi: 10.1016/j.future.2017.04.016.

[28] M. R.Kaseb, M. H. Khafaqy, I. A. Ali, and E. M.Saad, "An Improved Technique For Increasing Availability in Big Data Replication," *Futur. Gener. Comput. Syst.*, no. 91, pp. 493–505, 2019.

[29] P. Alves Lima, A. Sá Barreto Neto, and P. Romero Martins MacIel, "Data Centers Service Restoration Based on Distributed Agents Decision," *Proc. - 2018 IEEE Int. Conf. Syst. Man, Cybern. SMC 2018*, pp. 1611–1616, 2019, doi: 10.1109/SMC.2018.00279.

[30] M. Stoicescu, J. C. Fabre, and M. Roy, "Architecting resilient computing systems: A component-based approach for adaptive fault tolerance," *J. Syst. Archit.*, vol. 73, pp. 6–16, 2017, doi: 10.1016/j.sysarc.2016.12.005.

[31] M. Jammal, H. Hawilo, A. Kanso, and A. Shami, "Generic input template for cloud simulators: A case study of CloudSim," *Softw. - Pract. Exp.*, vol. 49, no. 5, pp. 720–747, 2019, doi: 10.1002/spe.2674.

[32] A. Alelaiwi, "An efficient method of computation offloading in an edge cloud platform," *J. Parallel Distrib. Comput.*, vol. 127, pp. 58–64, 2019, doi: 10.1016/j.jpdc.2019.01.003.

[33] Y. Aldwyan and R. O. Sinnott, "Latency-aware failover strategies for containerized web applications in distributed clouds Cloud Failover Techniques :," *Futur. Gener. Comput. Syst.*, vol. 101, pp. 1081–1095, 2019, doi: 10.1016/j.future.2019.07.032.

[34] F. Tang, C. Liu, K. Li, Z. Tang, and K. Li, "Task Migration Optimization for Guaranteeing Delay Deadline with Mobility Consideration in Mobile Edge Computing," *J. Syst. Archit.*, p. 101849, 2020, doi: 10.1016/j.sysarc.2020.101849.

[35] J. H. Lee and J. M. Gil, "Adaptive fault-tolerant scheduling strategies for mobile cloud computing," *J. Supercomput.*, vol. 75, no. 8, pp. 4472–4488, 2019, doi: 10.1007/s11227-019-02745-5.