Studying and developing a resource allocation algorithm in Fog computing

An Truong Pham Thanh, Nguyen Quang Hung(*), Nam Thoai

Bach Khoa University, Ho Chi Minh City, Vietnam

Ho Chi Minh City, Vietnam

Email: nqhung@hcmut.edu.vn

Abstract— **Cloud computing has become an utility service. In the cloud model, companies or users will hire hardwares, HPC units or storage instead of spending money on hardware investments. Charges may vary depending on usage and the Internet will serve as a bridge between the user and the service. Cloud has become so widespread that almost any business or Internet user is using their own Cloud. As a result, the amount of data moving will be extremely large and processing of Big Data will gradually appear. But to handle all that data will take time, and time is a pretty important thing for everyone today. So, there is a solution to this problem. And Cisco has pioneered the model instead of concentrating data like the old Cloud, which will be scattered, that model called Fog Computing. The main goal of the study is to choose a solution to distribute modules to the entire network instead of clustering like the old Cloud Computing. The algorithm must run successfully on the emulator environment. With different environments, the algorithm will allocate different modules depending on the configuration of each device. To test the algorithm, this study uses the iFogSim toolkit. The Fog Computing environment emulator extends from the CloudSim emulator. The current version is sufficient to be able to implement some criteria necessary for this topic.**

Keywords—fog computing, cloud computing, module placement, first fit algorithm.

I. INTRODUCTION

The IoT is slowly becoming more and more popular, and with a large number of devices it will bring in more and more data moving within the network infrastructure. Cloud servers will be responsible for resolving incoming data and sending them after processing. But sending them to the Cloud and returning them to the devices will go a long way. This can lead to delays and energy wastage will be quite high. Therefore, Fog Computing was born to help solve these problem for Cloud Computing. This paper introduces the Fog Computing architecture and the algorithm for resource placement to the Fog Computing system. Emphasize on the implementation and deployment of the simulator for Fog Computing testing.

But what exactly is an algorithm for resource placement to the Fog Computing system? As we knew, cloud server can run a HPC application. If we want to run that application in Fog Computing system, we'll need to split it into modules with smaller computing requirement. The algorithm will be responsible for placing these modules on the devices in the system to meet the proposed QoS (Quality of Service).

The main goal of the project is to choose a solution to distribute modules to the entire network instead of clustering like the old Cloud Computing. The algorithm must run successfully on the emulator environment. With different environments, the algorithm will allocate different modules depending on the configuration of each device.

The minimum requirement of the algorithm is to meet the criteria of Fog Computing, which is a matter of latency. In this paper, we will focus only latency criteria. According to the current Internet model, the "distance" or low bandwidth connection will cause high latency. And with Cloud Computing, the data that starts with the end devices has to go to the cloud and return back. The "interval" can be very remote and it is difficult to achieve real-time applications. Therefore, the algorithm must solve the problem of cloud computing latency.

To test the algorithm, this study uses the iFogSim toolkit[7]. The Fog Computing environment emulator extends from the CloudSim emulator. The current version is sufficient to be able to implement some criteria necessary for this topic.

II. RELATED WORK

Cloud Computing in IoT geared toward centralized data processing. In contrast, the Fog Computing model focuses on using the full computing power of network devices, storage and device control. The key to the success of the IoT project is to accurately select the appropriate computing model for the needs set.

Fog Computing is a term defined by Cisco to be geared toward extending cloud computing to the "edge" (the edge device-in the report will be called the Fog device) of the network. The Fog layer consists of controllers, gateways, switches and IO devices that provide computing, storage and connectivity.

As mentioned above, Fog Computing can be considered as an extension of Cloud Computing, extending it with certain benefits. Essentially, the development of Fog Computing gives businesses the choice of how the data is processed. For some applications, data needs to be processed as quickly as possible - for example, gaming will require a low response time. Fog Computing can create low-latency network connections between devices and final points for data analysis. This architecture in turn reduces the amount of bandwidth required for sending data to a data center or cloud and waiting for a response. It can also be used in situations where there is no connection bandwidth to send data, and that data will be processed immediately "near" where it is generated. Another benefit is that users can manually set

security features in the fog network, such as setting virtual firewalls to protect.

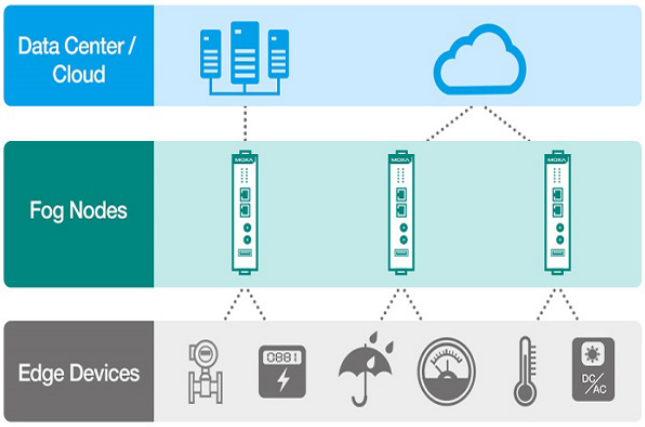


Figure 1 Network layer

A. The architecture

Fog Computing's criteria is to calculate the requests that are sent to the request processing unit. We will name it Fog Cell (which can be placed in the Fog Node). In addition, Fog Computing must allocate resources for each request to help keep the system as long as possible and consume as little resources as possible. Fog orchestration control will take care of that. And the two components are contained within the larger component, the Fog Colony. To understand simply, the fog colony is essentially a container of parent and devices connected to it. Sometimes Fog Device is not capable of executing a Request, so it will need to be sent to the Cloud for execution. And it will need a receiving and processing unit like Cloud-Fog Control Middleware (see figure below).

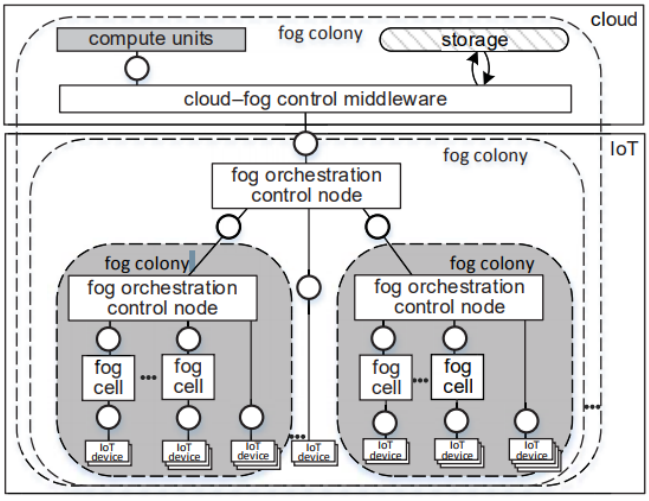


Figure 2 Fog Computing Architecture [10]

As mentioned above, the tissue model consists of modules and the relationship between them. The relationship here means that this module will send to the other module tuple and their tuple.

- *Cloud-Fog Control Middleware*

The cloud-fog control middleware is the central unit that manages the execution of task requests in the cloud, and supports the underlying fog landscape. The middleware performs cloud resource provisioning for task requests that are not delay-sensitive or cannot be executed in the fog, e.g., very resource-intensive Big Data analysis task requests. If necessary, the middleware performs global optimization of underlying fog colonies by restructuring them. For this, the cloud-fog control middleware is supplemented by both the means to control the cloud and the means to manage the underlying fog colonies. Such control is performed continuously or on-demand, depending on system events, e.g., if new fog devices appear, which could be used to deploy fog cells, or to recover after faults or damage of fog cells. Importantly, the cloud-fog control middleware can overrule fog orchestration control nodes in fog colonies, but the latter may also act autonomously in the case that no middleware is available.

- *Fog Cells*

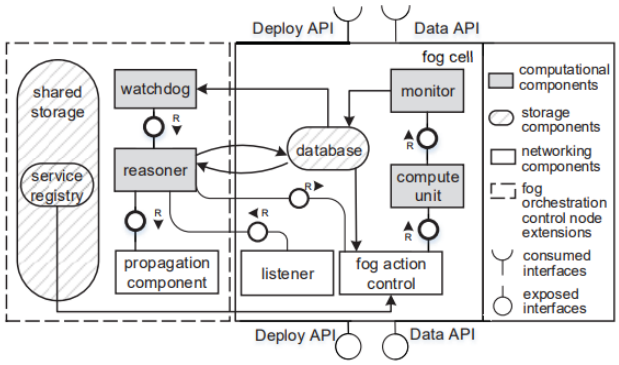


Figure 3 Fog Cell Components

Fog cells are software components running on fog devices. Fog cells serve as access points allowing the control and monitoring of the underlying IoT devices, e.g., sensor nodes. In order to do so, the fog cells are able to receive task requests, perform data analysis, allocate their own computational resources if available, or propagate task requests to upper layers of the fog colony hierarchy, i.e., to the fog orchestration control nodes (as discussed below). Cells may interact with an arbitrary number of IoT devices, however, in practice, the number of devices to be controlled by a fog cell is limited by the cell's computational resources.

Each fog cell consists of the following components (Figure 3): The listener receives task requests from other fog cells or IoT devices. The monitor observes service executions in the compute unit. The database stores data about received task requests, the current system state of the fog cell, i.e., available computational and storage resources, and monitoring data. The fog action control performs actions according to the provisioning plan produced by the fog orchestration control node, e.g., to deploy and start a particular service. Fog cells access the distributed storage to share data, e.g., service implementations, in a colony. This component allows faster data access compared to data storage in the cloud. The compute unit provides the actual computational resources for the deployment and execution of services.

Fog cells expose REST APIs for data transfer and control actions: The Data API allows basic CRUD operations over the data stored within a fog cell, and the Deploy API allows performing control actions for the services running in the fog cell, i.e., instantiating, deploying, starting, stopping, undeploying, and deleting the service. To become part of a fog colony, a fog cell needs to use the Data and Deploy APIs of the corresponding fog orchestration control node, and at the same time expose its own Data API and Deploy API.

- *Fog Orchestration Control Nodes*

A fog orchestration control node's main task is to support a fog colony by orchestrating the involved fog cells. Each fog colony features exactly one head fog orchestration control node. The fog orchestration control node is itself a (powerful) fog cell, which manages the resources offered by subordinated fog cells and performs resource provisioning to execute task requests. Also, the control node is able to propagate task requests to the cloud-fog control middleware or to other fog colonies (via their fog orchestration control nodes), if task requests cannot be handled by the current fog colony. For this, a resource management mechanism for vertical scalability is necessary to identify how task requests can be delegated in the entire fog landscape. Apart from resource provisioning, fog orchestration control nodes (i) perform infrastructural changes in the fog colony, (ii) analyze resource utilization within the colony, (iii) create a provisioning plan to allocate resources for task requests, and (iv) monitor IoT devices and fog cells.

As previously mentioned, fog orchestration control nodes are extended fog cells. On the left-hand side of Figure 3, the extensions needed for control nodes are depicted. The reasoner component produces a provisioning plan for the colony's resources to execute received task requests. The provisioning plan determines which services should be used to fulfill task requests, and where these services should be deployed, i.e., what fog cell to use.

The reasoner also gets the information about the system state, i.e., available fog colony resources, and controls the connected fog cells, i.e., plans infrastructural changes in the fog colony, if necessary. The database additionally stores the resource provisioning plan produced by the reasoner. After the reasoner produces a provisioning plan, the fog action control performs the orchestration of fog cells according to the plan. If there are no sufficient resources in the considered fog cell, or further processing is needed, such task requests are separated and propagated to the other fog colonies by the propagation component via the control node. The watchdog component features means to receive up-to-date information about the utilization of the connected fog cells. It observes monitoring data in the database and compares it to the expected Quality of Service (QoS) level, i.e., measures the consumption of the cell's computational resources as well as QoS parameters, e.g., the execution time. This information influences the decision making in the reasoner component. The service registry component hosts service implementations and enables the fog action control to search for services and deploy them on the fog cell compute units. As storing service implementations is resource-consuming, the service registry is located in the storage unit of the control node.

It should be noted that an alternative approach would be a decentralized orchestration of fog cells in a fog colony, i.e., without a centralized fog orchestration control node. While this leads to higher fault tolerance, it also involves extensive coordination and voting between the involved fog cells. Therefore, we opt for a more centralized approach in this work. However, we still foresee that another fog cell becomes the fog orchestration control node in a fog colony, if necessary, e.g., in case the original fog orchestration control node fails.

III. Solution, Implementation & Evaluation

A. Problem formulation

Fog Computing is geared towards taking full advantage of the nodes within the network. The Fog Computing application model will be represented by modules and the relationship between them. The resource placement algorithm assigns modules to the nodes in the network to match the policy. The problem set forth in this paper will consider setting the application priority latency module.

The computing power of a node is represented by a finite set of constants, and can also be represented by three basic values of a computational unit: the CPU (measured in MIPS), the RAM and bandwidth. In addition, there is still a problem that the storage capacity of the node, the node's traffic is large enough to accommodate the module or not. This factor can still be considered if the attribution policy considers storage issues. So if n_i represents the n-th node in the network. I will perform the following:

And the set N can be divided into two subsets: the set of nodes in Fog and the Cloud.

N_F : Set of fog nodes in fog layer

N_C: Set of cloud in cloud layer

$$N_F \cup N_C = N$$

$$N_F \cap N_C = \emptyset$$

Thus, to represent the computational power of node n_i equals:

$$Cap(n_i) = < CPU_i, RAM_i, Bandwidth_i >$$

The application model is represented by a directed acyclic graph. The data moving inside the model is called tuple. In an IoT application, tuples need to have a starting point (the point at which the data is generated) and the end point (where the tuple is finally processed and returns the response). So if the graph has a cycle, the data streams will probably run forever.

As mentioned above, the tissue model consists of modules and the relationship between them. The relationship here means that this module will send to the other module tuple and their tuple. Types of information include request and response. With these two types will generate two-way send information (upstairs or downstairs), specifically, the request will be sent on, the response will be sent down. This helps to minimize the hop count of the data. When data is generated (the sensor or IoT device captures information and generates data), data should be sent to the processor at the top of the receiver and processed. After the processing, the information should be returned (response) so the send direction should go down to go back to the IoT device. In short, the requesting module must be at the lower layer than the request receiving module, and the response module must be at a higher level than the response module. Note that the final module is different from the first module. An example of a graph as shown below: $G = < V, E >$

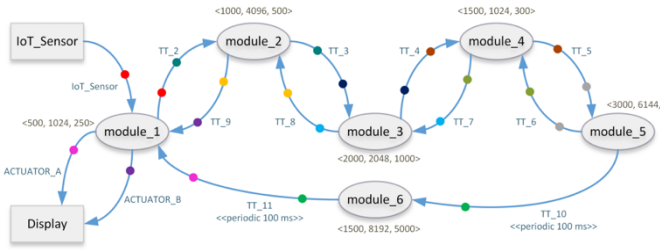


Figure 4 Example of application model[9]

V is the vertex of the graph. There are a vertex that generate or gather data and a vertex receiving response of the application. These points is placed inside devices such as sensors and monitors,... This means that we can consider that two vertices as both a module and a device. Those modules has been assigned to the devices and the algorithm will not consider those devices and modules to assign.

The tuples also have their own relationship. See the example on , we will see the colored dots as red next to IoT_Sensor - module_1 - module_2. IoT_Sensor sends to module_1 tuple named IoT_Sensor, when module_1 receives that tuple and processes it, converts it into tuple TT_2 and sends it to module_2. The probability of conversion between them also has a certain probability of being set individually. Furthermore, tuples also has 2 types – request and response.

B. Algorithm

Assume network map is represented in tree. An example is below.

The module placement algorithm is geared towards minimizing application latency. So we will use *First Fit Placement* for the module placement in this paper. The algorithm will queue modules and devices in priority. The devices are sorted in the order closest to the data source(sensor). The order of the modules has 2 constraints. If module A send a request tuple to module B, then module B must be placed at the device containing module A or parent of the device containing module A. The second constraint is, if module A send a response tuple to module B, then module A must be placed at the device containing module A or parent of the device containing module B. This makes sense to minimize network hop counts.

First Fit Placement

For all IoT_Sensor

P = Path from parent of IoT_Sensor to cloud

placedModules = ∅

For D ∈ P

modulesToPlace = *getModulesToPlace*

While (modulesToPlace != ∅)

D = Parent(D)

M = modulesToPlace.get(0)

If P had an instance of M as M'

If D kept M'

If D can host M

Add M to placedModule

Remove M from modulesToPlace

Update modulesToPlace

Else move M and involved modules to Upside

Else if D can host M

Add M to placedModule

Remove M from modulesToPlace

Update modulesToPlace

Remove M from modulesToPlace

If placedModules == allModules

Break;

The First Fit Placement algorithm begins to browse all possible paths from the End Device to the Cloud in the physical network and places modules in the application model on those paths. The browsing order on path P is from the lowest-level device to the highest (Cloud). For each device, the algorithm starts to initialize the list of modules to be assigned based on the 2 constraints mentioned above.

When the list of modules need to be place was defined, the algorithm starts assigning modules one by one. Suppose the module m is being considered to be placed on device d on the path p. The algorithm checks if an instance of m has been placed on the p. If d already contained m and continue to check if d is able to contain m or not. Otherwise it will shift m and involved modules to m to the parent of d. In contrast, m will be placed on the device d. If d or parent devices of d do not have an instance of m then the algorithm of considering d is likely to run m or not. If so, assign m to d and update the module list to assign.

And the complexity of First Fit Placement is O(S * M) (S is the number of sensors – M is the number of modules)

C. Implementation

In this paper, we will use iFogSim - A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments.

The First Fit Placement algorithm is implemented as the FirstFitPlacement class extends the existing ModulePlacement class of the iFogSim. After extending the ModulePlacement, we will have to override the mapModule() method to implement the proposed algorithm. And finally we provide list of devices, list of modules and application module for the Controller to run the simulation.

For the convenience of tracking and editing network topology, we have extend GUI of iFogSim. Once the necessary FogDevive has been created, iFogSim would build a network tree based on those devices.

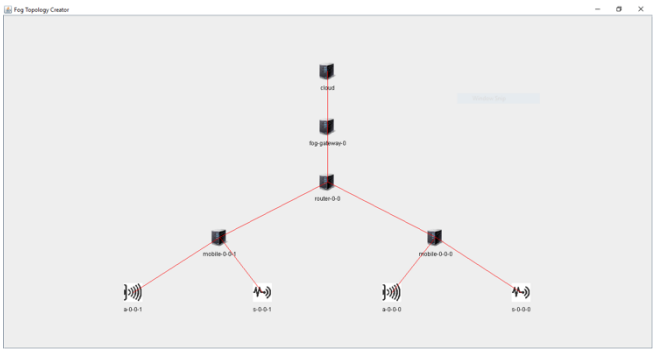


Figure 5 Network topology

Before submitting the FogDevice to the system, you can check and edit again according to the wishes of the user. When you want to change the configuration of the device, just click on the device and edit.

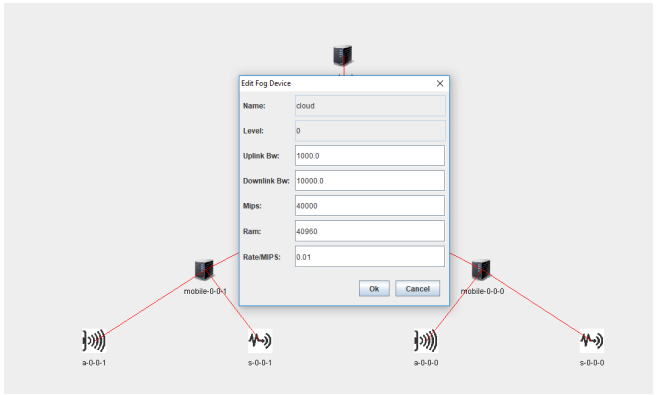


Figure 6 Edit network topology

D. Evaluation

The application used for the simulation was HealthCare System. The application module is shown below.

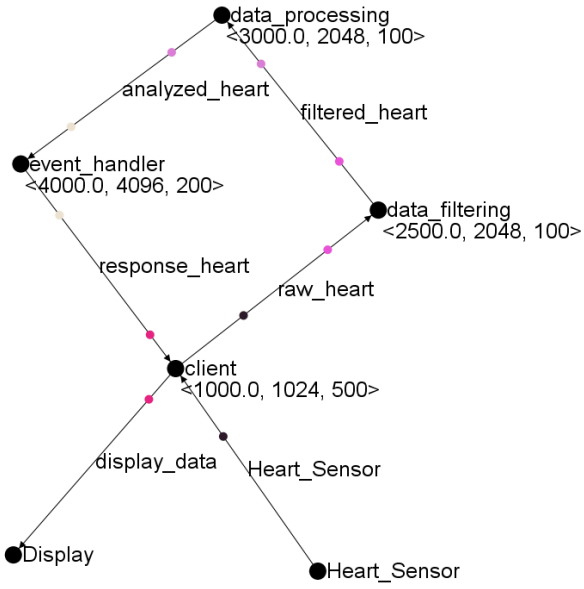


Figure 7 Healthcare System Application model

The application consists of four main modules - client, data_filtering, data_processing and event_handler.

1. **Client**: The client module is an application running on a smartphone, which receives all raw data that the sensor captures and sends that data to data_filtering to filter the data. After processing the data, it will output to the display of the smartphone.

2. **Data_filtering**: After the client module receives the data and sends it to the data_filtering, the filter starts to operate, which removes unnecessary data such as malfunctioning or other factors. Once it has filtered it sends that data to the main processor data_processing.

3. **Data_processing**: At this module, we begin to accurately calculate the information obtained from the patient's heartbeat and assess the patient's current state of health. And when the calculation is complete it sends the event-_handler module to the response module.

4. **Event_handler**: The module assumes the patient status and selects the information to be displayed to the user. If the situation is stable, then show up, if the alarm, alarm should be monitored timely treatment.

Table 1 EXPERIMENTAL NETWORK CONFIGURATIONS

Tuple	CPU (MIPS)	Bandwidth (kB)
Heart_Sensor	3000	500
raw_heart	2000	500
filtered_heart	1500	1500
analyzed_heart	1000	700
response_heart	1000	500
display_data	500	500
#The average tuple emission rate of a sensor is 10 milliseconds, specified by a deterministic distribution		

The network consists of 1 cloud, private areas of the hospital or place to be surveyed (each zone has 1 fog-gateway), two routers within each zone and finally two smartphones connected to a router. There are a total of 5 configurations with 1, 2, 3, 4 and 5 successive measuring ranges. Below is an example of the topology of config 2.

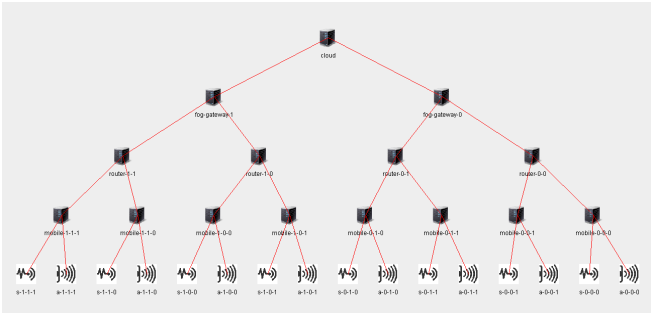


Figure 8 Network topology of Healthcare System

Table 2 Network Capacity of Fog Nodes

Device	Upbandwidth (Mbps)	Downbandwidth (Mbps)
Mobile	100	250
Router	10000	10000
Fog-gateway	10000	10000
Cloud	1000	10000

Table 3 Calculating Capacity of Fog Nodes

Device	CPU (MIPS)	RAM (MB)
Mobile	3000	2048
Router	6000	6144
Fog-gateway	7000	8192
Cloud	40000	40960

Table 4 Power Consumption of Fog Nodes

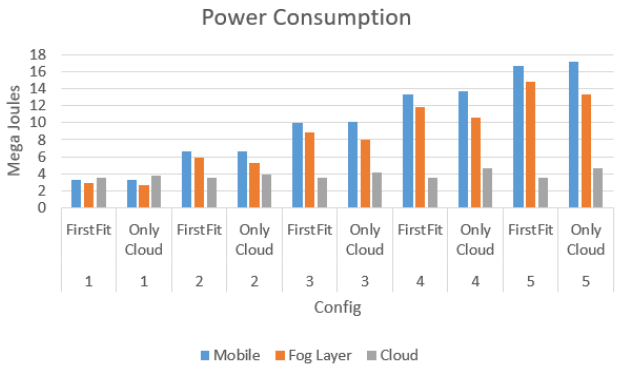
Device	Power Consumption (W)
Mobile	87(B), 82(I)
Router	107(B), 83(I)
Fog-gateway	200(B), 100(I)
Cloud	500(B), 300(I)

Table 5 Latency between Fog Nodes

Source	Destination	Latancy (ms)
Sensor, Display	Mobile	2
Mobile	Router	5
Router	Fog-gateway	20
Fog-gateway	Cloud	200

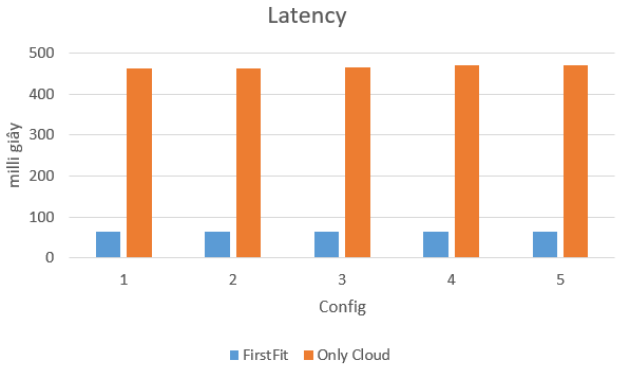
The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

1) Power Consumption



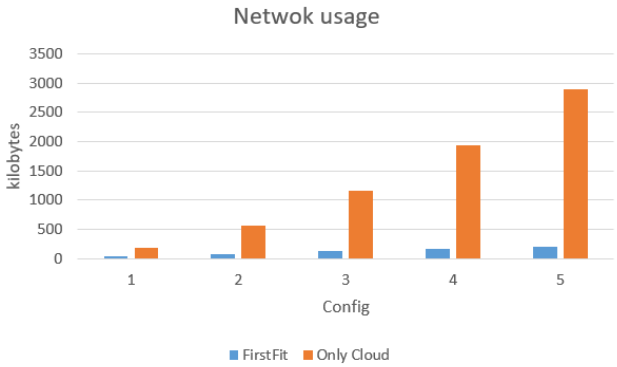
As we can see, using First Fit Placement, the energy consumption of devices in fog layer is slightly higher than in cloud computing. When the algorithm is complete, no cloud is needed to contain any modules so the energy level is static. But the difference is not much.

2) Latency



As you can see in the graph above, there is a huge difference between using Fog Computing and Cloud Computing as a latency issue. As we use the Fog device, we will minimize the amount of data transfer to the upper layer which means that the response time will also be shortened.

3) Network usage



As mentioned above, we will minimize the use of high-end devices while reducing the amount of network traffic that must be routed on the network. When using Cloud Computing, we will have to send from the bottom of the sensor to the top of Cloud, meaning that all traffic must be used and network traffic will be pushed up to the maximum. Conversely, when using Fog Computing, we can ignore some unnecessary traffic and traffic, so that bandwidth is more comfortable and more economical.

IV. Conclusions And Future Work

Fog Computing is still a new concept. It's just like Cloud Computing, but instead of choosing the largest server as data center, it chooses to build as a server network scattered over many places. Fog Computing is a huge computer, can run an application instead of using only cloud.

To summarize what has been done, we has built an algorithm for placing modules on network devices in a particular application. The algorithm is geared towards picking the device closest to the data source. This means that the latency issue is considered to be solved.

We was successfully run by the iFogSim tool and applied its algorithm to it. iFogSim also has an algorithm for distributing modules but the algorithm is still limited because it only considers each CPU element and ignores some other factors. We also adds some functions in iFogSim which are interface and print additional parameters to output for observation such as total energy and energy levels.

The future of development is to address the shortcomings of iFogSim and its algorithms. Specifically, we will develop some algorithms according to each criterion. The First Fit algorithm in the report only serves to satisfy iFogSim, not for real systems. We want to develop a dynamic algorithm that will help us to place modules during the application runtime in a flexible way.

Next, we will seek to build a real system, applying fog computing theories to the entire system. Estimated data will be unstable as in the simulator because of the fact that there will be more positive effects. The algorithm can be static or dynamic depending on whether the fog orchestration control is set to the system.

ACKNOWLEDGMENT

We thanks to Dr. Tran Minh Quan for his value comments.

REFERENCES

[1] F. Jalali, Energy Consumption of Cloud Computing and Fog Computing Applications, Australia, 2015.

[2] F. Jalali, K. Hinton, R. Ayre, T. Alpcan and R. S. Tucker, Fog Computing May Help to Save Energy in Cloud Computing.

[3] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh and R. Buyya, Fog Computing: Principles, Architectures, and Applications, 2016.

[4] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy and G. Jiang, Power and performance management of virtualized computing environments via lookahead control, 2008.

[5] X. Fan, W.-D. Weber and L. A. Barroso, Power Provisioning for a Warehouse-sized Computer, 2007.

[6] A. Beloglazov and R. Buyya, Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers, 2012.

[7] H. Gupta, A. V. Dastjerdi, S. K. Ghosh and R. Buyya, iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments, 2017.

[8] R. Deng, R. Lu, C. Lai, T. H. Luan and H. Liang, Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption, 2016.

[9] M. Taneja and A. Davy, Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm.

[10] J. Santos, T. Wauters, B. Volckaert and F. D. Turck, Resource Provisioning for IoT application services in Smart Cities.

[11] O. Skarlat, S. Schulte, M. Borkowski and P. Leitner, Resource Provisioning for IoT Services in the Fog.