Journal of Simulation

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tjsm20

Mobile experimentation using modelling and simulation in the Fog/Cloud

Khaldoon Al-Zoubi & Gabriel Wainer

To cite this article: Khaldoon Al-Zoubi & Gabriel Wainer (2021): Mobile experimentation using modelling and simulation in the Fog/Cloud, Journal of Simulation, DOI: 10.1080/17477778.2021.1964393

To link to this article: https://doi.org/10.1080/17477778.2021.1964393

Published online: 20 Aug 2021.

JOURNAL OF SIMULATION
https://doi.org/10.1080/17477778.2021.1964393

Mobile experimentation using modelling and simulation in the Fog/Cloud

Khaldoon Al-Zoubi[a] and Gabriel Wainer[b]

[a]Faculty of Computer & Information Technology, Jordan University of Science & Technology (JUST), Irbid, Jordan; [b]Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada

ABSTRACT

Recently, we have seen an increase of the use of Cloud computing to provide Modelling & Simulation services. In general, mobility support is lacking since clients normally contact the same centralised clouds regardless of their locations. In this research, we propose a method and algorithms to build *Fogs* as private services in which different middleware running in varied Virtual Machines (on various distributed *Fogs* and *Cloud* backbone) to expose different services. Clients access services via nearby Fogs, which then discover the required resources to run experiments (on Fogs or cloud backbone). Our focus is on mobility, to permit clients to conduct experiments (and visualise simulation results) using mobile devices. Mobility support is further improved by introducing the novel concept of a *mobile simulation experiment*: if a device moves away from a given Fog zone, the experiment moves with the device. We present a prototype implementation, evaluation results, and different case studies.

ARTICLE HISTORY
Received 9 April 2021
Accepted 29 July 2021

INDEX TERMS
Modelling & Simulation (M&S); mobile computing; cloud computing; Fog computing; edge computing

1. Introduction

The popularity of mobile devices (which are geographically dispersed and use wireless network access) has raised what is expected from them. Mobility support can improve Modelling and Simulation (M&S) by accessing and consuming simulation services from virtually anywhere. Users can try to predict certain outcome scenarios by inputting real parameters into a simulation on demand (for instance, by studying how the weather would affect traffic based on real input factors like snow or rain).

Mobility access to M&S services can be complemented by cloud computing, which can offer computing resources, which is important in those cases where simulation execution needs advanced computation resources to obtain meaningful results. On the other hand, using M&S on the mobile devices has major restrictions: battery life, storage, processing power, and network connectivity. So far, this problem has been solved by partially offloading computation onto cloud resources. Simulations are divided into tasks where the mobile device drives the simulation and decides whether to execute tasks locally or to offload them onto the Cloud. This approach treats mobile devices as computing units like cloud resources, which add computation overhead on devices and require specific simulation tools to be installed on those devices. In addition, devices (regardless of their locations) need to access centralised datacenters. A diversity of issues related to accessing centralised resources in the Cloud have been researched for the Internet of Things (IoT). In IoT applications, mobile nodes (sensors, vehicles and phones) must be provided with adequate mobility support, geo-location awareness, and low latency (Hu et al., 2017; Naha et al., 2018; Yi et al., 2015). To deal with these issues, in 2012 Fog computing was introduced as a new method to decentralise resources by building mini clouds, called Fogs (Bonomi et al., 2012). The idea is that near users perform full or partial computations on behalf of the cloud backbone. Users access services through nearby Fogs, which are usually built as private clouds with local organisations resources, avoiding contacting the main cloud backbone, when possible, hence, opening the way to improve mobility support and quality of service in the way clouds provide services.

Our research focuses on how to build actual Fog services and the cloud backbone to run simulation as a service for different models. Fogs are built in a similar way of the cloud backbone, but with lesser resources, hence, they are mini clouds spread near users (Figure 1).

We propose a Fog/Cloud architecture and present a prototype implementation using actual private clouds based on the OpenStack (Gai, 2020), a well-known open-source software (started by Rackspace and NASA in 2010) that can be used to build public/private clouds. OpenStack core components are compute nodes (to deploy virtual machines; hence, servers), network nodes (to handle communication between virtual machines and the external world), and

CONTACT Khaldoon Al-Zoubi ktalzoubi@just.edu.jo; khaldoon_alzoubi@hotmail.com Faculty of Computer & Information Technology, Jordan University of Science & Technology (JUST), Irbid, Jordan

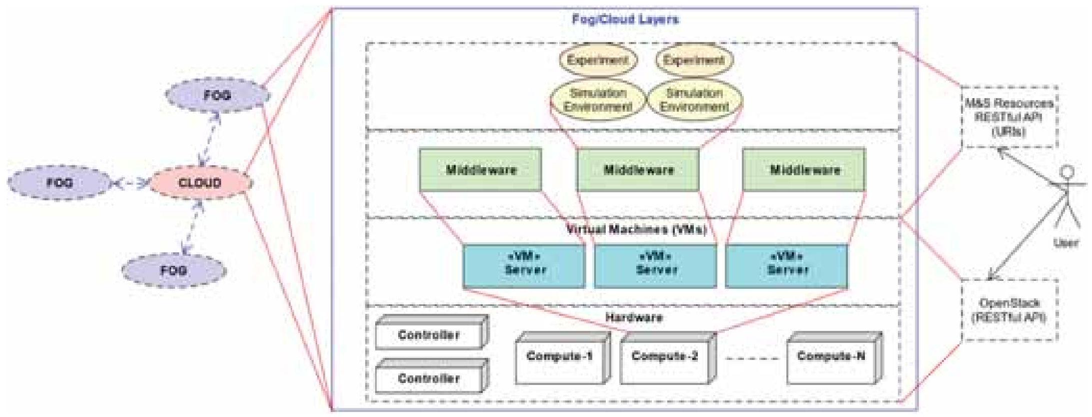


Figure 1. Fog/Cloud infrastructure overview.

controller nodes (to orchestrate VMs deployed on compute nodes). Within each Fog or Cloud stack (Figure 1), OpenStack manages and controls the bare hardware and the virtual machines (VMs) layers. Each VM runs a RESTful-based middleware with the capability of exposing different simulation environment services. Accordingly, Fogs and Cloud simulation and computing capabilities can be built and configured by system administrators as desired. Note that our work in (Al-Zoubi & Wainer, 2020a) describes how such resources are managed and orchestrated, while our work in (Al-Zoubi & Wainer, 2020b) discuss the scalability of such methods. This research focuses on mobility support for simulation experiments while being aware of the location of those devices to enhance mobility support. Clients can setup/start their experiments via nearby Fogs, which then discover the required VMs to run those experiments within specific simulation environments, according to that experiment settings. These servers may exist on a local Fog, another Fog, or on the cloud backbone.

To handle mobility, the devices are built as clients that communicate with an API for experimentation via a nearby Fog using RESTful web services. The mobile devices are used for visualisation of simulation results from the Fog/Cloud, and they can also be used to input parameters into the simulation loop. The mobile devices are not involved in the actual simulation cycle, avoiding installing simulation software on the device and saving power and storage.

This is the first research platform providing simulation experiments as services via real Fog/Cloud while allowing actual mobile devices to consume such services, and further move those services to be closer to mobile devices locations. Our contribution includes the definition of an architecture and the construction of real *Fogs* to expose M&S resources, allowing users to consume those services and run their experiments using their mobile devices (in contrast to most works that focus on simulating Fogs to study their behaviour and performance). We are also interested in reducing the computation overhead of the mobile devices using our platform, by building mobile Apps as lightweight RESTful clients that manipulate the experiments on the *Fogs* (and visualise returned results). No specialised simulation software is needed to be installed on devices (in contrast, environments that consume services directly from the cloud backbone normally treat mobile client as computing units that drive the simulation and decide where to execute tasks, increasing overhead). Finally, we also want to enhance *mobility support* with the help of Fog computing. We introduce a new concept: the *mobile simulation experiment*. If a device moves away from its current Fog zone, the experiment moves with the device to a closer Fog, based on the device's new location.

The rest of the paper is organised as follows: Section 2 surveys related work. Section 3 presents the novel concept of the mobile simulation experiments (that reside on the *Fog* side), and Section 4 presents number of use cases from the mobile device perspective. Section 5 presents mobility support evaluation results while conclusions are presented in Section 6.

2. Related work

As discussed earlier, the research presented here allows users to use their actual mobile devices to consume and run simulation experiments as services on actual privately built *Fogs*. Those experiments are mobile as they may travel according to devices locations to nearby Fogs. Here, we present different research efforts related to our research, including: **(1)** Simulation software built to simulate Fogs and Clouds; **(2)** Mobile/Cloud-based Simulation (i.e., simulations executed on real mobile devices and real cloud computing resources), where no Fogs were used, **(3)** Fog Computing Applications (i.e., data collected by distributed devices and processed by Fogs and clouds, these researches not related to M&S); **(4)** Fog

Computing infrastructure (i.e., resource allocation and scheduling).

2.1. Fog/Cloud simulator tools

There has been extensive work done on Cloud, Fog and Mobile Simulators (i.e., models and simulators used to study Clouds and Fog computing systems), including CloudSim (CloudSim, 2020) (simulates service brokers, provisioning, and network), CloudNetSim++ (Malik et al., 2017) (simulates datacenters energy consumption and communication between datacenters), or GroudSim (Ostermann et al., 2010) (simulates cloud job executions and load distribution), or simulators devoted to study Fogs like iFogSim (Gupta et al., 2017) (simulates IoT devices connected Fogs), and for mobile devices like simulators MofySim (Ju et al., 2016) and 5 G K-Simulator (Kim et al., 2018). Further, migrating resources between Fogs were simulated to accommodate mobility in some tools like the presented work in (Carlo et al., 2020), which simulates migrating entire VMs between Fogs. Interestingly, VMs migration was ruled out from our presented system early on. This is because, in our case, VMs are middleware that could be shared by different users and might be running different experiments for the same/different users. Therefore, we only migrate experiments, which usually represented as few files.

This research is not on the simulation of *Fogs*, but on the use of *Fogs* computing services to provide M&S as a service.

2.2. Mobile/Cloud based simulation services

M&S researchers have used clouds to expose and execute simulation services where the simulation can then be accessed and consumed by clients (e.g., Web browsers) (Rehman et al., 2019; Shekhar et al., 2016). To provide mobility support, some research enabled running simulations using mobile devices. In this case, the simulation is divided into tasks queued on the mobile device, which drives the simulation cycle deciding if the tasks run on the local device or are offloaded to the cloud. For instance, (Guan et al., 2016) proposes an HLA-based mobile simulator, which partitions the simulation into tasks and defines the high-computing tasks that require to be executed remotely or locally. This research concludes that in most cases it is better to offload all tasks onto the cloud. In (Amoretti et al., 2015) multiple mobile devices offload tasks to a cloud component called Dispatcher, which distributes the tasks among VMs to balance the computation load. The work in (Yang et al., 2017) developed a system model that divides the edge computing resources wireless bandwidth into virtual channels. Each mobile device sends the cost of an application execution to the edge computing unit, which then assigns it one of the virtual channels; hence, it partitions the whole edge-computing bandwidth among users' devices.

Although these research results have some relation to ours, their objectives are different to the ones we outlined in the Introduction: we want the need for Fog services to run simulations for different models; our focus is on mobility, saving computations cycles, avoiding installing specific software locally on the device and saving power and storage.

2.3. Fog computing applications

Fog computing-based applications main purpose is to enhance users' quality of service by bringing some computation close to users like in the case of healthcare systems or transportation systems such as (Giang et al., 2016; Hou et al., 2016). In those type of systems, based on Internet of Things (IoT), the scalability in the number of mobile devices is essential. IoT mainly deals with interconnecting any number of mobile nodes (e.g., sensors, vehicles and phones) via the Internet, with high mobility support, geographical distribution, location awareness, and low latency (Bonomi et al., 2012). However, as it has been realised in recent years that these requirements go against the nature of clouds, which mainly exist in centralised datacentres. In this case, all devices (regardless of their locations) need to go to the centralised cloud datacenters to consume their services.

Healthcare sector-based applications process data from a large number of spread devices, like in (Mahmud et al., 2018), which proposes an IoT healthcare-based system to optimise data communications and power consumption; or (Rahmani et al., 2018), which uses fog computing layers between the sensor nodes and the cloud, in which the fog layers perform partial sensor computation on behalf of the cloud. Smart Cities also apply Fog computing technology. For example, IoT-based connected vehicles develop smart transportation systems providing driving directions in an urban area (Giang et al., 2016). Others use vehicles as the means of computation and communication, and they use the vehicle resources (Hou et al., 2016).

As can be seen that other related work is mainly used to collect, store, or communicate data. In contrast, we apply here the Fog computing technology to allow mobile devices to setup and execute simulation experiments as services.

2.4. Fog computing infrastructure

In this category, researchers proposed methods for *resources allocation* such as (Pooranian et al., 2017; Taneja & Davy, 2017) and for *Fog collaboration* such as (Alsaffar et al., 2016; Al-Zoubi & Wainer, 2020a).

Resource allocation methods mainly deal with allocating and scheduling VMs to compute tasks. For example, in (Taneja & Davy, 2017), the algorithms deploy applications modules, as needed, onto Fog layers close to users and improve their user experience and quality of service. Energy-based dynamic resources allocation can be used to improve energy use in the Fog computing centres as in (Pooranian et al., 2017), where groups of VMs are allocated in the physical servers, and the VMs are selected based on energy-related parameters in each computation iteration.

Fog collaboration mainly deals with where to execute jobs. This requires coordination between Fog and Cloud resources. For example, a linearised decision tree can be used to balance users submitted jobs, categorising each job using three parameters: size, completion time, and VM capacity (Alsaffar et al., 2016). Those methods then decide where to execute a job either on the Fog or on the Cloud. Similarly, *Fog* collaboration can execute heterogeneous simulation jobs, if allocating VM(s) only is not enough but ensuring that those servers also support the tools to run the simulation based on users' requirements (Al-Zoubi & Wainer, 2020a). This means that servers/VMs cannot be compared to each other based merely on computing capabilities.

This research focuses on enhancing mobility support on the *Fog* to conduct simulation experiments using mobile devices. However, we reused the M&S resources discovery and management from our research in (Al-Zoubi & Wainer, 2020a), which studied on large scale in (Al-Zoubi & Wainer, 2020b). It is important to note that migrating experiments, in our case, are not expensive since they are usually few files defining experiment settings and other files like model scripts; hence, we ruled out VM migrations early in the project. In our case, each Fog is responsible for one or more geographical zones; the mobile Apps embed their device locations within their regular requests sent to experiments, and based on this, contacted Fog decides what to do about this device and the subject experiment. If an experiment is migrated to another Fog, the new URI for that experiment is passed to that device. This is complex process as multiple devices could be connected to the same experiment from same or multiple Fogs, the experiment could be active and running simulation over distributed resources, and so on.

3. Mobile simulation experimnets

This section presents the novel concept of mobile simulation experiments over the *Fog* architecture. We first present the overall experiments organisation over the *Fog* architecture; we then discuss the factors (like the experiment state) used in determining on how to move an experiment to another Fog. Finally, we discuss the procedures taken by the system to complete experiment travelling between Fogs.

3.1. Experiment Fog architecture and requirements

Figure 2-1 shows the relationship between users and experiments. Each user is an account with username and password. Users can create any number of experiments where each experiment is exposed to the outside world as URIs with defined RESTful API, as described later. This makes experiments independent from each other regardless of their owners. It is worth to point out that mobile experiments move with devices, thus, it is possible for a user experiments to spread out over multiple Fogs.

Devices (i.e., mobile Apps) access simulation experiment services through Fogs (Figure 2), and to be specific, via experiment URIs that are exposed via Ingress Fog servers. In our case, those Fog servers are addressed by the same floating IP address, hence seen by the outside world as a single server. For discussion simplicity, we refer to those group of servers as Fog server throughout this paper. Experiment URIs are prefixed with the *<Fog-Server-base-URL>*, which contains the

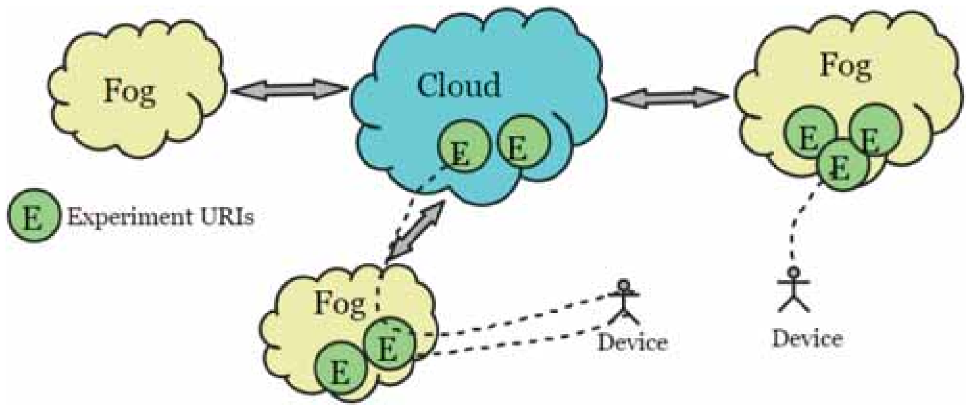


Figure 2. Overall experiment Fog/Cloud architecture.

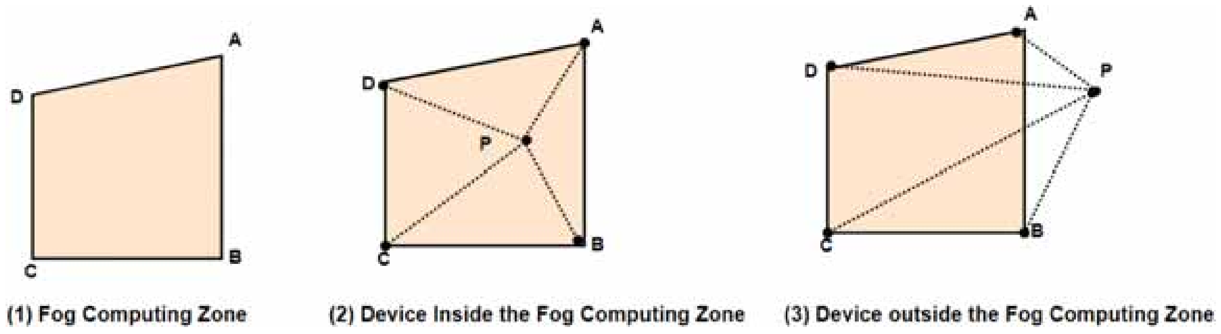
ingress Fog server-specific information like the IP address; hence, it serves the client's (e.g., mobile App) pointer to a certain Fog. As a result, this part is the only one changes in an experiment URI when it moves to a different Fog. This makes sense, because from a client perspective, the only thing changes is the server gateway of that experiment. However, the rest of the experiment URI structure stays the same regardless of its actual location. This will further be discussed in more details in Figure 4 (Section 3.2). It is worth noting that Figure 2 simplifies the overall picture. However, in practice experiment might be distributed over different locations (i.e., experiment partitions) as in the case of distributed simulation, see details in (Al-Zoubi & Wainer, 2015).

Based on above, when an experiment is moved to a different Fog, the device only needs to be updated with the new Fog base URI so that it can attach it to the beginning of the URI string when it interacts with the experiment.

For an experiment to be mobile (i.e., that can move), users, devices, and Fogs should support experiment mobility, as follows:

(1) *Usernames*. User accounts are a single user or a team. If users choose to support mobility, the primary copy of their information (e.g., credentials) is stored on the Cloud (while cached on Fogs, as needed).
(2) M*obile device* (i.e., *phone App*). If supports mobility, it includes its geographical location coordinates within its regular requests to experiments. In our case, this location is sent within the XML message that is usually sent via HTTP POST and PUT requests, while it is sent in the query parameters for the HTTP GET requests. For example, a device is willing to move the experiment, if it includes the following in its request *<mobile><location> <X > 32.491998</X> <Y > 35.988219</Y> </location>ile>*. This allows device holders to disable experiment mobility on their devices if they do not grant location access for the mobile App.
(3) *Fogs* should be configured to be responsible for one or more geographical zones. This allows a server upon a request receipt to determine if a device current location is inside or outside its Fog zone. To simplify this calculation, in our case, Fog zones are always ensured to be defined as quadrilaterals (Figure 3-1).

The Fog zone (Figure 3-1) is limited to four points to reduce unnecessary computation overhead when determining a device still within a Fog zone or outside of it. To do so, Fogs first need to pre-calculate the area of their zones. In our example, the area of the shown quadrilateral in Figure 3-1, which is the sum of the two triangles of *ABD* and *BCD* (i.e., by connecting points *B* and *D*). After that, upon a message receipt from a device, each pair of points in the Fog zone are connected to the device reported location *P*. This leads to


A
D
C
B
(1) Fog Computing Zone
A
D
P
C
B
(2) Device Inside the Fog Computing Zone
A
D
P
C
B
(3) Device outside the Fog Computing Zone
Figure 3. Example of determining a device inside or outside a specific Fog zone.

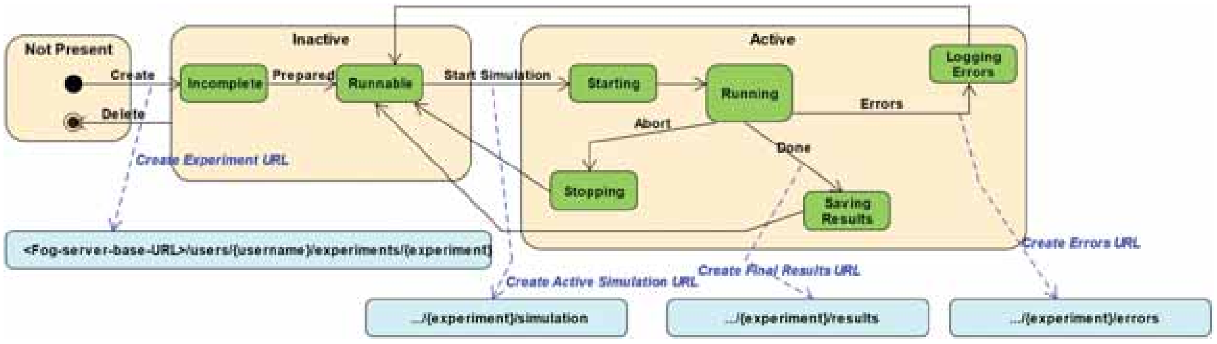


Figure 4. Experiment state machine.

the formation of four triangles: *APB, BPC, CPD* and *APD*, as shown in Figure 3-2 and Figure 3-3. Figure 3-2 shows when the reported location *P* is inside the Fog zone; hence, the sum of the four formed triangles (with respect to *P*) equals the Fog zone precalculated area. Otherwise, it is outside the Fog zone (Figure 3-3), hence, the area of the four formed triangles larger than the Fog zone area.

3.2. Moving experiment deciding factors

Based on the above requirements, we assume that all experiments are moveable, as our focus here is on mobile experiments. Of course, moving a specific experiment from a Fog to another depends on several factors. The first of those factors is obviously the *experiment state*. Each experiment can only evolve in different states independently from other experiments (Figure 4) regardless of their owners.

Figure 4 shows the three possible composite states of an experiment: *Not Present, Active*, and *Inactive* states. *Not Present* state indicates that the experiment does not exist on the contacted Fog by a device (but might exist on other Fogs). An experiment in the *Active* state indicates that this experiment is currently executing simulation; hence, M&S resources have been allocated to this experiment where those resources might exist on the contacted Fog, other Fogs, or the cloud backbone. The *Inactive* state indicates that the experiment exists (on the contacted Fog), but simulation is not currently being executed on this experiment; hence, M&S resources are not allocated to run simulation.

The *Inactive* state (Figure 4) contains two internal states: *Incomplete* and *Runnable* substates. The experiment goes into the *Incomplete* substate upon the experiment *creation* request is received from a user. Creation request is HTTP PUT on URL *<Fog-Server-base-URL>/users/{user}/experiments/{experiment}* where *{user}* is user's username and *{experiment}* is the experiment given name by the user. However, before creating the new experiment, the Fog needs to make sure that the experiment does not exist in some other Fog, as discussed later in Table 1 and Figure 8. The *Runnable* substate indicates that simulation can be executed for this experiment. This happens when all required settings and data have been uploaded onto the experiment. From this time forth, simulation can be executed within this experiment. This done by sending HTTP PUT request to create URL . . . */{experiment}/simulation*. This moves the experiment into the *Starting* substate (within the *Active* state).

The *Starting* substate ((Figure 4)) indicates that the experiment is in the process of discovering and selecting M&S resources that can execute the simulation. Those resources may exist anywhere on the local Fog, on other Fogs, on the cloud backbone, or mixed between them. Here, we reuse the scheduler component from our previous research in (Al-Zoubi & Wainer, 2020a) to carry out the discovery & selection operation. After those resources are allocated, the experiment (on Fog) creates other experiments (like any other experiment) on those found servers to run the simulation. Accordingly, the experiment becomes in the *Running* substate (where live results could be read by the mobile App, if required). Of course, reading live results from a running simulation needs to be supported by that specific simulation environment since our proposed platform is independent from any specific simulation tool. For example, the CD++ (Wainer, 2009) version that supports this feature requires clients to include in their HTTP requests the simulation time window (i.e., start and end time) that they need results for. For example, window [t1, □) extracts results from t1 until last available simulation time. This is according to REST principles that clients should include all needed information to perform their requests. Thus, it becomes the responsibility of the clients to keep track of the last simulation time they got results for and how to advance time in the following requests. This means that they also control the frequency of sending those requests, as it is related to the situation of their internal processing. The main advantage of this approach is that experiments are decoupled from local specifics of clients. For instance, multiple clients can easily be served by same experiments without complicating experiment states and management on the server side.

At this point, the experiment goes back to the *Inactive* state through one of the following three substates: *Stopping* (i.e., if user intentionally aborts simulation), *Logging Errors* (i.e., if user's simulation model contains errors), and *Saving Results* (i.e., if simulation is completed, results then stored on URL . . . */{experiment}/results*), hence, results are always available for future download and simulation replay by clients.

To put above discussion in a simple example, Figure 5 shows a simple use case interactions between a client and the Fog/Cloud platform. In *step #1*, the client (mobile app) sends request to the nearby Fog (*Fog-A*) to create experiment. The client assumes the nearby Fog is the one that had cached its accessed URI (*Fog-A* in our example). For simplicity, we assume that this request was received from a device within the receiving Fog zone, and the experiment does not

Table 1. Mobile experiment cases.

Case	Experiment State	Experiment Type	Mobile Device Location	Action Taken
1	Not Present	N/A	Inside/Outside Fog Zone	Figure 8
2	Inactive	Primary/ Mirror	Inside/Outside Fog Zone	Figure 9
3	Active	Primary/ Mirror	Inside/Outside Fog Zone	Figure 11

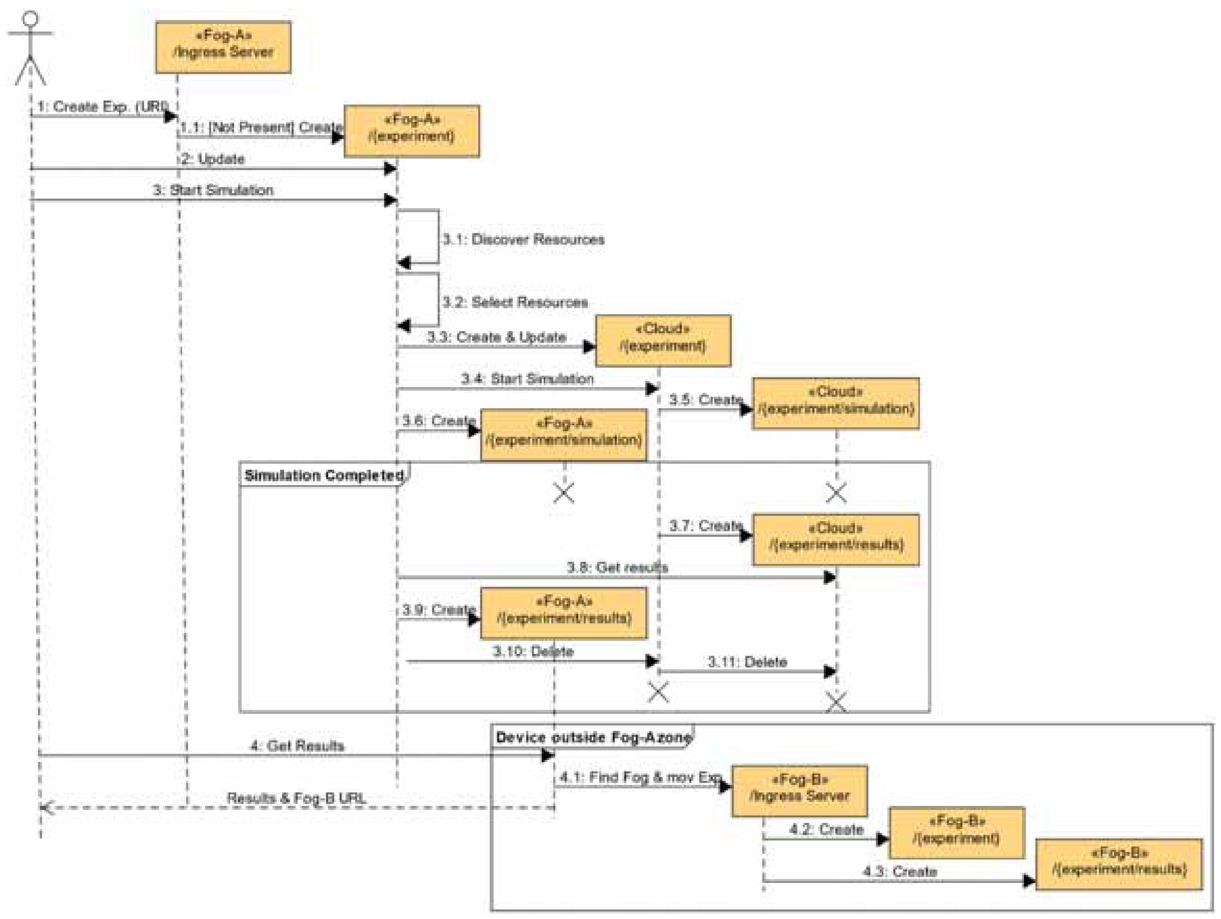


Figure 5. Simplified example of client/experiment interactions.

exist on this Fog or elsewhere across the entire Fog/Cloud platform (i.e., this is because experiments can move as discussed soon in Section 3.3). As a result, the experiment main URI is created; hence, the client can then update and build this experiment via this URI as desired (*step #2*). Now, when the client decides to start the simulation in this experiment (*step #3*), the experiment then needs to (1) discover the resources that can execute the simulation. For instance, if an experiment can only be executed by CD++, we only then need to find servers with CD++ capabilities. (2) select out of those found resources best ones to execute simulation considering factors like load, utilisation, and locations. Our work in (Al-Zoubi & Wainer, 2020a) have fully described this platform resources management including resources orchestration, discovery, and selection. As a result, resources to execute the simulation could be found on a Fog or on the cloud backbone. In this example (*step #3.3*), the simulation is offloaded to the Cloud. In this case, the experiment on the Fog, creates a temporary experiment on the cloud to run the simulation. At this point, URI . . . /{experiment}/simulation is used to interact with running simulation. Now, when simulation is completed, results are retrieved from the cloud and stored on the local Fog (in URI . . . /{experiment}/results). It further deletes all allocated resources on the Cloud. As can be seen that a request might be received from a device that has moved away from its original Fog. For example, Step #4 shows that the device has moved from Fog-A to Fog-B zone. This is detected by Fog-A based on the geo information in the device request that was just requesting the experiment results, as will be discussed soon. As a result, a new Fog (*Fog-B*) is found for that experiment based on its location. The new Fog URI is slipped back to the device along with the response to the original request. From now on, the device contacts Fog-B for that experiment.

In addition to the *experiment state*, the second factor that also plays a role in determining on how to move an experiment is the *experiment usage* within the current Fog zone. Of course, we do not want to abandon other devices that still in the current Fog zone because simply another device has moved. At the same time, we do not want to ignore the device that has moved to a different Fog zone. Because of this, we designed two types of experiments: *Primary* and *Mirror* experiments. *Mirror Experiment* is a cached-in experiment on a foreign Fog to serve devices within that foreign Fog zone. However, the *Primary Experiment* is the main experiment that exist on the home Fog to serve devices within the home Fog and to keep track of its mirror experiments on other foreign Fogs.

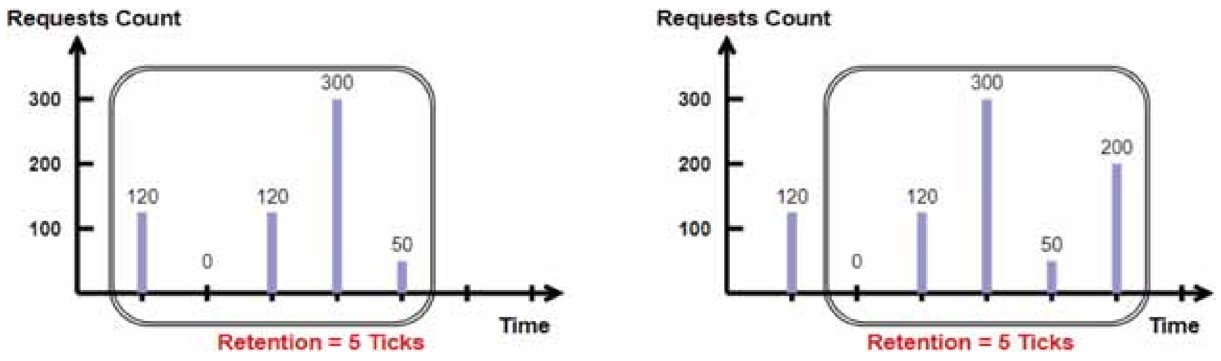


Figure 6. Measuring experiment usage.

It is worth noting that for obvious scalability and performance reasons, the experiment does not directly track connected devices. However, we measure the *experiment usage* factor by tracking how the experiment has recently been used (by devices) from the current Fog. In our approach, as shown in Figure 6, each experiment counts received requests from devices within its home zone at each tick (i.e., default is 5 minutes) over a retention period (i.e., default is 12 ticks). For example, Figure 6 shows an experiment with retention of five ticks, hence, the retention box moves one tick at a time, which always maintains five ticks in the retention box. Figure 6(left) shows 590 requests in the retention box. Now, on the next tick, the retention box moves one more tick to contain 670 requests, as shown in Figure 6(right).

Experiment usage factor (Figure 6) is mainly used to determine when to create *mirror* experiments on other Fogs, as discussed later. It further determines which experiment should be the *primary* experiment and which ones should be the *mirror* experiments throughout the Fogs. In our case, the primary experiment periodically performs *voting* among all its mirror experiments by asking them to report their usage metric (Figure 6). If a mirror reports usage more than the primary, this mirror experiment becomes the primary one.

In addition to *experiment state* and *experiment usage*, the third factor in determining how to move an experiment is the *experiment type* (i.e., primary or mirror) upon a request receipt from a device located outside its Fog zone.

3.3. Travelling experiment cases

As discussed above, the required taken actions on how to move an experiment is based on the combinations of multiple conditions. Those cases are summarised in Table 1. The taken actions based on those cases are carried out by number of components in the control plan (shown in Figure 7) via exchanging XML control messages using RESTful API.

Figure 7 shows the control plan components. In Figure 7, *Fog Mobile Controller* is a single component per Fog that contains location information about all primary experiments on its local Fog. *Cloud Mobile Controller* is the component that contains location information about all primary experiments on all Fogs. It further contains information about Fogs geographical zones.

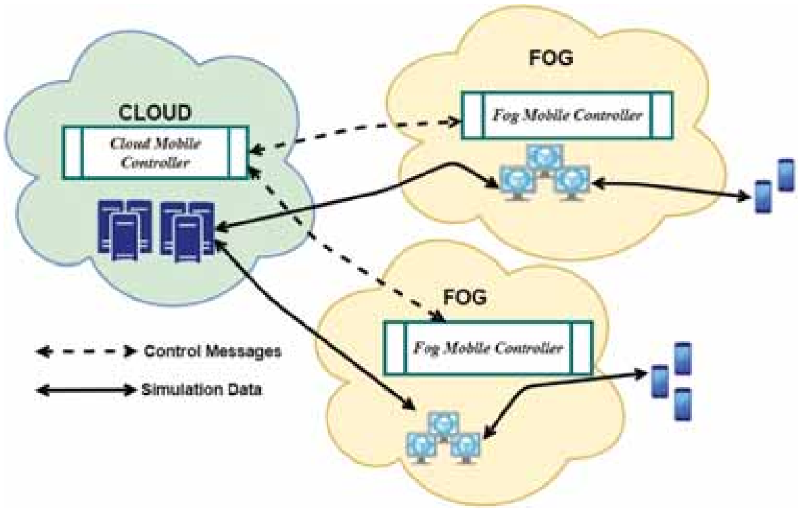


Figure 7. Control plan components.

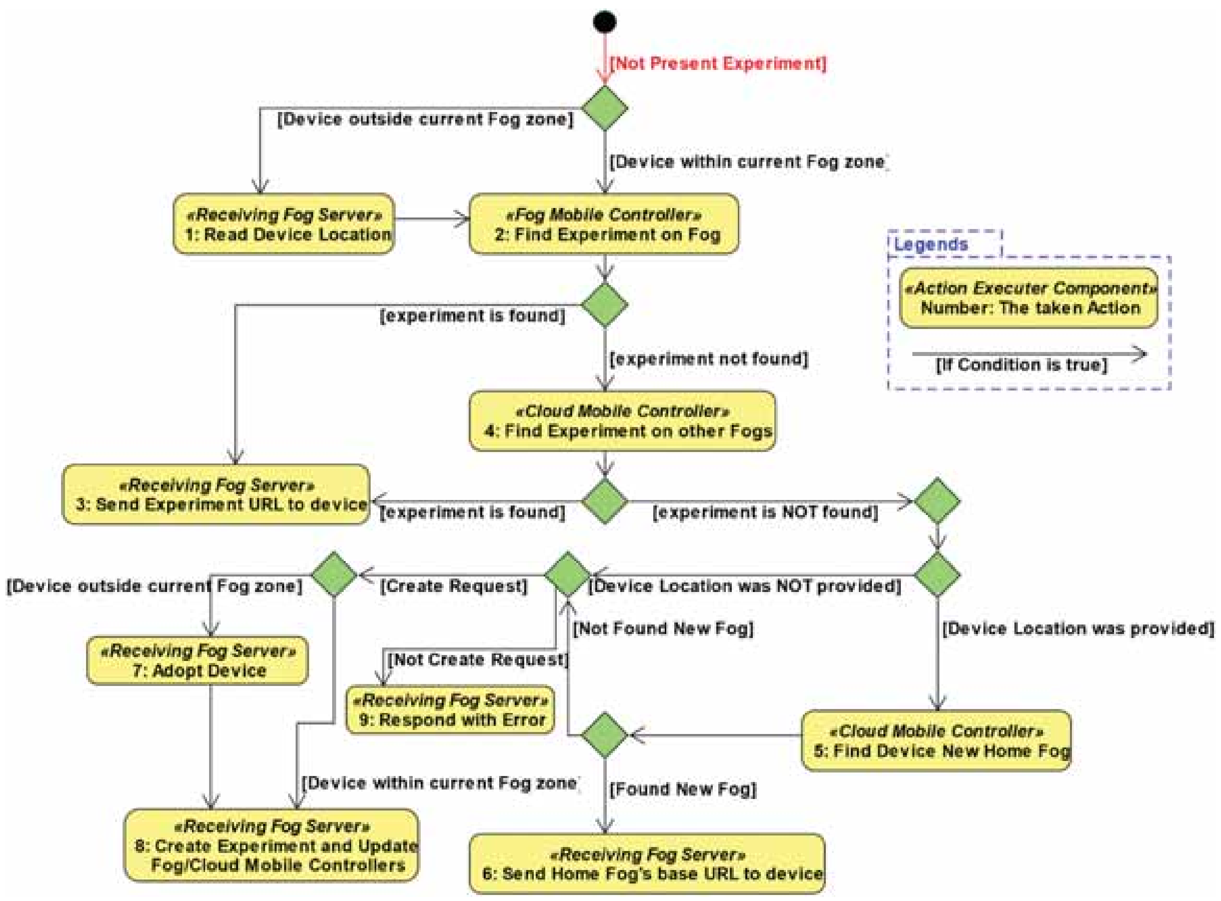

[Not Present Experiment]
[Device outside current Fog zone]
[Device within current Fog zone]
«Receiving Fog Server» 1: Read Device Location
«Fog Mobile Controller» 2: Find Experiment on Fog
Legends
«Action Executer Component» Number: The taken Action
[If Condition is true]
[experiment is found]
[experiment not found]
«Cloud Mobile Controller» 4: Find Experiment on other Fogs
«Receiving Fog Server» 3: Send Experiment URL to device
[experiment is found]
[experiment is NOT found]
[Device outside current Fog zone]
[Create Request]
[Device Location was NOT provided]
[Not Found New Fog]
[Device Location was provided]
«Receiving Fog Server» 7: Adopt Device
[Not Create Request]
«Receiving Fog Server» 9: Respond with Error
«Cloud Mobile Controller» 5: Find Device New Home Fog
[Device within current Fog zone]
[Found New Fog]
«Receiving Fog Server» 8: Create Experiment and Update Fog/Cloud Mobile Controllers
«Receiving Fog Server» 6: Send Home Fog's base URL to device

Figure 8. Not-present experiment use case.

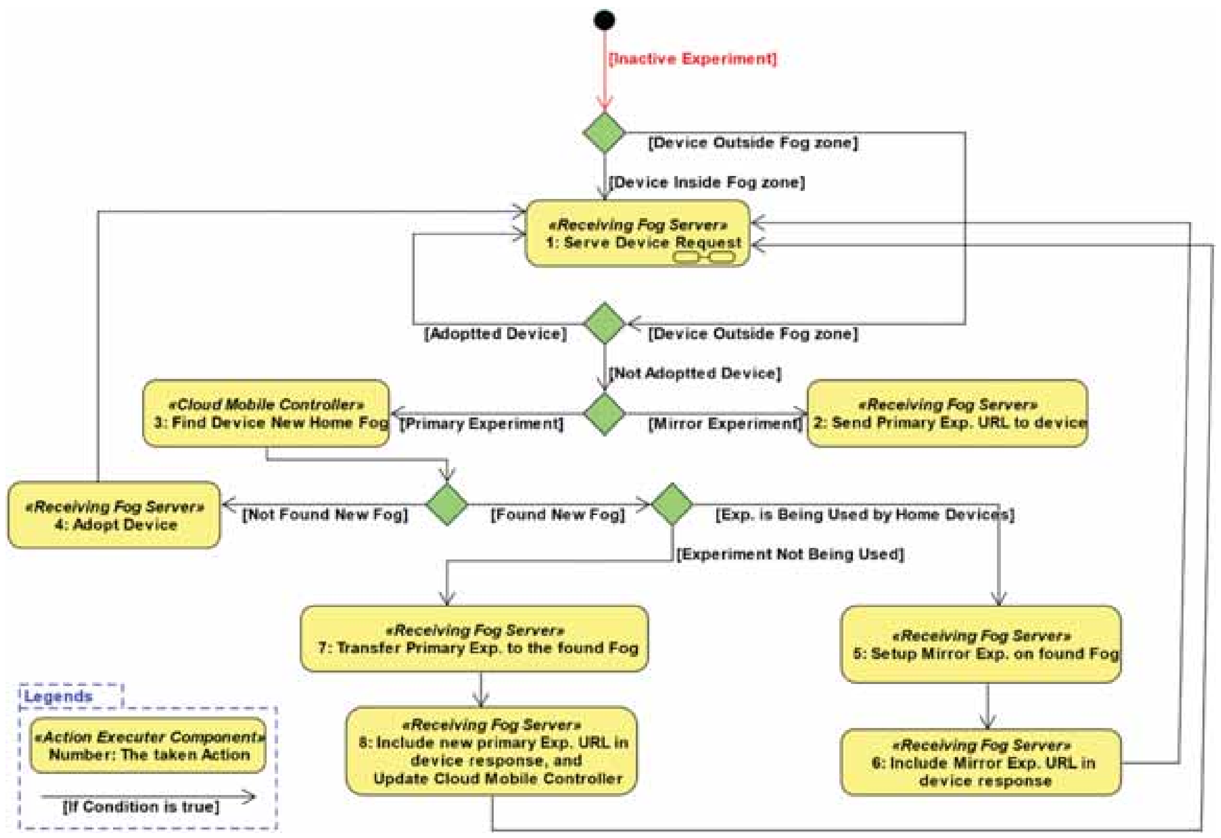

[Inactive Experiment]
[Device Outside Fog zone]
[Device Inside Fog zone]
«Receiving Fog Server» 1: Serve Device Request
[Adoptted Device]
[Device Outside Fog zone]
[Not Adoptted Device]
«Cloud Mobile Controller» 3: Find Device New Home Fog
[Primary Experiment]
[Mirror Experiment]
«Receiving Fog Server» 2: Send Primary Exp. URL to device
«Receiving Fog Server» 4: Adopt Device
[Not Found New Fog]
[Found New Fog]
[Exp. is Being Used by Home Devices]
[Experiment Not Being Used]
«Receiving Fog Server» 7: Transfer Primary Exp. to the found Fog
«Receiving Fog Server» 5: Setup Mirror Exp. on found Fog
Legends
«Action Executer Component» Number: The taken Action
[If Condition is true]
«Receiving Fog Server» 8: Include new primary Exp. URL in device response, and Update Cloud Mobile Controller
«Receiving Fog Server» 6: Include Mirror Exp. URL in device response

Figure 9. Inactive experiment use case.

Table 1 summarises the required taken actions based on the experiment state and experiment type. These cases are discussed next. Note that as previously stated that the Fog server term in our discussion is multiple servers with the same floating IP address.

3.3.1. Case 1: Handling "not present" experiments

Case #1 (in Table 1) is the situation when a request is received to an experiment that does not exist on the receiving Fog server. However, before this server can decide on what to do about this request, the server needs to know if this experiment does not actually exist or indeed exist somewhere else. This situation manages the possibility of an old device that is unaware of deleting the subject experiment or unaware of moving it to different Fog. The taken actions based on *Case #1* (in Table 1) are illustrated in Figure 8.

As shown in Figure 8, if a request is received from a device that is located outside the current Fog zone to an experiment that does not exist, the Fog server reads the device location (i.e., *Action #1*, Figure 8) and sends *Find Experiment* request to the *Fog Mobile Controller*. The *Find Experiment* is an XML message sent via HTTP POST method to the *Mobile Controller* URL, which includes information like the subject experiment URL and device location.

The *Fog Mobile Controller* (i.e., *Action #2*, Figure 8) tries to lookup this experiment info based on the owner's username. If found on local Fog, the new experiment URL is returned to the Fog server. The Fog server (i.e., *Action #3*, Figure 8) then returns this new URL to the device to retry the request again. However, if the experiment still not found on the Fog, the request is then forwarded to the *Cloud Mobile Controller* to check if another Fog has this experiment.

Upon requesting receipts on the Cloud, the *Cloud Mobile Controller* (i.e., *Action #4*, Figure 8) tries to lookup it up based on owner's username. If experiment is found, the experiment new URL is then sent back until it reaches the receiving Fog server. The Fog server (i.e., *Action #3*, Figure 8) then returns this new URL back to the device to retry the request again.

However, if the experiment still not found and the device location is known, the *Cloud Mobile Controller* (i.e., *Action #5*, Figure 8) finds a new Home Fog for the mobile device based on its location and then responds back with the found Fog base URL. When this response reaches the Fog server, it updates (i.e., *Action #6*, Figure 8) the device with the new home Fog URL to retry the request again with the newfound Fog. Otherwise, if cloud could not find a new home Fog for this device, the *Cloud Mobile Controller* replies with "New Fog Not Found" response.

Once the "New Fog Not Found" response is received by the Fog server, the Fog server (i.e., *Action #8*, Figure 8) creates the experiment and updates the *Fog Mobile Controller* with the experiment creation event, which in turn updates the *Cloud Mobile Controller*. This causes experiment to go into the *Inactive* state (as discussed in Figure 4). However, if the device original request was other than creation, the Fog server (i.e., *Action #9*, Figure 8) responds to the device with an HTTP error.

It is worth noting that the Fog server allows experiments creation for devices outside its zone if the *Cloud Mobile Controller* was not able to find a Fog for this device. In this case, the Fog server (i.e., *Action #7*, Figure 8) temporary adopts this device. This means that the device UUID is cached in the experiment in a list called adopted devices for some configured grace time (i.e., 6 hours by default). The device is removed from this adopted list when the grace time expires for this device. After that, if a request is received from this device and is still outside the Fog zone, the taken actions will be according to Table 1 listed cases.

3.3.2. Case 2: Travelling experiments during inactive state

Case #2 (Table 1) deals with the case when a request is received by an experiment in the *Inactive* state. The taken actions based on this case are illustrated on Figure 9. During the *Inactive* state (see Figure 4), mobile devices use experiment URIs to read information (e.g., cached simulation results) or to update experiment (e.g., simulation model), hence, simulation is being executed.

As shown in Figure 9, if the experiment receives a request from a device *inside* the current Fog zone, it simply serves the device request (i.e., *Action #1*, Figure 9). The steps for serving the device request are illustrated in Figure 10. In this case, if the request does not change the experiment, the request is granted and response is sent back to the device (i.e., *Action #1*, Figure 10). However, if this request changes the experiment settings, the next action depends on the experiment type. If this experiment is mirror, the request is then forwarded to the primary experiment to deal with it (i.e., *Action #2*, Figure 10). On the other hand, if the experiment is primary, it grants the request and updates all mirror experiments, if any (i.e., *Action #3*, Figure 10), apply changes, and replies to the device (i.e., *Action #4*, Figure 10).

To continue with Figure 9, when a request is received from a device *outside* the current Fog zone, in addition the device is currently being adopted, the experiment simply serves the device request (i.e., *Action #1*, Figure 9), as previously discussed in Figure 10.

However, if the device is not adopted, the next action depends on the experiment type. If this experiment is mirror, the request is forwarded to the primary experiment to deal with it (i.e., *Action #2*, Figure 9).

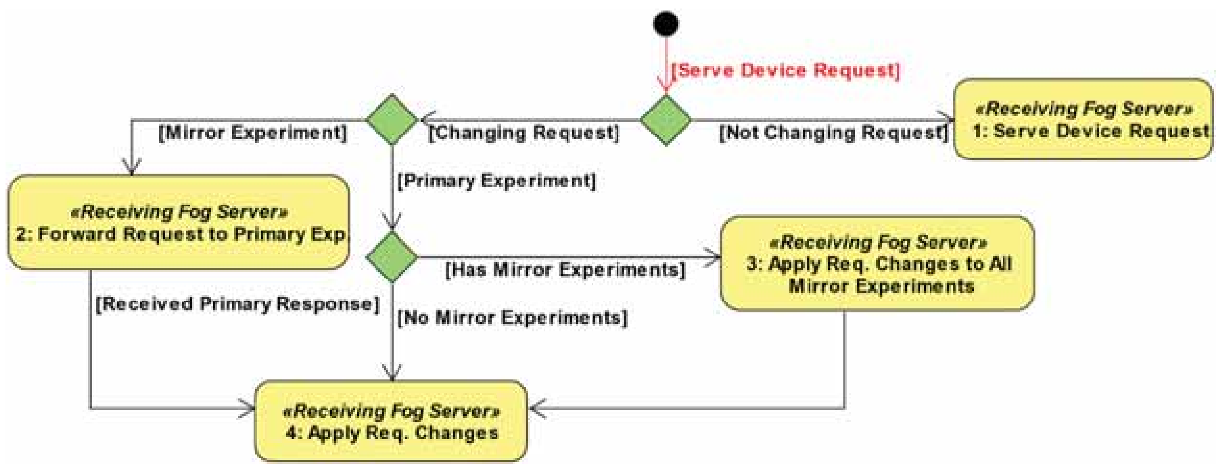

[Serve Device Request]
[Mirror Experiment]
[Changing Request]
[Not Changing Request]
«Receiving Fog Server» 1: Serve Device Request
«Receiving Fog Server» 2: Forward Request to Primary Exp
[Primary Experiment]
[Has Mirror Experiments]
«Receiving Fog Server» 3: Apply Req. Changes to All Mirror Experiments
[Received Primary Response]
[No Mirror Experiments]
«Receiving Fog Server» 4: Apply Req. Changes

Figure 10. Serving-received-request composite state (see Figures 9 and 11).

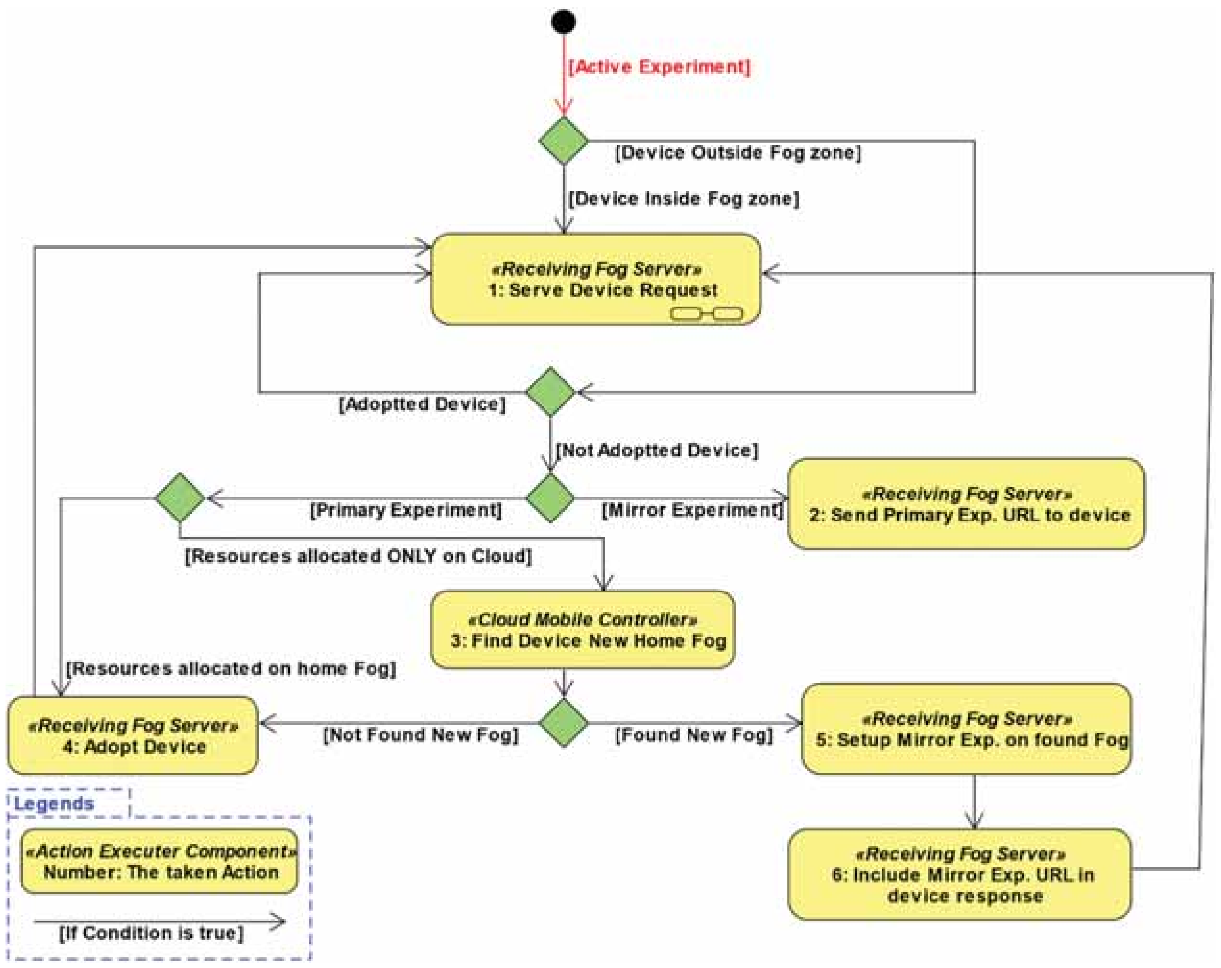

[Active Experiment]
[Device Outside Fog zone]
[Device Inside Fog zone]
«Receiving Fog Server» 1: Serve Device Request
[Adoptted Device]
[Not Adoptted Device]
[Primary Experiment]
[Mirror Experiment]
«Receiving Fog Server» 2: Send Primary Exp. URL to device
[Resources allocated ONLY on Cloud]
«Cloud Mobile Controller» 3: Find Device New Home Fog
[Resources allocated on home Fog]
«Receiving Fog Server» 4: Adopt Device
[Not Found New Fog]
[Found New Fog]
«Receiving Fog Server» 5: Setup Mirror Exp. on found Fog
«Receiving Fog Server» 6: Include Mirror Exp. URL in device response
Legends
«Action Executer Component» Number: The taken Action
[If Condition is true]

Figure 11. Active experiment use case.

However, if the experiment is primary, a request is sent to the *Cloud Mobile Controller* to find a new home Fog for this device based on its location (i.e., *Action #3*, Figure 9). If the new Fog is not found, the Fog server temporary adopts this device (i.e., *Action #4*, Figure 9), and serves the device request (i.e., *Action #1*, Figure 9), as previously discussed in Figure 10.

On the other hand, if the new Fog was found for this device while this primary experiment has not recently been used (see Figure 6), the primary experiment is then transferred to the new Fog (i.e., *Action #7*, Figure 9). It further (i.e., *Action #8*, Figure 9) inserts the new primary experiment URL in the device response (to be used by the device in future requests), updates the *Cloud Mobile Controller*, and serve the original device request (i.e., *Action #1*, Figure 9), as previously discussed Figure 10.

In contrast, if the new Fog was found for this device while this primary experiment has recently been used (see Figure 6), a mirror is then created on the new Fog (i.e., *Action #5*, Figure 9). It further (i.e., *Action #6*, Figure 9) includes the new mirror experiment URL in the device response (to be used by the device in future requests) and serve the original device request (i.e., *Action #1*, Figure 9), as previously discussed Figure 10.

3.3.3. Case 3: Travelling experiments during active state

Case #3 (Table 1) deals with the case when a request is received by an experiment in the *Active* state (see Figure 4). In this state, simulation is running in the experiment, hence, M&S resources are allocated and being used by the experiment. During this state, mobile devices use experiment RESTful API (on the nearby Fog) to manipulate running simulation, where simulation could be distributed over allocated resources on Fog, Cloud, or both.

As shown in Figure 11, if the experiment receives a request from a device *inside* the current Fog zone, it simply serves the device request (i.e., *Action #1*, Figure 11), as previously discussed in Figure 10. It also serves the device request (i.e., *Action #1*, Figure 11) for adopted devices even if they are located *outside* the Fog zone. Otherwise, the next action then depends on the experiment type.

If mirror experiment, the request is then forwarded to the primary experiment to deal with it (i.e., *Action #2*, Figure 11). However, if this experiment is primary and the allocated resources to execute the simulation is fully or partially running on the home Fog, the device is temporarily adopted by this Fog (i.e., *Action #4*, Figure 11), and the device original request is served (i.e., *Action #1*, Figure 11).

However, if the simulation allocated resources only exist on the cloud, a request is sent to the *Cloud Mobile Controller* to find a new home Fog for this device (i.e., *Action #2*, Figure 11). If Fog is not found, the device is then adopted (i.e., *Action #4*, Figure 11) and the original request is served (i.e., *Action #1*, Figure 11), as previously discussed in Figure 10. However, if a new Fog is found for this device, a mirror experiment is setup on that found Fog (i.e., *Action #5*, Figure 11), inserts the mirror experiment URL with the response to the device (i.e., *Action #6*, Figure 11), and the original request is served (i.e., *Action #1*, Figure 11).

4. Use cases

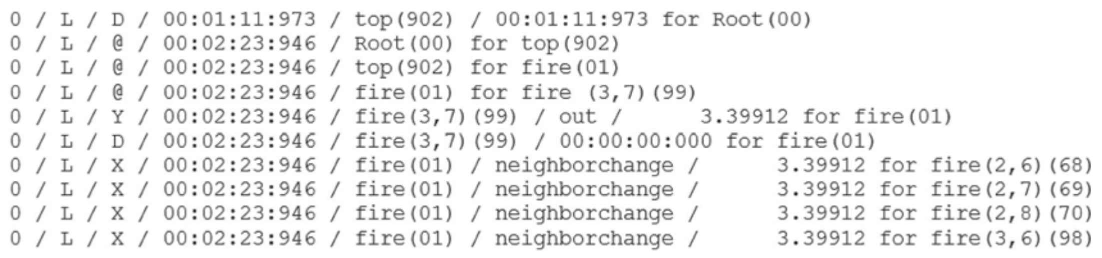
This section main purpose is to show how users view simulation results on their mobile apps. In this case, the mobile App retrieves results from the experiment resource on the Fog/Cloud side. This could happen during live simulation using URI … */simulation*, or after simulation have been completed via URI … */results* to download cached results for replay on client side, as previously discussed in Figures 4 and 5. Thus, in this case, the mobile app is used to view (and visualise) the simulation results while the actual simulation is executed on one or more middlewares that are installed on VMs across the Fog/Cloud platform. It is worth noting that the experiment view on the client side sees any experiment on the Fog/Cloud side as URI that begins with the Ingress server URI (called <Fog-Server-base-URL> in Figure 4), which is the only part of an experiment URI that gets updated when it moves from a Fog to another. Of course, the Fog/Cloud platform mechanism is the one that makes the decision to move an experiment from a Fog to another (as discussed earlier), by simply overriding the experiment URI on the mobile side. This RESTful API mechanism simplifies the client side and relieves it from many complex details on the Fog/Cloud side.

To this end, mobile Apps (on mobile devices) are RESTful clients, allowing users to access their experiments through the nearby Fog according to a defined RESTful API. Experiment framework related API has already been discussed in Figure 4 (in Section 3.2). As previously discussed, experiment URIs are URI templates, which means they have the same URI structure but with two variables: *{username}* and *{experiment name}*. This ensures experiment URI uniqueness for the same user and across different users. This also allows users to list and view their existence experiments and switch between those experiments.

Now, when a device needs to send to a request to an experiment, it builds the experiment URI string according to the URI template string (see Figure 4). The differences between experiments URIs are the *{username}*, the *{experiment name}*, and the *<starting base URI>*. The *<starting base URI>* needs to be attached at the beginning of the URI string to reach the services. This basically is the Fog entry URI that previously discussed. Of course, each Fog has a unique entry URI; hence, in our case we use the IP addresses to ensure this uniqueness. Thus, if an experiment is moved to another new Fog, the old Fog embeds the new Fog entry URI in its regular response to the device, allowing future requests to go to the new home Fog. This is smooth transition since RESTful requests are stateless, which means requests are independent of each other and contain all necessary information to conduct the service.

Based on above, mobile Apps do not require simulation engines to be installed on devices. However, they need to know how to parse simulation results to visualise the simulation. In our case, those results are received as text format (e.g., Figure 12).

Therefore, from the mobile App viewpoint, it owns the experiment by itself regardless of the number of other devices communicating with the same experiment. It further assumes the experiment is fixed in one



```
0 / L / D / 00:01:11:973 / top(902) / 00:01:11:973 for Root(00)
0 / L / @ / 00:02:23:946 / Root(00) for top(902)
0 / L / @ / 00:02:23:946 / top(902) for fire(01)
0 / L / @ / 00:02:23:946 / fire(01) for fire (3,7)(99)
0 / L / Y / 00:02:23:946 / fire(3,7)(99) / out /      3.39912 for fire(01)
0 / L / D / 00:02:23:946 / fire(3,7)(99) / 00:00:00:000 for fire(01)
0 / L / X / 00:02:23:946 / fire(01) / neighborchange /      3.39912 for fire(2,6)(68)
0 / L / X / 00:02:23:946 / fire(01) / neighborchange /      3.39912 for fire(2,7)(69)
0 / L / X / 00:02:23:946 / fire(01) / neighborchange /      3.39912 for fire(2,8)(70)
0 / L / X / 00:02:23:946 / fire(01) / neighborchange /      3.39912 for fire(3,6)(98)
```

Figure 12. Snippet of simulation results sent from Cloud to the Android App.

location that is reachable through the constructed URI.

For the remaining of this section the simulation will be shown on the handheld device Apps (i.e., client side) using two platforms: Android and Windows phone. The Android mobile App is developed in Java for the Android phone platform while the Windows phone platform is built in C#. Further, the actual simulation on the *Fog* side will be conducted using the CD++ simulation tools (Wainer, 2009), that is provided via RESTful-based middleware (Al-Zoubi & Wainer, 2013).

4.1. Forest fire simulation on android

As a case study, we present a model to simulate forest fires. In this model, fire propagation considers certain parameters such as heat, density, fuel type, vegetation size, wind speed, terrain topology and humidity. The GPS and compass values (obtained from the mobile device sensors) are used to advise fire fighters for the best way to move.

Figure 12 shows an example of the forest fire simulation results sent from the Fog to the App. The App then draws these results (i.e., fire propagation) on top of Google Maps (see Figure 13). The mobile App visualises the fire propagation to advise fire fighters of the best way to move in case of being too close to a dangerous area.

Figure 13 shows the simulation and Google Maps visualisation. It also shows the simulation major control buttons: *Play, Pause* and *Recommend* Direction. The first two buttons are used to start and stop the animation playback. The third button (*Recommend Direction*) calculates the user position with respect to the fire location and suggests the direction the user should go.

Figure 14(right) shows the state of the screen once the *Recommend Direction* button is pressed, which is waiting for GPS response. Figure 14 (middle) shows a "go Ahead" direction suggestion, while Figure 14(left) shows a "turn left" direction suggestion. The way that this Android App manages visualisation is as follows: The mobile App sends a request to the experiment (via nearby Fog) to read simulation results. Upon results receipt, the

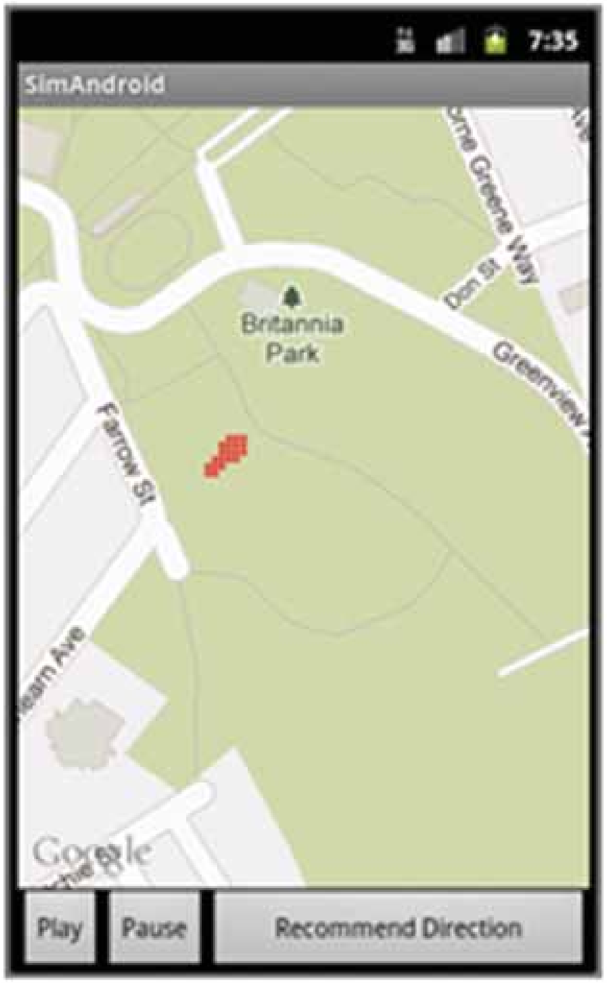


Figure 13. Forest fire simulation visualisation on android device.

App extracts the time, the fire spread rate, and the coordinate position. The coordinate position is then converted to geographic location. This conversion is done by method *getProjection()* in the *MapView* class (this class is part of Google Android API (Google APIs for Android, 2020)). Afterwards, the cells that contain fire are converted to red cells (see Figures 13 and 14). This overlay code is sub-classed from class *Overlay* in package *com.google.android.maps* (a part of Google Android API (Google APIs for Android, 2020)). This also used to draw the direction arrows shown on Figure 14. The *Time* class (see Google Android API (Google APIs for Android, 2020)) is used to

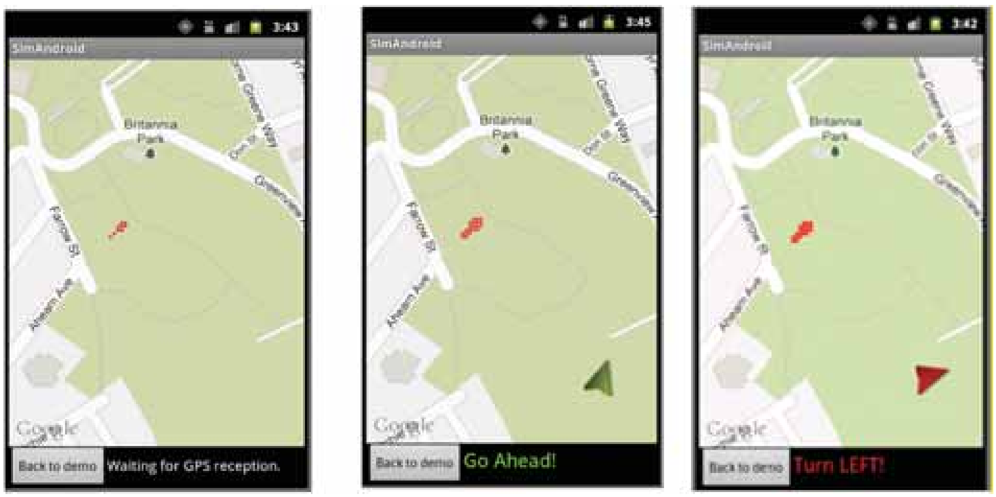


Figure 14. Forest fire simulation visualisation with direction recommendation on android device.

provide fire spreading animation. This also allows the simulation to be played faster (or slower) than the real-time simulation.

Further, when the *Recommend Direction* is pressed, the app gets position updates from GPS, and gets orientation updates from the compass. With each update, the distance between the user and nearby cells with fire (i.e., red cells on the screen) are calculated. Based on this, it calculates the angle between the handheld device direction (as indicated by the compass), and the vector starting at the closest cell with fire (as indicated by simulation and GPS position), as shown in Figure 15. Now, if this angle is less than 30 degrees (i.e., an arbitrarily chosen value), the app suggests moving ahead for the user to flee the fire. Otherwise, it suggests turning right or left, depending on whether the angle is positive or negative.

The *bearingTo()* method (in the *Location* class in package *android.location* (Google APIs for Android, 2020)) returns α angle in Figure 15. Further, the Compass angle is obtained via the *SensorManager* class (Google APIs for Android, 2020), which continuously updated with any change of the handheld device orientation. This information is used to determine β in Figure 15, which is the device orientation around the Z axis. The *GeomagneticField* class is used to compute δ in Figure 15, where it is the magnetic declination from true north. With those three variables, the application then calculates the angle between the line from the closes fire cell to the GPS location and the compass, as shown in Figure 10. As a result, the angle of the arrow (Figure 14) is given by (α-(β + δ)). If result is zero, the direction of the user (i.e., given by compass) is moving away from the fire (i.e., given by simulation and GPS), hence the arrow becomes green. The arrow rotates right or left as a reaction to the user rotating the device around the Z-axis. Likewise, the arrow turns red if the arrow angle gets larger than 30.

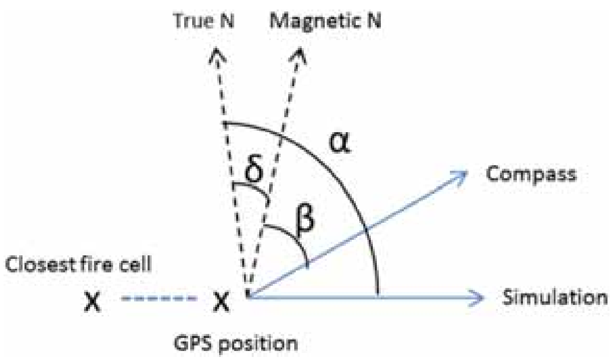


Figure 15. Recommended direction calculation.

4.2. Ant Colony simulation on windows phone

Ant Colony model shows the behaviour of ants in an environment with random obstacles and food locations. Thus, the model studies how ants find the hidden food locations, and how they find their way back to the nest once a food location is found.

Ants live in what is called *superorganism*, which means that ants interact with each other to define the overall behaviour of the entire colony. This is shown in the way they search for food, avoid obstacles, choose the location of their nest, and in the way they find the shortest path to go back to their nest.

In this model, ants search randomly by smelling nearby food. They further mark their path by leaving behind chemical traces called pheromones, allowing them to find their way back to the nest (Dorigo, 2006). Further, the closer ants get to previously found food, the stronger pheromones substance smell becomes. In doing so, ants eventually form lines between the nest and food locations (Dorigo, 2006).

Figure 16 shows a snippet from the simulation results produced by the CD++ simulation engine. The “Y” result lines represent the visual aspects of the experiment where ant(x, y) represent the next

```
0 / L / D / 00:00:00:000 / ant(0,0)(02) / 00:00:00:000 for ant(01)
0 / L / Y / 00:00:00:000 / ant(0,1)(03) / out /      0.00000 for ant(01)
0 / L / D / 00:00:00:000 / ant(0,1)(03) / 00:00:00:000 for ant(01)
0 / L / Y / 00:00:00:000 / ant(0,2)(04) / out /      0.00000 for ant(01)
0 / L / D / 00:00:00:000 / ant(0,2)(04) / 00:00:00:000 for ant(01)
0 / L / Y / 00:00:00:000 / ant(0,3)(05) / out /      0.00000 for ant(01)
0 / L / D / 00:00:00:000 / ant(0,3)(05) / 00:00:00:000 for ant(01)
0 / L / Y / 00:00:00:000 / ant(0,4)(06) / out /      0.00000 for ant(01)
0 / L / D / 00:00:00:000 / ant(0,4)(06) / 00:00:00:000 for ant(01)
0 / L / Y / 00:00:00:000 / ant(0,5)(07) / out /      0.00000 for ant(01)
```

Figure 16. Snippet of simulation results sent from Cloud to the Mobile App.

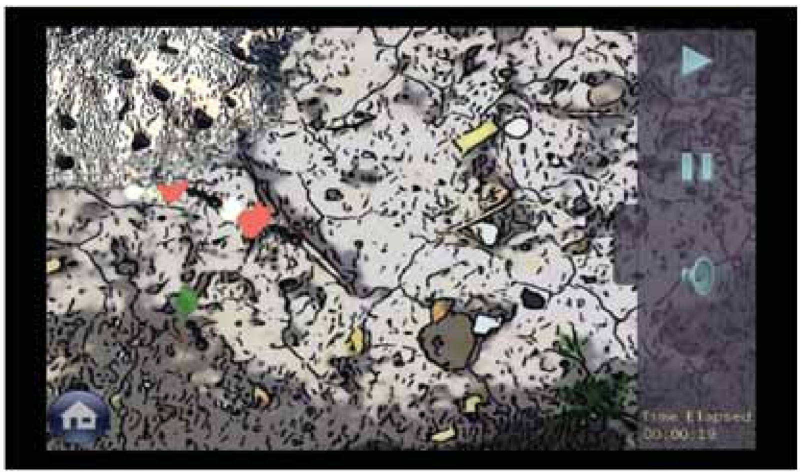


Figure 17. Windows phone visualisation for Ant Colony simulation.

coordinate, and the value between the parentheses "()" represents the encoded data. This encoded data indicates cell state information, such as pheromones strength in the cell, ant direction, amount of food, the load that an ant carries, and so on.

Figure 17 shows the Windows Phone visualisation based on received simulation results (Figure 16) from the Fog. The App uses Microsoft's XNA game studio framework to visualise the simulation results. In Figure 17, the pink spots and the leaf symbolise the food, while the yellow rectangles symbolise the pheromone that ants leave behind.

4.3. Cancer spread simulation on windows phone

This model (Figure 18) studies the spread of cancer cancerous cells in tissues. The simulation rules decide on new cancer cells being created or dead. In this model, the simulation starts with the initial values defined in the input initial values (i.e., CD++ VAL file (Wainer, 2009)). The Cell-DEVS dimensions for this model was defined as 20 by 20 where they need to be fit on 480 × 800 pixel screen. To resolve this issue, the screen was cropped so that an area of 480 by 480 was used, which was done by multiplying each position by 24 and then subtracting 160 out of the y coordinate. Figure 18 shows a snippet of the cancer simulation spreading in tissues.

4.4. Fire spreading simulation on windows phone

The Fire model is used to predict how a fire spread. This model considers certain parameters such as heat, minerals, density, fuel type, vegetation size, wind speed, territory inclination and humidity. In this model, the user can speed up or slow down the simulation. The user can also pause the simulation, zoom in and zoom out to get a better idea about the location of the fire.

This model has different parts to it: the *Bing Map*, the *User Control buttons*, and the *Fire Spread* state. The *Bing Map* was loaded using Silverlight. The Bing Map uses touch controls, zoom levels, and several types of map views (Aerial and Road). The *User Control buttons* (i.e., Play, Pause, Fast Forward, Speed and Sound Effects) were also loaded using Silverlight. This allows users to start, pause, fast forward, simulation sound and change the fire spread speed. The *Fire Spread* cells were constructed using XNA. These cells have used a polygon to adapt to the fire shape and a gradient colour was used to fill in the polygon. These polygons are pined onto the map (drawn on the map using geo coordinates) where the size can change by zooming in or out. Each polygon represents an area of 1 km^2 on the map.

The fire spread state is drawn based on the received simulation results from the running experiment on the

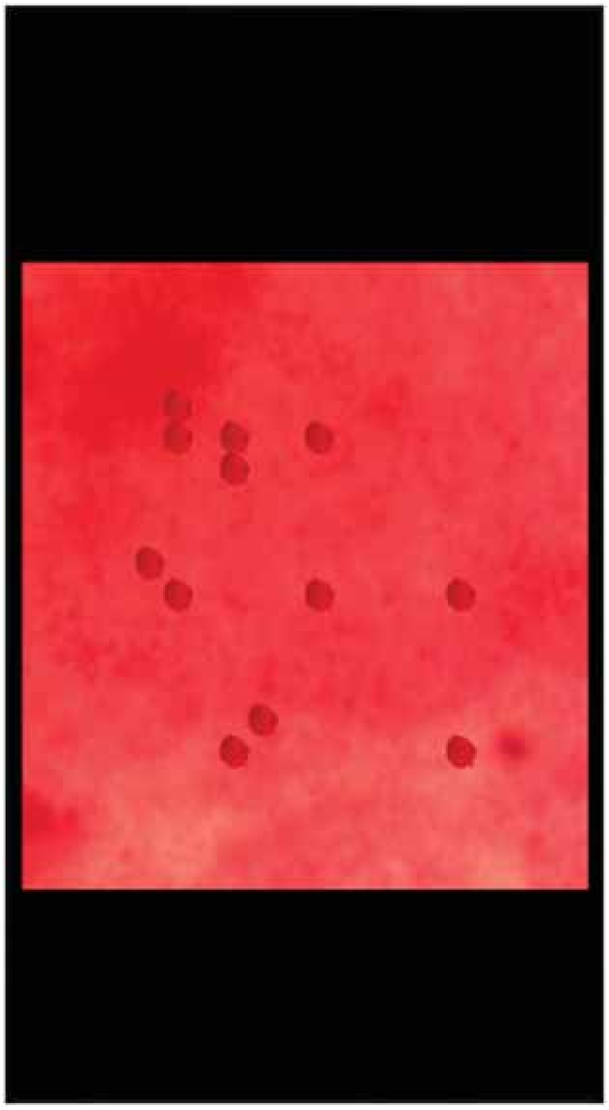


Figure 18. Windows phone visualisation for cancer spread simulation.

Fog. For example, the following line was taken from the simulation results:

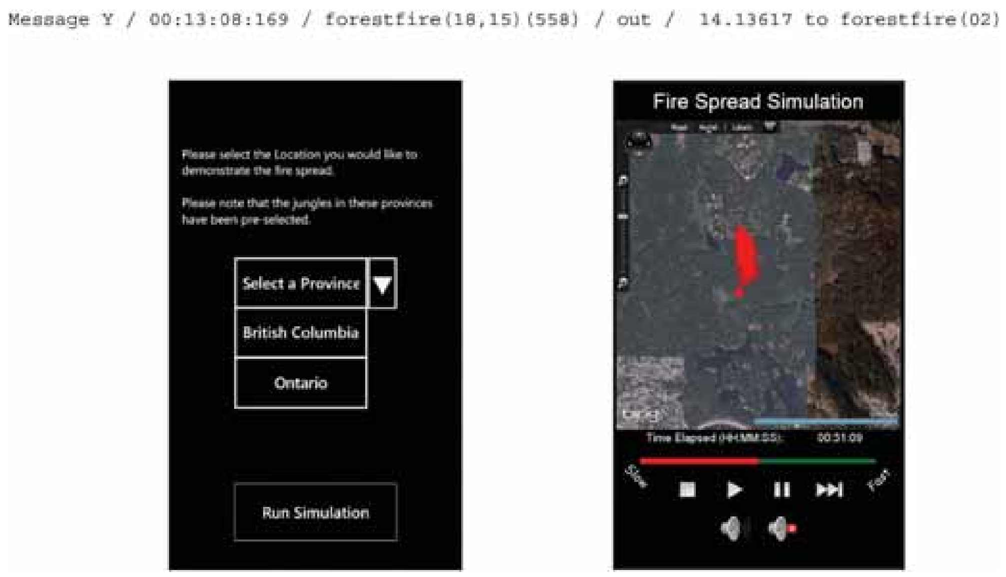
Message Y/13:08:169/forestfire(18,15)(558)/out/14.13617 to forestfire(02)

The first part of this line shows the message type and simulation time. The coordinates represent the location of the fire. These coordinates (x, y) are then converted to Geo Coordinates (on the Bing Map) by the mobile app as follows: (x/3-130 + 0.25, y/3 + 40). The "Y" message value (which is 558 in the above example) is then converted and scaled to fit within the range of the gradient colour factors by the mobile app.

This model targets Canadian forests as shown in Figure 19(left). Figure 19 (right) shows a snippet of a running simulation over a forest in British Columbia, Canada. As the figure shows, the user can slow, fast, zoom in, zoom out and pause the simulation.

5. Mobility support evaluation

This section focuses on the evaluation for the mobility support aspects of the overall Fog/Cloud system as it is within the scope of this paper. However, it is worth noting that our work in (Al-Zoubi & Wainer, 2020a) have fully evaluated the performance and scalability of the Fog/Cloud collaboration algorithms that dealt with the resources management, and with the M&S discovery & selection to run experiments on best and compatible resources (that could be allocated somewhere on the Fogs or Cloud backbone). In addition, the scalability of those algorithms on large size was additionally evaluated in our work described in (Al-Zoubi & Wainer, 2020b).



```
Message Y / 00:13:08:169 / forestfire(18,15)(558) / out /  14.13617 to forestfire(02)
```

Please select the Location you would like to demonstrate the fire spread.

Please note that the jungles in these provinces have been pre-selected.

Figure 19. Windows phone visualisation for forest fire simulation.

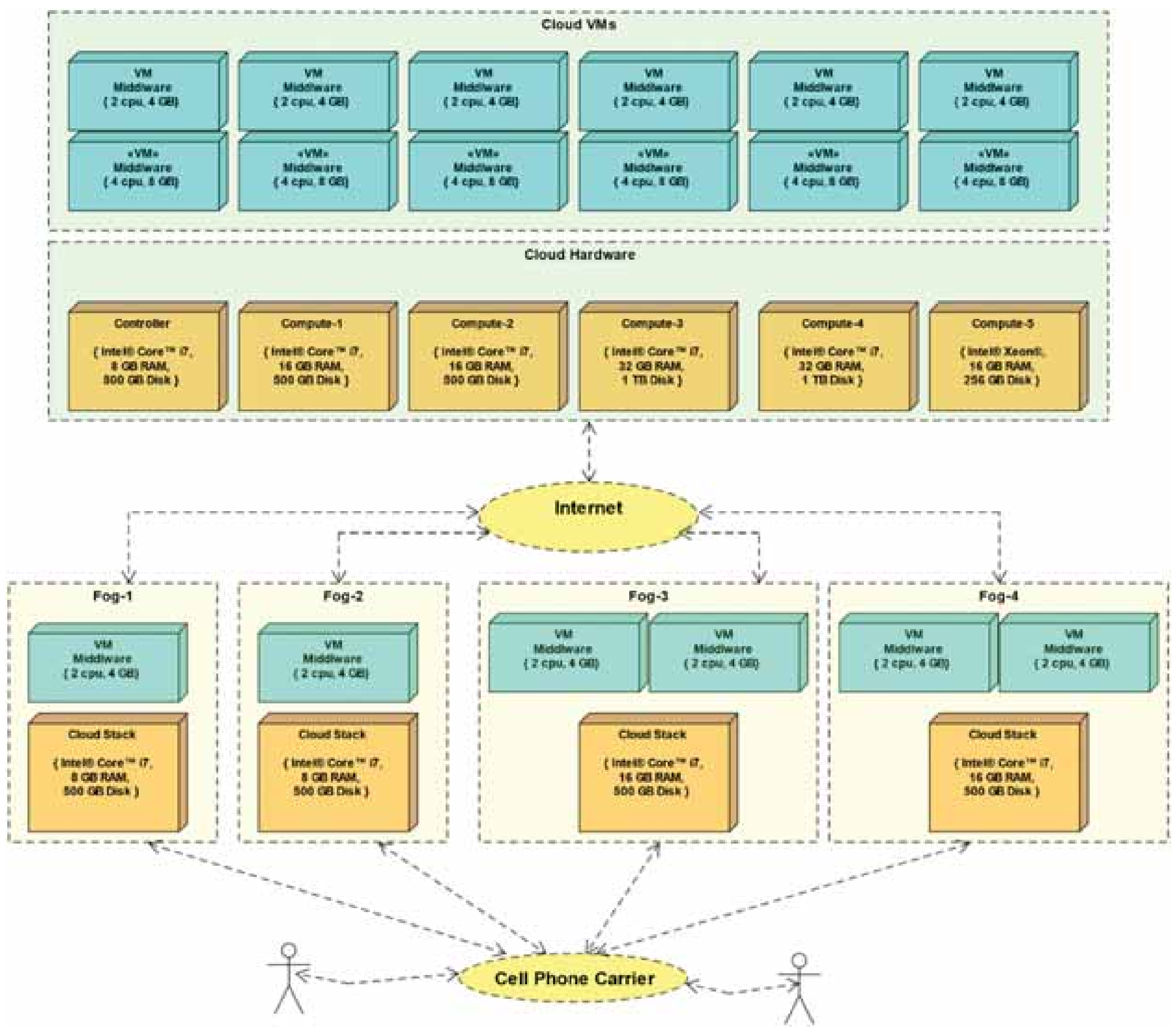

Cloud VMs
VM Middleware (2 cpu, 4 GB)
«VM» Middleware (4 cpu, 8 GB)
Cloud Hardware
Controller (Intel® Core™ i7, 8 GB RAM, 500 GB Disk)
Compute-1 (Intel® Core™ i7, 16 GB RAM, 500 GB Disk)
Compute-2 (Intel® Core™ i7, 16 GB RAM, 500 GB Disk)
Compute-3 (Intel® Core™ i7, 32 GB RAM, 1 TB Disk)
Compute-4 (Intel® Core™ i7, 32 GB RAM, 1 TB Disk)
Compute-5 (Intel® Xeon®, 16 GB RAM, 256 GB Disk)
Internet
Fog-1
Fog-2
Fog-3
Fog-4
VM Middleware (2 cpu, 4 GB)
Cloud Stack (Intel® Core™ i7, 8 GB RAM, 500 GB Disk)
Cloud Stack (Intel® Core™ i7, 16 GB RAM, 500 GB Disk)
Cell Phone Carrier

Figure 20. Real system setup & configuration.

The mobility support is evaluated here with respect to the followings: (1) Changing Fogs for a moving mobile device according to the previously defined scenarios in Section 3.3 (this discussed soon in Figures 22 and 23), (2) balancing executed simulation tasks between Fogs and the Cloud backbone (this discussed soon in Figure 25), and (3) the response time for clients served by the nearby Fogs and the Cloud backbone (this discussed soon in Figure 24).

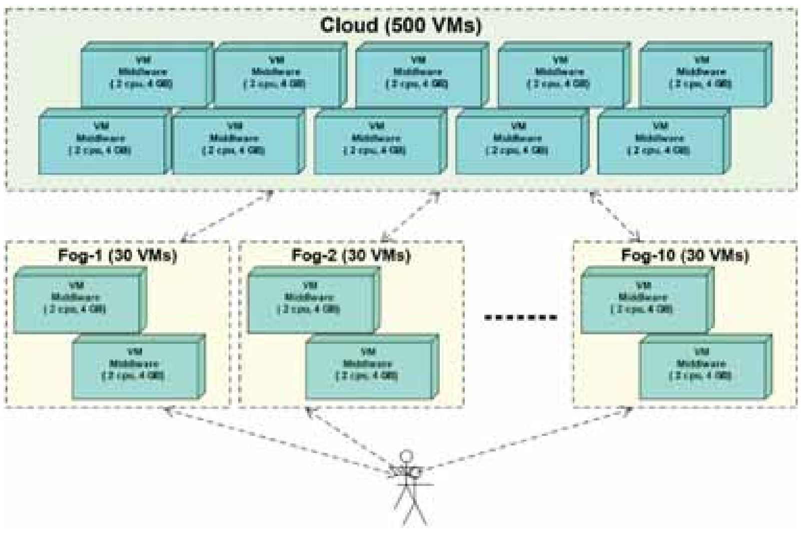


Figure 21. Simulated system setup & configuration.

Figure 20 shows the real physical and configuration setup that we used to collect the presented results in this section. As can be seen that the actual system is like a container that you can put into it your available hardware, and accordingly create your middleware (i.e., which are created quickly via OpenStack since they are based on a VM image that is already had all required software installed on it). This comes from our knowledge as we experienced with many different setups like the shown one in Figure 20. However, as can be seen we were always faced with two practical constraints: the available computing hardware for us, comparing to real-world clouds, and (2) the geographical area size. For example, the setup, shown in Figure 20, spreads within a single city and its suburbs, which then places Fogs and Cloud in a range of 50–60 km from each other.

Because of this, in addition to the real system setup (in Figure 20), we will be using our simulated setup shown in Figure 21 to study some issues on larger scale and wider geographical area, as needed. This model was built and simulated by CD++ to study our Fog/

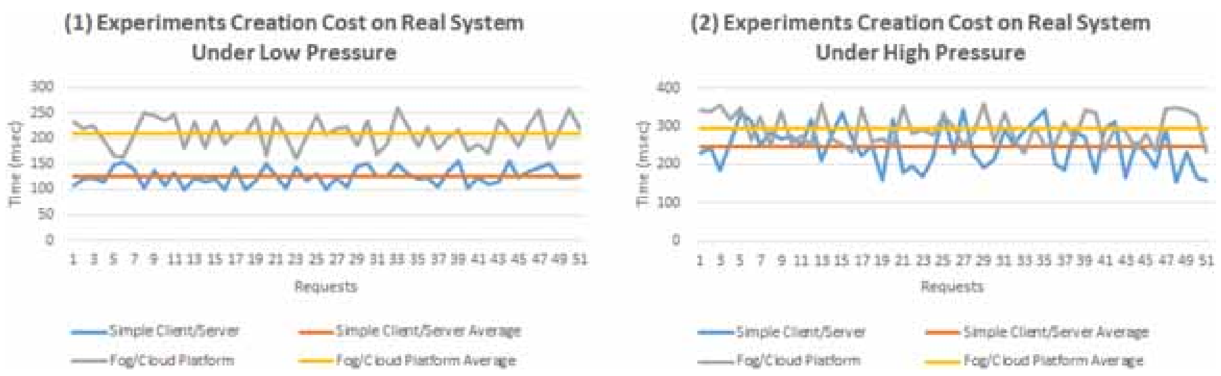


Figure 22. Example of experiment creation cost using real system setup.

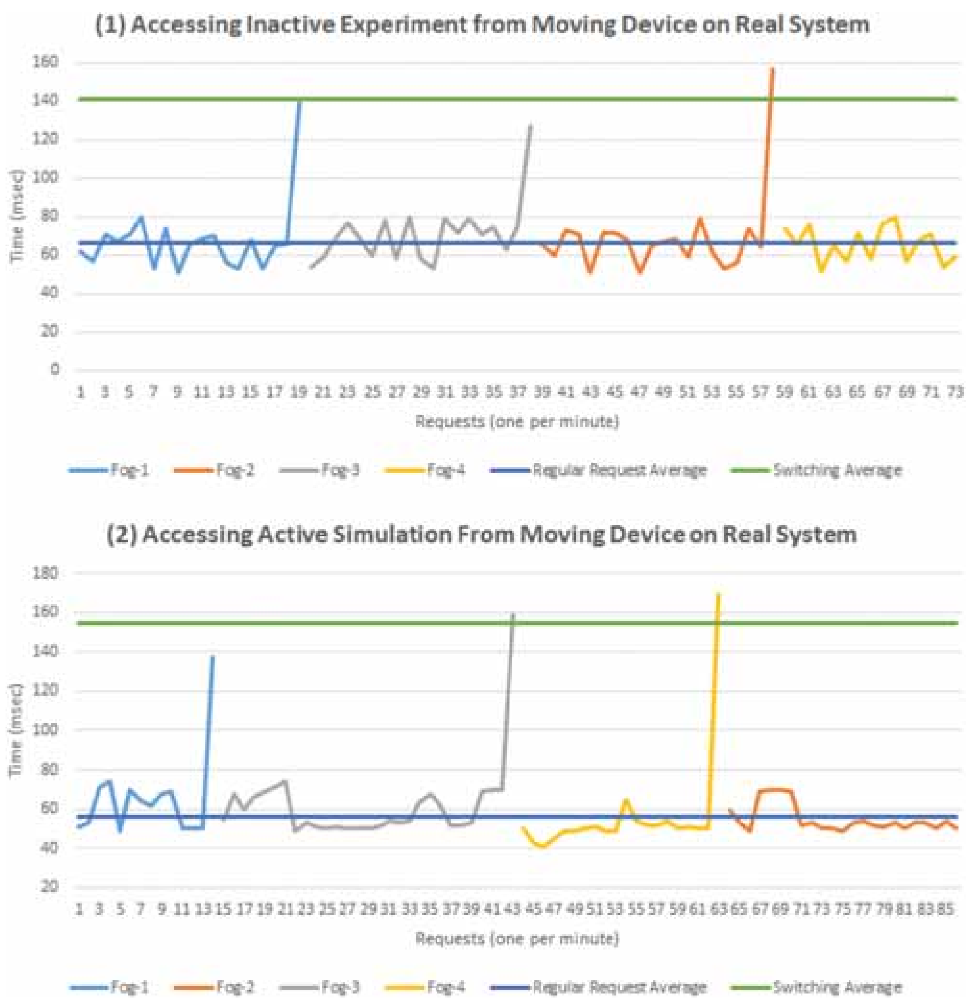


Figure 23. Example of mobility support during active/inactive experiment.

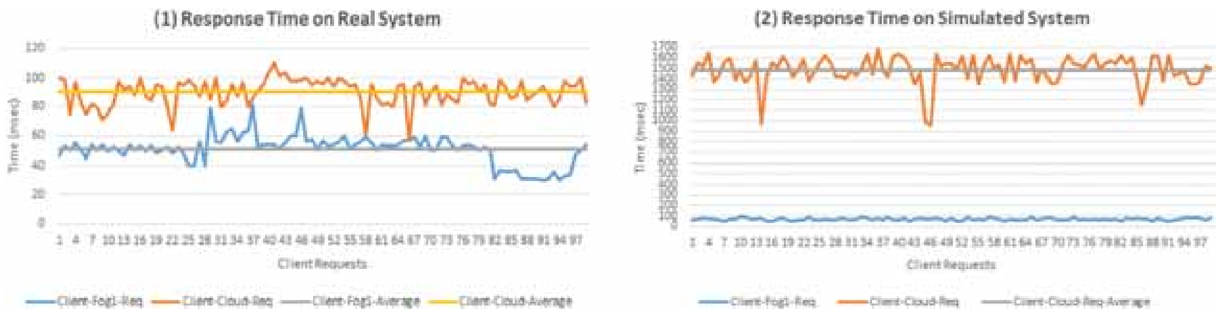


Figure 24. Example of Fogs/Cloud client response time.

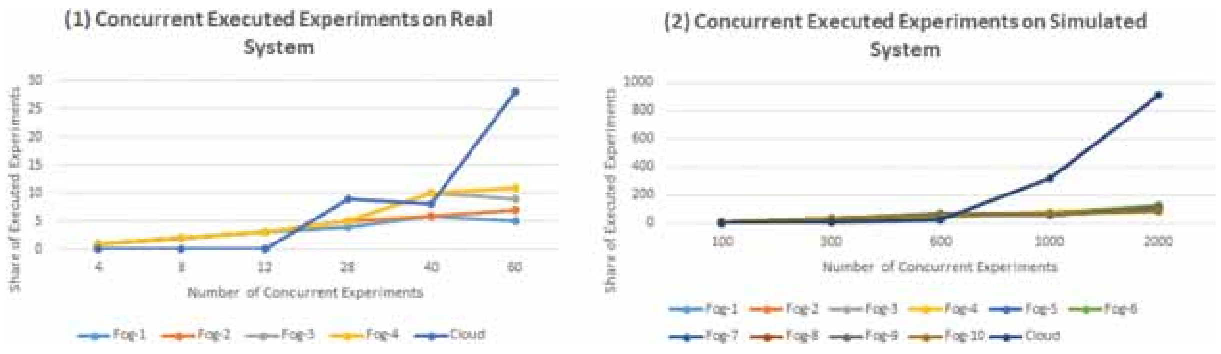


Figure 25. Example of balancing simulation tasks between Fogs and Cloud.

Cloud platform on large scale. The full details of this model and its validation was presented in our work in (Al-Zoubi & Wainer, 2020b).

To this end, the presented results were gathered using two environment setups: **(1)** The real system setup (shown in Figure 20) contains the Cloud backbone (with 12 VMs and 6 physical machines), created, and controlled by OpenStack (as previously discussed in Figure 1). It also contains four Fogs, each of the Fogs contains one physical machine; hence, this machine contains the entire OpenStack solution. Each of Fog 1 and 2 contains one VM while each of Fog 3 and 4 contains two VMs. Finally, all middleware's (VMs) on the Fogs and Cloud have exposed the CD++ simulation services. Further, each experiment has run with the fire forest model (discussed in Section 4). **(2)** The simulated system setup (shown in Figure 21) contains a cloud with 500 VMs and 10 Fogs where each contains 30 VMs. Fogs and Cloud are simulated with 1000s of kilometres apart.

In the first defined scenario in Section 3.3, when a client (Figure 8) sends a request to an experiment that does not exist, the receiving Fog server assumes that the subject experiment may exist on some other Fog. This could happen for many reasons like a device has outstanding experiments in one of its old Fogs. Therefore, our proposed solution has added extra cost to verify an experiment existence, comparing to a simple client/server system when a server has all needed information locally. Figure 22 (using the real system setup) looks at this added cost by considering the case of creating new experiment, hence performing *step #8* action in Figure 8. Figure 22 compares the simple client/server system to the Fog/Cloud platform. In case of the simple server, the decision is local, and the experiment URI is accordingly created. However, in the Fog/Cloud platform with mobility support, the Fog server must first make sure the experiment does not exist elsewhere before honouring the request. As can be seen in Figure 22-1, there is a cost of about 100 msec to verify that an experiment does not exist under low load pressure (comparing to the simple server case). This cost gets a little bit larger under high pressure (Figure 22-2), but the gap is also got closer between the two systems. This tells us that this value has maintained its stability while staying at a low level comparing to the low-pressure case.

To study the system when a mobile device changes Fog zones, a mobile device was placed in a travelling car where the car path was intentionally planned to cross through the four Fog zones of the real system setup. The mobile periodically transmitted a simple request each minute to check the experiment status, hence, the device also embedded its current location in those requests, allowing the contacted Fog to know if this device is within its zone or outside. This test was repeated twice: one for inactive experiment (i.e., an experiment with no running simulation as discussed in Figure 9), and the other one for active experiment (i.e., an experiment with running simulation as discussed in Figure 11). Figure 23-1 shows the case of inactive experiment. Specifically, the previous discussed case of *step #7* & *step #8* in Figure 9 whereas the experiment is transferred to the new Fog. Figure 23–2 shows the case of active experiment. Specifically, the previously discussed case of *step #5* & *step #6* in Figure 11 where a mirror experiment gets created on the found Fog (to avoid actual simulation migration). As can be seen on both cases (in Figure 23), the spike (of about 70–100 msec) in the last request before changing the Fog is because of the time to find the new Fog and experiments transfer coordination between Fogs and the Cloud. The current Fog responds to the device and slips the new Fog URI in the response. After this, the next request transmission will go through the new Fog. It is worth mentioning that the actual transfer of experiment settings and files occurs concurrently with the device response to reduce delay.

Figure 24 compares requests response time when clients served by the Fog or when served by the Cloud backbone. Figure 24-1 shows the response time using the real system setup. In this test, the client has sent about 100 requests to Fog 1 (from its zone) and repeated those requests to the cloud backbone. In our case, clients are not allowed to contact the cloud directly for security reasons, but we forced it in this case to conduct this test. As can be seen, the gap is not big (i.e., about 40 msec) since those Fogs are relatively placed close to each other (within the same city). Because of this, we repeated this test using our simulated system. In this model, we placed Fogs and Cloud 1000s of kilometres apart. As expected, the difference has jumped substantially as shown in Figure 24-1. This case is more practical as Fogs are expected to be located closer to users while clouds could be located somewhere around the globe.

Figure 25 looks at balancing simulation tasks between the Fogs and the Cloud backbone. This is more related to the Fog/Cloud collaboration and scheduling methods that we presented in (Al-Zoubi & Wainer, 2020a) rather than the presented mobility support in this research. However, it is still important enough to not ignore. The idea here is to keep evenly adding simulation tasks onto the system via all Fogs. This means starting simulation experiments, which lead to allocating resources to run the simulation. Figure 25-1 shows how the real system behaved under these circumstances. As can be seen, Fogs tried to handle all tasks until a certain point. After that, the cloud backbone starts getting involved (i.e., at 28 experiments). At the end, Fogs reached their ability where the Cloud backbone got more involved. This behaviour was also shown in Figure 25-2 using the simulated system. This tells us, the more resources invested in the Fogs, the more capability they would have to serve clients without going to the Cloud. This is more important to know since Fogs are expected to be built from privately owned resources while the Cloud backbone are generally expected to be paid services on public clouds located somewhere around the world.

6. Conclusion

Indeed, providing Modelling and Simulation (M&S) as a service via clouds enables users to take advantage of such massive computing resources. However, this approach lacks true mobility support since clients normally contact the same centralised clouds regardless of their locations. This means that mobility support goes beyond simply having users consume services from the Cloud using their mobile devices (as in current M&S works). As discussed in this paper, mobility support means several things like services should be aware of clients' geo-locations (as in the cases of mobile devices), allowing them to consume services from nearby locations (to enhance local resources utilisation and quality of service). Fog computing comes with the promise of enhancing mobility support in Clouds, decentralising cloud resources, and of allowing users to utilise their local resources. Fog computing decentralises such cloud resources by building mini clouds, called Fogs, near to the users to perform full or partial computations on behalf of the main cloud backbone. This lends itself to enhance clouds mobility support and quality of service. However, as we previously discussed, M&S researchers have so far approached the Fog computing technology by developing models or complete simulation tools to simulate Fog and Cloud related issues.

In this paper, we have built a complete Fog/Cloud infrastructure out of privately owned resources. In this system, each VM contains a RESTful middleware that can expose different simulation environments. This means when a user (via mobile App) starts an experiment, the nearby contacted Fog need to find the appropriate simulation environments to run the subject experiment. We used mobile Apps to conduct experiments through nearby Fogs that discover needed M&S resources to run that experiment. Mobile Apps visualise returned results once received. We fully presented the novel concept of the *mobile simulation experiment*: if a device moves away from its home Fog, the experiment moves with the device to a closer Fog. We further showed several use cases that we conducted using actual mobile devices over an actual existing *Fog* system.

We finally presented evaluation for those presented methods. We have showed how such methods can work in practice and how experiments can change Fogs when executed by a moving a mobile device (i.e., placed in a moving car). We further studied the response time when experiments conducted on the nearby Fog comparing to the Cloud backbone. These results showed that the geographical distance can affect the clients' response time. We furthermore showed how resources allocations are balanced between Fogs and the Cloud backbone. These results showed that tasks are executed on nearby Fogs until a certain point, which then offloaded to the cloud backbone. This means the more capabilities you put into the Fogs; the less tasks are offloaded to the cloud backbone. This is important to know since Fogs are expected to be privately owned while Clouds might be privately owned or leased on public clouds.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Khaldoon Al-Zoubi http://orcid.org/0000-0001-6194-3449

References

Alsaffar, A., Pham, H. P., Hong, C. S., Huh, E. N., & Aazam, M. (2016). An architecture of IoT service delegation and resource allocation based on collaboration between Fog and Cloud computing. *Mobile Information System*, *2016 (1)*, 1–15. https://doi.org/10.1155/2016/6123234

Al-Zoubi, K., & Wainer, G. (2013). RISE: A general simulation interoperability middleware container. *Journal of Parallel and Distributed Computing. Elsevier*. *73*(5), 580–594. https://doi.org/10.1016/j.jpdc.2013.01.014

Al-Zoubi, K., & Wainer, G. (2015, June). Distributed simulation of DEVS and cell-DEVS models using the RISE middleware. *Simulation Modelling Practice and Theory. Elsevier*, *55*, 27–45. https://doi.org/10.1016/j.simpat.2015.03.010

Al-Zoubi, K., & Wainer, G. (2020a, May). Fog and Cloud collaboration to perform virtual simulation experiments. *Simulation Modelling Practice and Theory. Elsevier*, *101*, 102032. https://doi.org/10.1016/j.simpat.2019.102032

Al-Zoubi, K., & Wainer, G. (2020b, December). Modelling Fog & Cloud collaboration methods on large scale. In *The IEEE Proceedings of the Winter Simulation Conference*. IEEE Press. https://doi.org/10.1109/WSC48552.2020.9384058

Amoretti, M., Grazioli, A., & Zanichelli, F. (2015, November). A modeling and simulation framework for mobile cloud computing. *Simulation Modelling Practice and Theory*. *Elsevier*, *58*, 140–156. https://doi.org/10.1016/j.simpat.2015.05.004

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the Internet of Things. In *The Proceedings of the first MCC Workshop Mobile Cloud Computation* (pp. 13–16). Association for Computing Machinery (New York).

Carlo, P., Diogo, G., Lopes, M., Martins, L., Madeira, E., Mingozzi, E., Rana, O., & Bittencourt, L. (2020). MobFogSim: Simulation of mobility and migration for fog computing. *Simulation Modelling Practice and Theory*. *Elsevier*. *101*. https://doi.org/10.1016/j.simpat.2019.102062

CloudSim. (2020). Retrieved December, 2020, from http://www.cloudbus.org/cloudsim/

dev.m, Dorigo M., Birattari M., & Stutzle T. (2006). Ant colony optimization. Computational Intelligence Magazine. IEEE. 1(4), 28–39, Nov. 2006, doi:10.1109/MCI.2006.329691

Gai, S. (2020). *Building a future-proof cloud infrastructure: A unified architecture for network, security, and storage services*. Addison-Wesley.

Giang, N. K., Leung, V. C. M., & Lea, R. (2016). On developing smart transportation applications in fog computing paradigm. In *The Proceedings of the 6th ACM Symposium on Development and Analysis of Intelligent Vehicular Networks and Applications* (p. 2016). Association for Computing Machinery (New York).

Google APIs for Android. (2020). Google. Retrieved December, 2020, from https://developers.google.com/android/reference/packages

Guan, S., Grande, R., & Boukerche, A. (2016). An HLA-based cloud simulator for mobile cloud environments. In *The Proceedings of the 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE (Piscataway, NJ).

Gupta, H., Dastjerdi, A., Ghosh, S., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Software: Practice and Experience (SPE)*, *47*(9), 1275–1296. https://doi.org/10.1002/spe.2509

Hou, X., Li, Y., Chen, M., Wu, D., Jin, D., & Chen, S. (2016). Vehicular fog computing: A viewpoint of vehicles as the infrastructures. *IEEE Transactions on Vehicular Technology*, *65*(6), 3860–3873. https://doi.org/10.1109/TVT.2016.2532863

Hu, P., Dhelim, S., Ning, H., & Qiu, T. (2017). Survey on Fog computing: Architecture key technologies applications and open issues. *Journal of Network and Computer Applications*, Elsevier, *98*, 27–42. https://doi.org/10.1016/j.jnca.2017.09.002

Ju, M., Kim, H., & Kim, S. (2016). MofySim: A mobile full-system simulation framework for energy consumption and performance analysis". In *The proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE.

Kim, Y., Bae, J., Lim, J., Park, E., Baek, J., Han, S., Chu, C., & Han, Y. (2018). 5G K-simulator: 5G System simulator for performance evaluation. In *The Proceedings of the IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE.

Mahmud, R., Koch, F. L., & Buyya, R., (2018). Cloud-fog interoperability in IoT-enabled healthcare solutions. In *Proceedings of the 19th International Conference on Distributed Computing and Networking. (ICDCN)*. ACM (Association for Computing Machinery, New York).

Malik, A., Kashif, B., Malik, S., Zahid, Z., Aziz, K., Kliazovich, D., Ghani, N., Khan, S., & Buyya, R. (2017). CloudNetSim++: A GUI based framework for modeling and simulation of data centers in OMNeT++. *Services Computing IEEE Transactions*, *10*(4), 506–519. https://doi.org/10.1109/TSC.2015.2496164

Naha, R., Garg, S., Georgakopoulos, D., Jayaraman, P., Gao, L., Xiang, Y., & Ranjan, R. (2018). Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, *6*, 47980–48009. https://doi.org/10.1109/ACCESS.2018.2866491

Ostermann, S., Plankensteiner, K., Prodan, R., & Fahringer, T. (2010). GroudSim: An event-based simulation framework for computational grids and clouds. In *The Proceedings of the Euro-Par 2010 Parallel Processing*

Workshops. Springer. https://doi.org/10.1007/978-3-642-21878-1_38

Pooranian, Z., Shojafar, M., Naranjo, P., Chiaraviglio, L., & Conti, M. (2017). A novel distributed fog-based networked architecture to preserve energy in fog data centers. In *The Proceedings of the 14th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE.

Rahmani, A., Gia, T., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., & Liljeberg, P. (2018). Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: A fog computing approach. *Future Generation Computer Systems*, Elsevier,*78*, 641–658. https://doi.org/10.1016/j.future.2017.02.014

Rehman, K., Kipouridis, O., Karnouskos, S., Frendo, O., Dickel, H., Lipps, J., & Verzano, N. (2019). A cloud-based development environment using HLA and Kubernetes for the co-simulation of a corporate electric vehicle fleet. In *The Proceedings of the IEEE/SICE/SICE International Symposium on System Integration (SII)*. IEEE.

Shekhar, S., Abdel-Aziz, H., Walker, M., Caglar, F., Gokhale, A., & Koutsoukos, X. (2016). A simulation as a service Cloud middleware". *Annals of Telecommunications*, *71* (3), 93–108. https://doi.org/10.1007/s12243-015-0475-6

Taneja, M., & Davy, A., (2017). Resource aware placement of IoT application modules in fog-cloud computing paradigm. In *The Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE.

Wainer, G. (2009). *Discrete-event modeling and simulation: A practitioner's approach*. CRC/Taylor & Francis.

Yang, L., Liu, B., Cao, J., Sahni, Y., & Wang, Z. (2017). Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. In *The Proceedings of the IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE.

Yi, S., Li, C., & Li, Q. (2015). A survey of Fog computing: Concepts applications and issues. In *The proceedings of the Workshop Mobile Big Data*. ACM (Association for Computing Machinery).