Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

MODELLING FOG & CLOUD COLLABORATION METHODS ON LARGE SCALE

Khaldoon Al-Zoubi

Faculty of Computer & Information Technology
Jordan University of Science & Technology
JORDAN

Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
CANADA

ABSTRACT

Fog Computing is expected to decentralize clouds to improve users quality of service, in technologies like the Internet of Things, by pushing some computing onto mini-clouds (called Fogs) located close to end users. To enhance M&S with this concept, we have developed a complete Fog/Cloud collaboration methods to conduct simulation experiments: Users manipulate their experiments though nearby Fogs servers while M&S resources are dynamically discovered and allocated throughout the Fogs/Cloud. We have already built those methods using privately owned clouds. However, it was difficult in practice to study those methods scalability using real system setups. As a result, we present here the simulation *model* that we developed to mimic this real system in order to study the proposed *collaboration* methods on large scale. This model was validated with reference to the real system. The results have clearly shown the *scalability* of those proposed methods at the *structure* and *coordination* levels.

1 INTRODUCTION

Fog computing (Bonomi et al. 2012) has recently been gaining popularity mainly because of its promising support for technologies like the Internet of Things, in which countless of daily used mobile devices by people like phones, vehicles, sensors and health monitoring devices are being connected together via the Internet. This is because connecting those type of devices directly through the cloud can complicate certain issues mainly related to the way clouds work. Clouds are usually centralized and clustered in huge datacenters where devices regardless of their location always served by those datacenters. Therefore, issues like mobility support, service latency, and geographical-location awareness need to be solved to better serve cloud based applications (Bonomi et al. 2012). Fog computing is the cloud extension to help with those issues.

The main idea of Fog computing concept is to decentralize clouds by building close-to-users mini-clouds, called Fogs, to perform some computation locally. This is expected to enhance cloud issues like mobility support, service latency, geographical-location awareness, and utilizing local resources that are usually idle off-peak hours, as suggested by several surveys (Hu et al. 2017, Naha et al. 2018, Yi et al. 2015).

Based on above, to allow Modelling & Simulation (M&S) Cloud based systems to cope with such expected future concepts, they need then to support Fog computing technology. In this way, devices consume their services through nearby Fogs rather than going directly to the cloud backbone, as in the case of current M&S cloud-based systems.

In doing so, collaboration methods between distributed Fogs systems and the Cloud backbone play a major role in making all of those systems work together. This is because Fogs need to make the correct decisions on what to compute locally and what to offload to the cloud backbone. Thus, to allow M&S Cloud-based systems to involve Fog computing, Fog/Cloud collaboration methods need to exist first.

As a step to advance M&S field into this direction, we've developed the needed Fog/Cloud collaboration mechanisms to allow users to setup their experiments on Fog servers. After that, the experiment can dynamically discover and select the required resources to execute the simulation for those experiments. This workflow is summarized as follows: Clients create simulation experiments on Fog servers (i.e. services access points). Once simulation is started on an experiment, the experiment then (1) discovers the M&S resources that can execute the simulation (i.e. M&S resources are heterogeneous, hence only execute compatible models). (2) selects the best resources to execute the simulation (that could be anywhere on the Fog/cloud), and finally (3) the experiment directly execute simulation on the selected resources. We have built and verified this system using private Fogs and Clouds with various physical steps, as detailed in (Al-Zoubi et al. 2020).

Obviously, using the real system is good in a sense of achieving the real purpose of such system, in which to allow modelers to run their simulation experiments via nearby Fog servers without being concerned on how and where the required M&S resources can be found to execute the simulation.

However, because this system is expected in practice to operate on large scale, thus, it is necessary to study this system *scalability* when it gets large, since it is hard to prove this quality with real systems constructed out of 20 or 30 real servers.

Our main purpose here is to present the proof of that proposed Fog/Cloud collaboration methods that are indeed scalable on large Fog/Cloud environment.

To do so, we have developed a Discrete Event System Specification (DEVS) (Wainer 2009) based simulation model to mimic our real system on large scale. This model uses the same proposed Fog/cloud collaboration algorithms (i.e. same implementation) as they are the main purpose of developing this model to study their scalability on large scale. The presented DEVS model here was mainly based on real collected results from various physical setups of the real system. It was then validated by comparing it to the whole real system behavior or part of it. Based on this model, the simulation results have shown the scalability on two levels: at the level of the overall system structure and at the level of Fog/Cloud collaboration mechanisms.

It is worth noting that the presented work here is developed because of the need to study an existing Fog/Cloud platform on large scale. Thus, it was validated with respect to a real system. This distinguishes this work from other related works that study Fogs and Clouds based on general communications and job executions.

This paper is organized as follows: Section 2 discusses related works. Section 3 summarizes the Fog/Cloud collaboration proposed ideas using real constructed private Fogs and Clouds. Section 4 presents the simulation model discussion and its validation. Section 5 presents the simulation model on large scale and results. Conclusions presented in Section 6.

2 RELATED WORKS

As stated previously, our main purpose is to build shared environment to conduct simulation experiments as service using the Fog/Cloud concepts. Obviously, Fog/Cloud collaboration methods play essential role to put all distributed systems in one collaborated environment. Therefore, we proposed in (Al-Zoubi et al. 2020) the necessary Fog/Cloud collaboration methods to allow M&S resources to be discovered and used (within experiment framework) throughout the Fog/Cloud systems. These methods were developed and implemented using private clouds, as detailed in (Al-Zoubi et al. 2020). However, because it is complicated to study those collaboration methods using real constructed systems, the simulation model presented here main purpose is to study those methods on large scale. The presented model here is developed using the Discrete Event System Specification (DEVS) formalism and was executed using the CD++ (Wainer 2009) simulation environment.

It is worth noting that Fog/Cloud concept still not used to provide modelling & simulation experiments as services. Thus, the use of modelling and simulation to study Fog and Clouds becomes the only comparison aspect between the presented work here and other related works.

So far, existing works used simulation to study clouds quality of service and networking related issues. For example, CloudSim (CloudSim 2020), which simulates service provisioning, and network, CloudNetSim++ (Malik et al. 2017), which simulates datacenters energy consumption and communication. GroudSim (Ostermann et al. 2019), which simulates job executions and load distribution. Other works like (Etemad et al. 2017) simulates how fog computing could cut down service latency to enhance user experience. All of those mainly focused on services provisioning like VMs and networking, and on high volume communication latency.

In contrast, our presented work here main purpose is not to study clouds and Fogs in general as in the case of above existing works, but to study Fog/Cloud collaboration proposed methods in our real system on large scale. This is important because the presented simulation model is validated with respect to various real system setups collected data, including job execution, load distribution, communication, concurrent job executions, etc. On the other hand, related works are usually based on general communication data and general job executions. However, jobs like simulation jobs are extremely hard to estimate since any slight changes to experiment settings can affect the internal simulation events execution cycles, which affect the time it takes a job to complete.

3 BACKGROUND: REAL SYSTEM DEPLOYMENT AND MANAGEMENT

This section summarizes the proposed Fog/Cloud collaboration methods that were presented in (Al-Zoubi et al. 2020). Those methods assume that M&S resources are heterogeneous and arbitrarily spread out across all Fogs (i.e. mini clouds) and the main cloud backbone. Users interact with this system via nearby Fogs to setup and manipulate experiments. This solution is in conformance to the Fog/Cloud architecture proposed to the Internet of Things based applications (Bonomi et al. 2012).

In our case, when a user sends a request to the experiment (on a Fog) to start the simulation, the experiment *first* (with the help of a local component called *scheduler*) needs to *discover* the compatible M&S resources that can simulate the model understudy, hence models are only executable on specific simulation engine environment. Those discovered compatible resources might exist on the local Fog, other Fogs, or the cloud backbone. The second stage is to select the best resources to run the simulation. The deciding factors of this selection are the discovered resources location and computing load.

Those methods were implemented using real Fog/Cloud systems and were fully studied and presented in (Al-Zoubi et al. 2020). However, it was difficult to study them on large scale using real systems, which what we are doing in this paper using modelling and simulation.

Next, we present a brief background about those Fog/Cloud collaboration proposed methods while the full detail of those methods is presented in (Al-Zoubi et al. 2020).

The workflow starts when a user (via a client device) creates an experiment on nearby Fog. At this point, the user only communicates with the experiment (via nearby Fog) to perform any changes. Now, once the user sends a request to start the simulation, the experiment asks a component, called scheduler, on its local Fog server to (1) discover simulation assets that can execute the simulation, and to (2) select least loaded M&S resources, hence implicitly one or more servers will be selected to run the simulation. At this point, the experiment becomes an interplay unit between the client devices and the server(s) that executing the simulation. It is worth noting that selected server(s) could be the local Fog server itself or other server(s) located on the Cloud.

As can be seen, the scheduler component, which is located on every server, plays a major role in the overall Fog/Cloud collaboration. This is because the experiment depends on the scheduler (that resides on its Fog server) to discover compatible M&S resources, and then to select best servers to execute the simulation. This means schedulers on Fog servers must have global knowledge to make such decisions. Of course, these Fog/Cloud collaboration mechanisms need to provide Fog servers this global knowledge while respecting M&S resources heterogeneity and the overall scalability. This Fog/Cloud collaboration is briefly explained the next example.

Figure 1 shows an example of a Fog/Cloud physical setup in our platform. In our case, Fogs and Clouds are built as private clouds based on OpenStack. OpenStack (Shrivastwa et al. 2016) is an open-source cloud tool developed by NASA and Rackspace. This physical setup example (Figure 1) shows one Fog and one Cloud. For simplicity, the Fog (mini cloud) is built using one physical machine with one launched VM (server). The Cloud backbone is built using three physical machines: one Controller (to manage cloud services like networking), and two Compute machines to run VMs. As shown, the Cloud has launched four

VMs (servers). In our case, each server (VM) can be used to host the middleware that can execute the simulation.

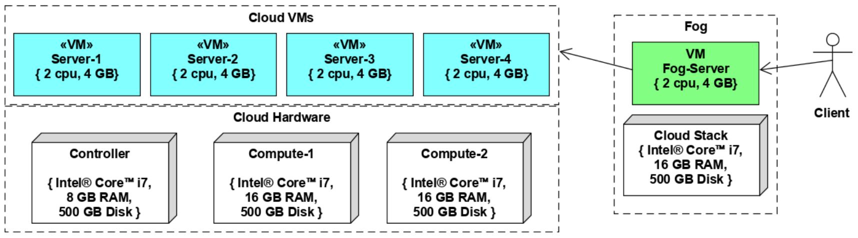
Cloud VMs
«VM» Server-1 { 2 cpu, 4 GB}
«VM» Server-2 { 2 cpu, 4 GB}
«VM» Server-3 { 2 cpu, 4 GB}
«VM» Server-4 { 2 cpu, 4 GB}
Cloud Hardware
Controller { Intel® Core™ i7, 8 GB RAM, 500 GB Disk }
Compute-1 { Intel® Core™ i7, 16 GB RAM, 500 GB Disk }
Compute-2 { Intel® Core™ i7, 16 GB RAM, 500 GB Disk }
Fog
VM Fog-Server { 2 cpu, 4 GB}
Cloud Stack { Intel® Core™ i7, 16 GB RAM, 500 GB Disk }
Client

Figure 1: Example of fog/cloud physical setup.

Those servers are then deployed in a parents-children hierarchical structure. For example, Figure 2 shows possible deployment configuration for the physical system shown in Figure 1. This achieved by sending either (1) one XML deployment configuration message to configure the whole hierarchy by propagating this message downward the hierarchy, or (2) to send this XML message to one or more servers to join/disjoin them to/from the overall hierarchical structure.

In our solution, as shown in Figure 2, Fog servers need to be on the top of the hierarchy (i.e. root nodes). This means that Fog servers have only access to the M&S resources within its subtree of the hierarchy. Accordingly, clients can only access the resources that are accessed by their Fog servers.

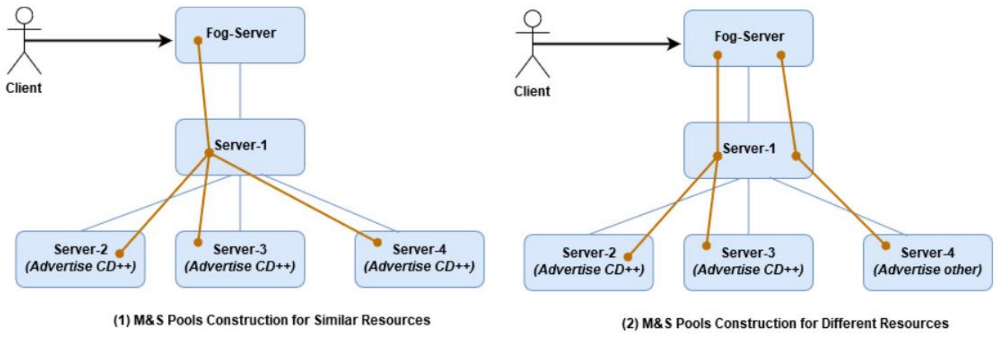
Client
Fog-Server
Server-1
Server-2 (Advertise CD++)
Server-3 (Advertise CD++)
Server-4 (Advertise CD++)
(1) M&S Pools Construction for Similar Resources
Client
Fog-Server
Server-1
Server-2 (Advertise CD++)
Server-3 (Advertise CD++)
Server-4 (Advertise other)
(2) M&S Pools Construction for Different Resources

Figure 2: Examples of deployment and M&S resources pools.

As previously stated, schedulers on Fog servers must have global knowledge to Discover & Select M&S resources within its subtree. To do so, servers with M&S capabilities sends their parents advertisements (i.e. as XML) describing their capabilities. This message is passed upward the hierarchy as necessary to construct the M&S Resources Pools. For example, suppose, in Figure 2-1, Server-2, Server-3, and Server-4 have the CD++ simulation capabilities. Now, when the first advertisement message reaches Server-1, the message is passed to the Fog-Server and a pool is constructed. However, when the other two advertisements messages reach Server-1, it just joins them to the existing resources pool, since they advertise the same resources. Now, assume Server-4 (in Figure 2-2) changed its capabilities to something other than CD++. As a result, Server-1 creates a second resources pool and passes the message up to the Fog-Server. Further, the load of those M&S resources pools is periodically computed by passing a special message called Compute-Load from parents to their children. In this way, these M&S resources pools provide Fog servers of global view of their subtree resources.

Now, when an experiment on a Fog server request server(s) to execute its simulation, the scheduler (on that Fog server) matches experiment requirements to those pools, hence finding the pools that can execute the simulation. It then selects the best pools according to their current loads. After that, it sends Get-Server message through those pools (i.e. to the children that connected to those pools). Now, when this message reaches the servers that originally advertised such resources, they can accept or reject this request. If accepted, the experiment on the Fog server will then have the URIs of those found servers. From now on, the experiment deals directly with those servers to the run the required experiment simulation. See (Al-Zoubi et al. 2020) for more in depth details about this Fog/Cloud platform.

4 SIMULATION MODEL: STRUCTURE AND VALIDATION

To validate the presented model, we compared the model's simulation results against real system setups using different physical and deployment configurations. In fact, the main reason behind developing this model is to study our real system on a very large scale. This is because, as in our case, building private Fogs and Clouds with numerous nodes can be very expensive. However, it is worth noting that we have built the real system in a range of tens of physical and VMs with different setups and configuration. We further conducted extensive analysis using the real system setups, as presented in (Al-Zoubi et al. 2020). Therefore, in this case, the model can study the system on a large scale (i.e. thousands of nodes) while the real system is used as a reference system to validate the simulation model.

The presented DEVS model main purpose is to mimic a real physical setup and launched VMs (e.g. Figure 1). Once this is done, there is no difference (i.e. uses same programming code) between the real system and the model with respect to the Fog/Cloud collaboration mechanisms particularly with the M&S resources deployment and management (see Figure 2 discussion).

To put this in an example, Figure 3 shows the DEVS model that resembles the real system shown in Figure 1. As can be seen, there are three types of coupled models: Top (to wrap the whole model), Cloud (to structure cloud elements), and Fog (to structure the Fog elements). Figure 3 shows one instance of each of those coupled models. In our case, these coupled models are written as textual script describing the shown diagram in Figure 3. As can be seen by this figure, these coupled models are made of four types of atomic models: Client, Router, Link, and VM. These atomic models are explained next.

The Client atomic model simulates client devices requests sending to the Fog servers. The example in Figure 3 has one instance of the Client atomic model. The Client atomic model is connected (via Link atomic model) to the highest Router instance, allowing clients to reach any other destination as explained next.

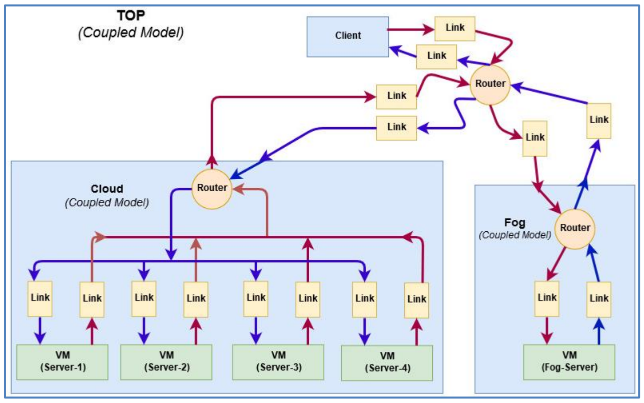

Figure 3: DEVS model for real system shown in Figure 1.

Router atomic model simulates Layer-3 network routing and Layer-2 switching. Each router is assigned a unique prefix IP address. The router matches this prefix address to incoming messages IP addresses to know how to route them. If the destination IP address matches the prefix address, the message is then forwarded to that router local network. Otherwise, it is passed to the parent default router. For example, if a router prefix is 132.179.0.0, a message with IP address 132.179.2.10 will be forwarded to this router local network. However, incoming message with IP address 132.180.2.10 will be sent to the parent (default) router. This keeps going until it reaches a router up in the hierarchy that can forward it within its local network. Of course, if the message reaches the highest router in the hierarchy and still cannot route it, the message is then returned to the sender with an error. In our example, Figure 3 has three router instances: One to route between Fog and Cloud, one to route within the Fog, and one to route within the Cloud. It is acceptable in this case since it is still with some respect a small setup, but we surely need to add more routers if we have thousands of VMs.

Link atomic model simulates networks latency. Network latency could be caused for many factors such as network congestion, distance, and network devices capabilities, etc. Link atomic model simplifies these factors by simply holding the transmitted message going through it for a certain delay. This is done by (1) stamping each received message being transmitted by the time it needs to be released, (2) and then placing it in an internal queue in a descending order. Accordingly, the simulation triggers internal event in the Link model when it is the time to release the messages in the front of the queue. This means one or more messages need to be released at the current simulation time. The issue now is how the Link model determines a message delay. Our first source was the results we collected from many different setups of the real system (e.g. Figure 1 shows an example of such setups). The collected results took into consideration if the message is being transmitted within the Cloud, Fog-to-Cloud, Client-to-Fog, etc. The Link model then used the occurrence probability of delay value with respect to the real system. For example, suppose from client device to Fog delay has measured 11 msec for 10% of all collected data in the real system. In this case, the Link atomic model between the client device and the Fog would delay a message for 11 msec a probability of 0.1. However, this was not enough since the transmission distance between senders and receivers also play a factor in the network latency as indicated by many studies like (Goonatilake et al. 2019). Thus, distance should be taken into consideration when communicating between Fogs and Clouds. However, in our real system setups, the distance between Clouds and Fogs is in the range of 30km to 60km, depending on the setup (i.e. were built within the same city). To overcome this issue, for long distances, we based network latency on the collected data in (Goonatilake et al. 2019), which gathered measurements between many different cities.

VM atomic model represents the server in the real system. As previously discussed, in the real system, a server is a RESTful-based middleware runs on a VM that can run different simulation environments. Further, each server contains the scheduler component that is in the core of the overall Fog/Cloud collaboration and resources management as discussed in Figure 2. Because our main purpose is to study those algorithms on large scale, the VM atomic model should then apply the same Fog/Cloud proposed algorithms as in the real system. In fact, this is what we did. The VM atomic model (i.e. C++) uses the actual code of the scheduler component (i.e. Java) that was used by the real servers. Thus, VM atomic only simulates the load introduced by executing simulation Jobs (i.e. experiments). Otherwise, it uses the same management code as in the real system.

In other words, our interest in the VM atomic model is to simulate the time it takes a simulation experiment to complete. From our intensive study to the real system, servers load was obviously related to three factors: (1) the background load (that is not related to simulation), (2) the model that is being simulated alongside the experiment settings, and (3) the number of concurrent simulation experiments being executed on the same server. Since the background load affects jobs completion time, we can safely assume it is implicitly calculated along with jobs completion time. However, in practice, it is hard to estimate the simulation jobs completion time or the load they add onto servers. This because simulation keeps going as soon as there are events to execute (which may further generate other events, and so on). Thus, this depends on the simulation model that is being executed and the simulation events are being generated. In fact, some simulation executions may go infinitely until it is literally stopped.

Therefore, to make the VM model as accurate as possible in predicting the length of a simulation job to complete, is by basing it on real servers in the real system. To do so, we collected results based on three effects: (1) the computing capability of a server (e.g. CPUs, RAM, etc.). (2) The simulation experiments being executed. For example, fire model (to study forest fire) executes differently from a ship model (to study ship evacuation), hence they have different completion times. (3) The number of concurrent simulation jobs being executed. Obviously, executing one ship model is different from executing 5 or 50 concurrent executions on the same server. The way we used those three factors to collect the results: We first defined a set of real servers with certain capabilities. We further, defined a set of real models to simulate over those real servers. After that, we measured the time to complete a simulation experiment for each model over each server in a stepwise fashion. This means for single experiment simulation, two concurrent simulations, and so on until sixty concurrent simulations. For each of those runs we averaged the value over 10 different repeated rounds. It is worth noting that results collection was performed using scripts that left running over the real servers for several days.

Based on above, the VM atomic model is initialized (i.e. via loading a file) the information related to its computing capability. Using this information, it can estimate the completion job of each running local job. Accordingly, every time a simulation job is added/removed to/from a VM atomic model server, the server's load and the expected completion time for all other jobs in progress are recalculated.

For further proof, we run additional tests to compare the VM atomic model to real servers with similar circumstances to ensure similar behaviors. Figure 4 shows one of those comparing examples. The figure compares the total execution time for simulating ship evacuation model between the VM atomic model and the real server. The slight differences can be explained by other external factors like the background processes running on the real server. The VM atomic model has also considered this background processing since it was based on real collected results from real servers. However, there will always be some differences when an experiment is repeated at different times even on the same real server.

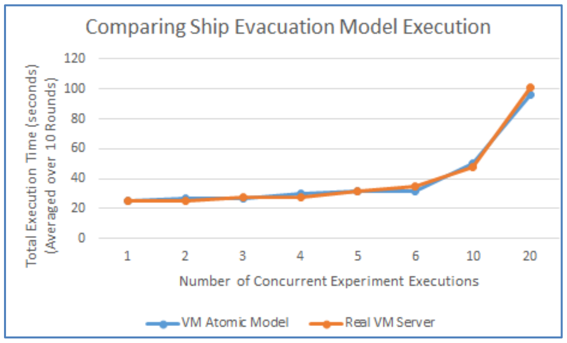

Figure 4: Example of comparing VM model to real server.

As discussed above, the first stage of validation was to compare internal atomic models to their equivalent real elements, as we did in the shown example in Figure 4. Now, the next level of validation is to compare entire real systems to their corresponding DEVS modelling with respect to the overall Fog/Cloud collaboration proposed mechanisms. As previously stated, the main purpose of the DEVS model is to mimic the physical structure. Beyond the physical structure, there is no difference between the DEVS model and the real system with respect to the deployment configuration and the overall Fog/Cloud collaboration algorithms. This necessary because the main purpose of this model is to study those algorithms on a large scale. However, before doing so, we must ensure the model behaves like the real system using various physical and configuration settings.

To put this in one example, the physical structure in Figure 1 is modeled in Figure 3. In this example, we have one Fog (i.e. with one server) and one Cloud (i.e. four servers). Both Figure 1 and Figure 3 were discussed earlier in detail. Figure 5 compares this real system (Figure 1) to the modeled system (Figure 3) using the deployment configuration in Figure 2-1 (see this figure earlier discussion). In this configuration, Server-2, Server-3, and Server-4 advertise CD++ simulation capabilities; hence, they can run CD++ simulation. On the other hand, Fog-server (i.e. used by clients to access services) and Server-1 (i.e. used for structural reasons) do not advertise any M&S capabilities. In this example, the client device created 60 experiments (to study spreading of fire in forests) on the Fog-server. After that, the client device started the simulation of those 60 experiments randomly over a period of ten seconds. This was repeated over 100 rounds while each run (in Figure 5) is an average of 10 actual rounds.

As expected, Figure 5 results show that the 60 experiments execution was reasonably spread over the three Cloud servers with CD++ capabilities. This means that the Fog-servers was able to discover the well-matched resources to execute the sixty jobs, and was further able to select the best resources to balance the load. As can be seen that the behavior of the real and modeled systems is very similar. Both executed the jobs on the same three servers (i.e. Server-2, Server-3, and Server4), and the servers got similar load (e.g. each executed on average 20 jobs). Of course, it is not expected to get the same numbers for both systems since many real-life factors can still play some role. This is even the case when comparing the same real system at two different runs, as shown in Figure 5-1. However, the overall behavior should be similar, which is met.

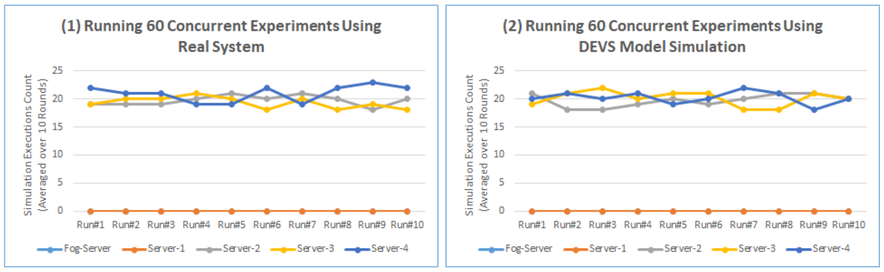

Figure 5: Comparing Real System to Model Behavior.

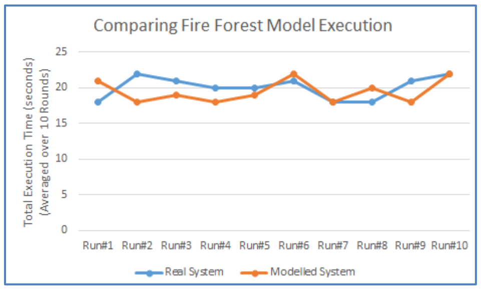

Figure 6: Comparing real to model job completion time.

Figure 6 shows the above results (in Figure 5) from the dimension of the total time it takes a job to complete execution. In Figure 6, each run is averaged over 10 different rounds, hence 100 rounds in total. On the real system, the fire model execution was 20.1 sec while it was 19.5 sec on the modeled system. The figure shows that both real and modeled behave in a very similar way. The difference is like in the case of a real system with same physical setups and configuration being run and compared at two different times. This fact can be observed when comparing the real system different runs on Figure 6. This makes sense since the modeled system uses real collected data from real system previous runs, as previously discussed.

5 STUDY ON LARGE SCALE: MODEL & RESULTS

As previously discussed, in our platform, clients create and setup their experiments on Fog servers. Once this done, a client then starts a simulation on an experiment via a Fog server. The experiment (on a Fog server) then discovers and selects the proper M&S resources to execute the experiment simulation. After the selection step, experiment (on the fog server) runs simulation directly on the selected servers, and becomes an interplay unit between client(s) and selected servers.

As can be seen, our main purpose of modelling our existing Fog/Cloud real system is to study the collaboration algorithms scalability on large scale. In the other words, our concern here is to study the issues that are hard to experience with the real system setups.

Thus, we more concerned in studying scalability by studying the effect of enlarging the system on two things:

(1) On the overall deployment structure (hence, we should always be able to keep adding/removing servers without affecting the overall structure), and

(2) The time it takes an experiment to discover and select resources. This *discovery & selection* time is measured from the time an experiment asks its local fog server host (i.e. scheduler) to find the compatible servers to execute the simulation until the experiment gets the URIs for those servers.

5.1 Modelling the Fog & Cloud Infrastructure

Figure 7 shows modelling the physical structure on large scale. The elements that make up this model have already been described in Figure 3. The highest level in the model (Figure 7) consists of one atomic model router, called Router-top, and six instances of coupled models called Clients, Cloud, Fog1, Fog2, Fog3, and Fog4. Router-top is responsible for routing all traffic between all Clients, Cloud, and Fogs. This is done by routing the traffic to/from other Fog/Cloud routers. Note that connections between elements is made via Links atomic models. Links have already been fully discussed in Figure 3. Clients coupled model contains several Client atomic models (to mimic clients' requests) that connected to router Router-clients. This Router is also connected to the highest router (Router-top) to reach other networks. Similarly, each of the four Fogs contains a server and a router to route traffic inside the Fog, and at the same time connected to the highest router (Router-top) to reach other networks.

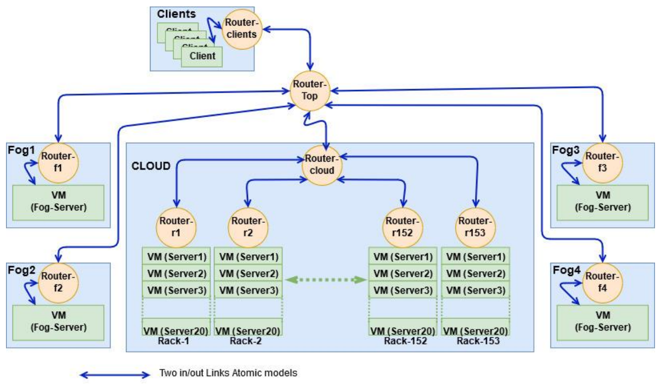

Figure 7: Modeling physical infrastructure on large scale.

The Cloud coupled model consists of 5 Routers and 153 internal coupled models called Racks. Each Rack contains 20 VMs (servers) atomic models and 1 Router. This makes the cloud contains 3,060 servers in total. Our approach treats servers as pool of resources that can be deployed dynamically in many ways. Deployment configuration for these servers is described soon in Figure 8. The Rack routers (i.e. Router-r1…Router-r153) route traffic between their internal rack servers and at the same time connected to the highest router (Router-cloud) in the cloud to reach other racks and other external networks. Router-cloud routes traffic between cloud racks and at the same time connected to the highest router (Router-top) to reach other external networks.

This model was simulated using CD++ distributed simulation (Al-Zoubi et al. 2015), hence was partitioned over three physical machines: The first partition is the Cloud coupling model, the second partition is the four Fogs coupled models, and the third partition is the Clients coupled model.

5.2 Deployment Configuration

As explained earlier, deployment configuration XML messages (to build the hierarchy structure) are sent to each server individually or to all servers altogether starting from top of the hierarchy. This step is the same on both real and modeled servers (i.e. same code implementation).

Figure 8 shows the deployment configuration based on the shown infrastructure in Figure 7. In this deployment, we created 50 groups of servers where each group contains 60 servers. Those groups of servers are the ones that advertise M&S resources and mimics simulation execution. Each group of servers are made children to one intermediate server (e.g., Server-1 is the parent of all servers in Group-1 and so on). For instance, to make servers in Group-1 accessed by the Fog servers, Server-1 needs to be made a child to Server-Root (i.e. the highest server on the cloud hierarchy). Likewise, we can remove a group of servers from being accessed by Fog servers. In this way, we can study the effect of adding more servers on the overall system, as presented later.

Further, to give access to a Fog server, it just needs to be made a parent to the Server-Root. Note that this hierarchy is only needed to manage and discover resources. However, once servers are discovered and selected by a Fog server, the Fog server then execute jobs directly on those selected servers

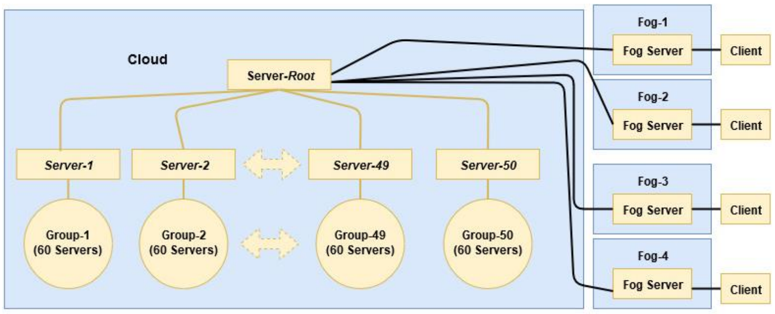

Figure 8: Large scale deployment configuration.

5.3 Case Study Results

The overall structure scalability and the discovery & selection time is studied from two dimensions: The first dimension is when the number of servers keep increasing. The second dimension is when the M&S resources (advertised by those servers) heterogeneity keep increasing. Note that the real processing time on the Fog servers to access its local data structure is considered in addition to the cloud and network simulation time. This is because modeled Fog servers build similar local data structures to the real Fog servers.

Figure 9 shows the discovery & selection results using those two dimensions (i.e. servers count and advertised M&S resources heterogeneity). All cases in Figure 9 starts with one group (i.e. 60 servers), 5 groups (i.e. 300 servers), and then keeps adding 5 more groups of servers with each step until it reaches the whole 50 groups (i.e. 3000 servers).

Figure 9-1 shows the case when all servers advertise the same M&S resources (e.g. CD++). This means that Fog servers on top of the hierarchy will always see one pool of resources regardless of the number of servers on the cloud. This also applies to the other intermediate schedulers (i.e. Server-Root, Server-1…Server-50). Therefore, local processing is trivial in this case comparing to network latency, which measured in a range of 5-7 msec. As a reminder, a network message needs to be passed downward the pools hierarchy to retrieve the servers URIs. As expected, Figure 9-1 shows that adding more servers did not affect the direction of the line in the figure. In fact, the variation of the line is more affected by the network latency rather than of the number of servers.

Figure 9-2 shows the case when each group of servers advertise different M&S resources. This means each of the group schedulers (i.e. Server-1…Server-50) sees one resources pool for all its 60 children servers. The Server-Root sees 50 resources pools since each of its children resources are different from each other. This also applies to the Fog servers.

The argument of Figure 9-1 also applies to the results of Figure 9-2. The local processing is trivial (i.e. data structures of at most 50 entries) with respect to network latency. Accordingly, the direction of the line is more affected by network latency rather than of the number of servers or the M&S resources heterogeneity.

Figure 9-3 shows the case when each server within each group advertises an M&S resources at random. This done by making each server selects at random an M&S resource from a list of 100 resources. This means that every time we add more servers with every run, the whole resources pools change as well since servers keep changing their advertised resources. This case has showed that the overall behavior remained stable as the system grows with random heterogeneity.

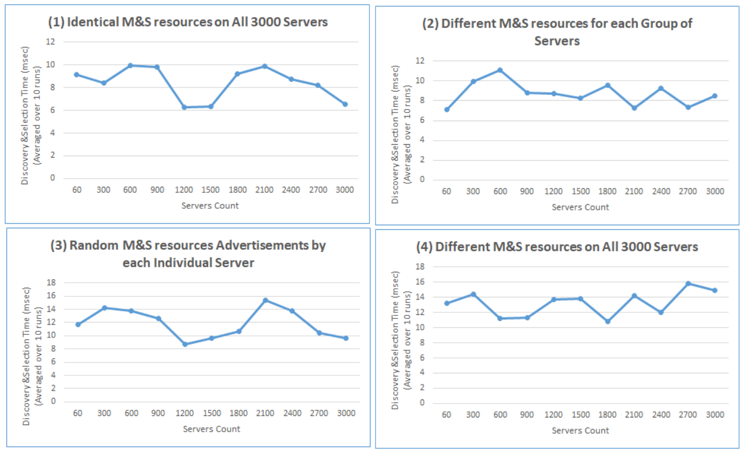

Figure 9: Results based on different resources and servers.

Figure 9-4 shows the worst-case scenario. In this case, we made every server advertises a different M&S resources from all other servers. This means the cloud support 3000 different M&S resource where each is installed on a different server. This means each of the group schedulers (i.e. Server-1…Server-50) sees 60 resources pool since it has 60 children servers. The Server-Root sees 3000 resources pools (i.e. 50 groups × 60 servers). This also applies to the Fog servers. Thus, local processing is expected to increase. However, when compared to the best-case scenario (in Figure 9-1), this increase is on average about five msec. Further, the overall behavior showed stable system despite enlarging the system with the worst-case scenario of heterogeneity.

6 CONCLUSIONS

We have built and developed full Fog/Cloud collaboration mechanisms to be used by modelers to run their simulation experiments via nearby Fog servers without being concern on how and where the required M&S resources can be found to execute the simulation. However, since these real systems are usually setup of tens of servers, it was then necessary to study this system on large scale using modelling and simulation.

We showed how this model was validated with respect to the real system. This was needed to have the required confidence before modelling the system on large scale. The presented results showed that the system overall structure and the Fog/Cloud collaboration met the scalability requirements on large scale.

REFERENCES

Al-Zoubi K. and Wainer G. 2015. "Distributed Simulation of DEVS and Cell-DEVS Models Using the RISE Middleware". *Simulation Modelling Practice and Theory*. Elsevier. Vol. 55:27-45

Al-Zoubi K., Wainer, G. 2020. "Fog and Cloud Collaboration to Perform Virtual Simulation Experiments". *Simulation Modelling Practice and Theory*. Elsevier. Vol. 101. DOI. https://doi.org/10.1016/j.simpat.2019.102032

Bonomi F., R. Milito, J. Zhu, S. Addepalli, "Fog Computing and Its Role in the Internet of Things", In the Proceedings of the first MCC Workshop Mobile Cloud Comput., pp. 13-16, 2012.

CloudSim. http://www.cloudbus.org/cloudsim/ .

Etemad M.; Aazam M.; St-Hilaire M. "Using DEVS for Modeling and Simulating a Fog Computing Environment", Proceedings of the 2017 International Conference on Computing, Networking and Communications (ICNC). 2017. Pages:849 - 854.

Goonatilake R., and Bachnak R. "Modeling Latency in a Network Distribution". Network and Communication Technologies; Vol. 1, No. 2; 2012. Also accessed via https://www.semanticscholar.org/paper/Modeling-Latency-in-a-Network-Distribution-Goonatilake-Bachnak/ff135d221678e6b542391c831e87fca56e830a73 . Accessed November 2019.

Hu, Dhelim S., Ning H., Qiu T., "Survey on Fog Computing: Architecture Key Technologies Applications and Open Issues", J. Netw. Comput. Appl., vol. 98, pp. 27-42, Nov. 2017.

Malik A., Kashif B., Malik S., Zahid Z., Aziz K., Kliazovich D.,Ghani N., Khan S., Buyya R., "CloudNetSim++: A GUI Based Framework for Modeling and Simulation of Data Centers in OMNeT++". Services Computing IEEE Transactions. Vol. 10, pp. 506-519, 2017, ISSN 1939-1374.

Naha R.; Garg S.; Georgakopoulos D.; Jayaraman P.; Gao L.; Xiang Y.; Ranjan R. "Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions". Institute of Electrical and Electronics Engineers, Inc.(IEEE) Access. Vol. 6, pp. 47980 – 48009. 2018.

Ostermann S., Plankensteiner K. Prodan R., and Fahringer T. "GroudSim: An Event-based Simulation Framework for Computational Grids and Clouds". https://pdfs.semanticscholar.org/4097/9f56038d443ad75b519ef8d94c11348a2b39.pdf . Accessed on January 11, 2019.

Shrivastwa A., Sarat S., Jackson K., Bunch C., Sigler E., Campbell T. "OpenStack: Building a Cloud Environment". Packt Publishing (September 19, 2016). Birmingham, United Kingdom.

Yi S., Li C., Li Q., "A Survey of Fog Computing: Concepts Applications and Issues", Proc. Workshop Mobile Big Data, pp. 37-42, 2015

Wainer G., Discrete-event Modeling and Simulation: A Practitioner's Approach, CRC/Taylor & Francis, UK, 2009

AUTHOR BIOGRAPHIES

KHALDOON AL-ZOUBI received both Ph.D. (2011) in Electrical and Computer Engineering and M.C.S (2006) from Carleton University (Ottawa, Ontario, Canada). He received a BSc in Electrical Engineering (1995) from the University of Louisiana at Lafayette (Lafayette, Louisiana, USA). Before joining the Faculty of Computer & Information Technology at the Jordan University of Science & Technology (JUST) in 2016, he worked for more than 20 years as a Senior Software Engineer, Researcher and inventor in leading hi-tech companies in Canada and USA. His current research interests are related to Parallel and Distributed computing, Modeling and Simulation, Fog/Cloud Computing, and Software Defined Networks (SDN). His patents list can be found at https://patents.justia.com/inventor/khaldoon-al-zoubi. His email is ktalzoubi@just.edu.jo .

GABRIEL A. WAINER , SMSCS, SMIEEE, received the M.Sc. (1993) at the University of Buenos Aires, Argentina, and the Ph.D. (1998, with highest honors) at the University of Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. After being Assistant Professor at the Computer Science Department of UBA, in July 2000 he joined the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada), where he is now Full Professor. He has held visiting positions at the University of Arizona, LSIS (CNRS), University Paul Cezanne, University of Nice, INRIA Sophia-Antipolis (France), UCM (Spain) and others. He is the author of three books and over 260 research articles; he edited four other books, and helped organizing over 120 conferences, including being one of the founders of SIMUTools and SimAUD. He was PI of different research projects (funded by NSERC, CFI, GRAND, MITACS, Autodesk Research, IBM, Intel, INRIA, CANARIE, Precarn, Usenix, CONICET, ANPCYT). Prof. Wainer is the Vice-President Conferences, and was a Vice-President Publications and a member of the Board of Directors of SCS. He is Special Issues Editor of SIMULATION, member of the Editorial Board of IEEE Computing in Science and Engineering, Wireless Networks (Elsevier), Journal of Defense Modeling and Simulation (SCS), and International Journal of Simulation and Process Modelling (Inderscience). He is the head of the Advanced Real-Time Simulation lab, located at Carleton University's Centre for advanced Simulation and Visualization (V-Sim). He is also the Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet. He has been the recipient of various awards, including the IBM Eclipse Innovation Award, SCS Leadership Award, and various Best Paper awards. He has been awarded Carleton University's Research Achievement Award (2005-2006), the First Bernard P. Zeigler DEVS Modeling and Simulation Award, and the SCS Outstanding Professional Award (2011). His current research interests are related with modelling methodologies and tools, parallel/distributed simulation and real-time systems. His e-mail and web addresses are gwainer@sce.carleton.ca and www.sce.carleton.ca/faculty/wainer .