Modelling VM Migration in a Fog Computing Environment

Pedro Juan Roig[1,2,*], Salvador Alcaraz[1], Katja Gilly[1], Carlos Juiz[2]
[1]*Department of Physics and Computer Architecture, Miguel Hernandez University, Avda. Universidad, s/n - 03202 Elche (Alicante), Spain*
[2]*Department of Computer Science, University of the Balearic Islands, Ctra. Valldemossa, km 7.5 - 07122 Palma de Mallorca, Spain*
pedro.roig@goumh.umh.es

***Abstract*—Fog Computing is created to efficiently store and access data without the limitations challenging Cloud Computing deployments, such as network latency or bandwidth constraints. This is achieved by performing most of the processing on servers located as close as possible to where data is being collected. When mobile devices are equipped with limited resources and small capabilities, it would be convenient to make their associated computing and network resources follow them as much as possible. In this paper, migration process is studied and an algorithmic model is designed, selecting a generic Fat Tree architecture as the underlying topology, which may be useful to get a list of all devices being traversed through each of the redundant paths available.**

***Index Terms*—Fog computing; Migration; Modelling; Networking.**

I. INTRODUCTION

Fog Computing paradigm is characterised by the allocation of computing resources at the edge of the network, thus bringing the cloud computing assets closer to the end user [1]. In this context, special attention may be set on its use in Internet of Things (IoT) deployments and, particularly, in IoT moving environments [2].

Such moving IoT devices have special characteristics according to its limited computing capacity, limited battery resources, and limited bandwidth [3]. So, a good solution for their implementation is to decouple their computing assets and move them to more resourceful facilities, powerful enough to take responsibility for a lot of IoT devices, but close enough to reduce latency and bandwidth usage [4]. Therefore, those facilities are composed of a bunch of capable servers being able to assign a Virtual Machine (VM) to each user to cover the computing needs of each IoT device.

When talking about moving IoT devices, this outlook is crucial as those devices lack resources of all kinds [5], such as those related before, and the use of a VM to carry the computing assets of each device may help to cope with the issues regarding the resources [6].

However, a new problem arises with the moving IoT devices because, as they are moving around, those VMs might end up being too far away in the complex network architecture from their associated devices. Hence, VMs should be moved as close as possible. This mechanism is called VM migration and must be taken into account in those environments [7].

Therefore, two types of movements can be distinguished: the movement of the device throughout the coverage area and the movement of the virtual machine associated to that device trying to get as close as possible to its owner.

The first sort of movement, the one regarding just the moving IoT device, is called mobility. Its study is all about trying to model the most usual movements as well as the not so usual ones. Regarding literature, there are some attempts to model general human movements in wireless environments, such as [8] and [9]. Also, there are some simple mathematical models, such as the ones proposed in [10], [11], and other more complex models related to crowd interaction, such as [12], [13], and [14].

The second kind of movement, the one related to VMs associated to moving IoT devices trying to follow them around, brings about the issue of trying to migrate a VM from the server hosting it to another one being located nearer to the actual position of the moving IoT device in order to facilitate the interaction between the device and its computing power, as a consequence of reducing the latency and bandwidth of such communications.

Regarding literature, a conceptual live VM migration framework is proposed in [15], also for Cloud Computing in [16] and for Fog Computing in [17], whereas a comparison between live VM migration in both environments for multimedia services is presented in [18].

In this paper, we are going to concentrate on studying the VM migration happening in the situations mentioned above. Furthermore, we will focus on getting a general algorithm for modelling it in a generic Fat Tree architecture for to make an interesting framework being able to support the necessary infrastructure for allocating VMs within physical servers and facilitating VM migration throughout any pair of available servers.

The paper is organized as follows. In Section II, a general procedure for live VM migration is introduced. In Section III, a Clos network overview is shown. The description of the behaviour of a Fat Tree architecture from the modelling point of view is given in Section IV. Later, in Section V, a

Manuscript received 18 December, 2018; accepted 1 July, 2019.

general algorithm aimed at modelling VM migration in a Fat Tree topology is proposed. A method for getting all devices on redundant paths is presented in Section VI, whereas final conclusions are drawn in Section VII.

II. LIVE VM MIGRATION PROCESS

Regarding VM migration, there are three main approaches to be taken into account [19], such as cold migration, where the VM is shut down before moving it, hot migration, where just its OS is suspended before the movement, and, finally live migration, thus allowing the services running on it to keep going in a seamless manner whilst the movement is performed. The latter one is the most interesting situation.

There are three key parameters to measure the performance of live migration [20], such as downtime, which represents the amount of time the VM is halted during the migration, total migration time carrying the amount of time elapsed for the whole process and the amount of dirty pages migrated referring to the data being changed during the process, and, therefore, having to be further sent over again.

Regarding the live VM migration process, a trade-off must be considered between the downtime and the total migration time. In order to achieve this, the memory transfer is the key player. However, connections to the local devices and network interfaces may also be taken into account.

Generally speaking, the memory transfers can be broken up into three stages [21]:

1. Push phase, where the source VM keeps running whilst the transfer process starts taking place;
2. Stop-and-copy phase, where the source VM is halted, pages are copied through, and destination VM is started;
3. Pull phase, where the new VM runs and, if a requested page has not yet been copied, it is retrieved from the source VM.

Some techniques are proposed to undertake the live VM migration process in an efficient manner by just focusing on one or two of the stages described above, such as pure stop-and-copy, pure demand-migration or post-copy live migration. However, it seems that the most efficient approach is the pre-copy migration composed by a combination of a bounded iterative push stage with a very short stop-and-copy stage, where a number of iterations take place until all dirty pages are transferred.

The pre-copy migration process between two hosts may be divided into six stages, where a VM transaction between the two hosts takes place according to a pre-established migration timeline:

1. Pre-migration, where a destination host with enough resources is preselected;
2. Reservation, where resources are allocated beforehand at that destination host;
3. Iterative pre-copy, where the whole RAM is sent in the first iteration and dirty pages are sent in the following iterations;
4. Stop-and-copy, where the source VM is halted, so as to copy its CPU state and remaining inconsistent pages to the destination VM;
5. Commitment, where the destination host acknowledges it has received a consistent VM copy and the source host acknowledges it back prior to discarding the original VM;
6. Activation, where the migrated VM gets activated and the device drivers are attached to the new VM.

To round it all up, Fig. 1 exhibits the timeline for the live VM iterative pre-copy migration process.

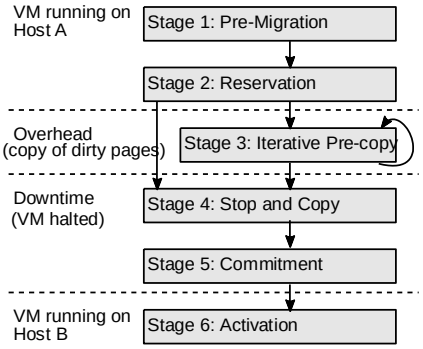

Fig. 1. VM Migration Timeline.

III. CLOS NETWORKS

Back in the fifties, Clos networks were designed in order to switch telephone calls in an efficient manner [22] by virtue of using crossbar switches. Basically, the point was the use of equipment with multiple stages of interconnection in order for the calls to be completed, hence providing alternative paths between sources and destinations allowing the phone call to be always connected and not blocked by any other call.

Later in the nineties, Ethernet switches came along and the concept of Clos networks was expanded for to achieve cost-effective, reduced operational complexity, and limited scalability [23]. The point there was to create multistage topologies built with commodity switches, so cost-effective deployments might be attained.

Afterwards, with the arrival of the 21st century, Data Centers and Cloud Computing facilities are still using those topologies with different proposals, such as two-stage designs [24], three-stage ones [25] or even alternative ones [26] each one having its own benefits and drawbacks, hence providing a full range of solutions in order to deal with different situations.

Those topologies may be well used regarding the underlying structure of the Fog Computing environments in order to host VMs and support the necessary live VM migrations, where two of the main proposals in literature are Leaf and Spine [27] and Fat Tree architectures [28].

Leaf and Spine is a 2-tier topology, where the lower one is composed of switches directly connecting with servers and the upper one is made of switches interconnecting the lower ones in a full mesh fashion. This design provides full redundancy, as there are always a number of redundant paths among any two given switches, which are equal to the number of switches being part of the upper layer. The main drawback of the design, as it is prone to scalability issues as the number of switches increases, is the number of redundant connections to be provided. Figure 2 depicts an example of such a topology with four switches in the Spine layer.

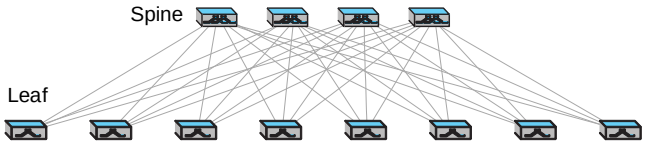

Fig. 2. Leaf and Spine topology.

On the other hand, the Fat Tree is an alternative to the above mentioned topology because it allows better scalability. It is a 3-tier topology subdivided in Pods, where the full mesh interconnection among the switches located in the lower layer and the upper layer of each Pod is. It is quite similar to the first topology introduced, except having the extra top layer, which is in charge of interconnecting the different Pods taking part in the topology.

This way, there are less redundant paths among any two given switches, but there are no scalability issues any more. Figure 3 exhibits an example of the Fat Tree topology with K = 4 and 1:1 oversubscription meaning that all theoretical links in the topology have been used.

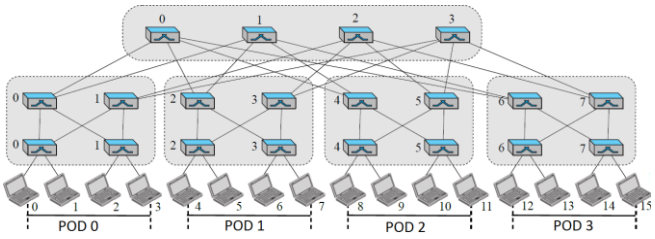

Fig. 3. Fat Tree topology.

IV. FAT TREE BEHAVIOUR FOR MODELLING

In order to obtain a modelling for the Fat Tree architecture, it is necessary to describe how each layer of the topology behaves. To start with taking the Fat Tree architecture depicted above as a ground for a formal model, devices are identified from left to right for each different layer, so as to design a model showing how this architecture allows the communication flow among servers.

Fat Tree architecture is composed of three layers of switches, where Edge layer is the lower one, Aggregation layer is the middle one, and Core layer is the upper one. Additionally, a bottom layer holding hosts or servers is also considered.

Regarding the name of the Fat Tree, the word "Tree" comes from the inverted tree-like structure of this architecture, where Core layer might be seen as the root layer and Hosts layer as the leaves layer.

Furthermore, following that analogy, the word "Fat" comes from the existence of more links on the root layer than in the leaves layer, such that there are $(K/2)^2$ links between upper and middle layers, $(K/2)^1$ links between middle and lower layers, and $(K/2)^0$, i.e., 1 link between lower and hosts layers.

The Fat Tree structure may be considered as a K-ary tree, being K the main parameter of this structure, as there are K Pods each of those containing K switches divided into lower and middle layers and each switch also has K ports.

In Figure 3, K = 4 is used, although it may be extended to any natural even number. Therefore, it will be represented by the algorithmic model to be shown.

The model might be built up by using Fig. 3 as a source of information. As it may be seen over there, switches at the all layers are numbered from 0 onwards considering left to right direction.

As a matter of fact, there is a total of (K/2) hosts hanging out of each lower switch, which means that there are $(K^2/2)$ hosts hanging out of each Pod, which also means that there are $(K^3/4)$ hosts in the whole topology. In addition to that, there is a total of $(K^2/2)$ switches in both the lower layer and the middle layer, whereas there are $(K^2/4)$ in the upper one.

As per nomenclature of the ports of each item, it is said that the servers have only one port, which is labeled as 0, whereas on the switches the ports are labeled from 0 to (K-1) from left to right starting from the bottom and finishing on the top (Fig. 4).

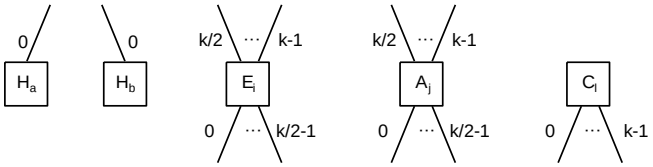

Fig. 4. Links on each layer of the Fat Tree architecture.

With all this in mind, the model is going to state the atomic actions for communicating messages involved in the live VM migration process, such as send or receive in a way that both will bear the item and the corresponding port taking part in such communication. Furthermore, the model is going to show the decision-making processes in order to guide a VM from a given source to the proper destination following the optimum paths that are one, two or three hops away.

The arguments for the send and receive actions are the source host (a), the destination host (b), and the proper VM to be migrated owned by a user identifier (u), i.e., VM(u).

All items within each layer are modelled in a generic manner, such as HOST H_h, EDGE E_i, AGGREGATION A_j, and CORE C_l, where each variable is bounded by the number of items for each layer exposed above. The model is expressed with some snippets coded in C-style for clarity purposes. It takes into account all considerations being made also.

V. VM MIGRATION MODEL

– *HOST H_h*:

The server layer, also known as the Host layer, is the easiest to model as each server (h) may perform just two actions, such as receiving or sending a VM. Those actions are the key actions chosen herein for modelling the VM migration process. Further considerations, such as VM creation or VM termination are not borne in mind for the simplification purposes. Let us suppose there is enough room to allocate the VMs assigned to all the users within the system.

All the servers in the topology are awaiting to undertake any of those two actions of receiving and sending at any given time as it is not usually known beforehand when a VM migration is going to take place. As per the receiving part, it is performed when the VM associated to a particular user is not located into a given host, hence the host is ready to

receive that VM in anytime. Otherwise, the sending part is carried out when the aforesaid VM is indeed located in that host and it is time to leave (see Algorithm 1).

Additionally, each server has only one port, namely eth0, so messages to and from the Edge layer move through that single port, whilst messages have the structure (a, b, and VM(u)).

Algorithm 1. HOST( ).

```
for (h = 0; h < (K^3/4); h++)
{
 while (1)
 {
  for (u = 0; u < TOTAL_USERS; u++)
  {
   if NOT (VM(u) in h)
    receive HOST {h}, PORT {eth0} (a,h,VM(u));
   else if LEAVE_NOW(VM(u))
    send HOST {h}, PORT {eth0} (h,b,VM(u));
   else
    STAY_IN(VM(u));
   }
  }
 }
}
```

– *EDGE E_i*:

The lower layer, also known as the Edge layer, is the one directly connected to the servers. Each switch located therein is continuously monitoring all its ports in order to receive a VM to be translated from a source host to a destination host.

It is to be remarked that any switch located in this layer has the lower half of its ports looking downwards, whereas the upper half are looking upwards.

On the one hand, in case a VM is received from a given host on any switch, it is going to come from any of its lower ports, and then two cases may be distinguished. If the destination is another host hanging on the same switch, it is going to be forwarded downwards straight to that other host. In this case, both hosts may be considered as being part of the same IntraNet, as they may be just one-hop away to each other. Otherwise, if that is not the case, it is going to be forwarded upwards.

On the other hand, in case a VM is received from any switch standing on the middle layer, it is going to come from any of its upper ports, and then it will be guided through the port directly connected to the destination host (see Algorithm 2).

Algorithm 2. EDGE( ).

```
for (i = 0; i < (K^2/2); i++)
{
 while (1)
 {
  for (m = 0; m < (K/2); m++) {
   if(receive EDGE {i}, PORT {m} (a,b,VM(u))){
    if (int[a/(K/2)] == int[b/(K/2)])
     send EDGE {i}, PORT {b mod (k/2)} (a,b,VM(u));
    else
     for (m' =(K/2); m' < K; m'++)
      send EDGE {i}, PORT {m'} (a,b,VM(u));
   }
  }
  for (m' = (K/2); m' < K; m'++) {
   if(receive EDGE {i}, PORT {m'} (a,b,VM(u))){
    send EDGE {i}, PORT {b mod (k/2)} (a,b,VM(u));
  }
 }
}
```

– *AGGREGATION A_j*:

The middle layer, also known as the Aggregation layer, has the same port configuration as the lower layer. It has to be noted that there is a full mesh topology among switches staying on both layers within a single Pod.

As in the previous layer, all switches therein are monitoring their ports and all the time waiting for incoming VMs. Therefore, if a VM is received from any of the lower ports of any of these switches, two case scenarios may be distinguished.

If the destination is another host situated on the same Pod, that VM will be forwarded downwards through the lower switch, where the destination is hanging on as both source and destination hosts may be considered as being a part of the same Pod, also known as IntraPod, being two-hops away from each other. Otherwise, if this is not the case, it is going to be forwarded upwards.

Alternatively, if a VM is received from any upper ports of any of those switches, it is going to be headed for the lower layer switch, where the destination host is hanging on (see Algorithm 3).

Algorithm 3. AGGREGATION ( ).

```
for (j = 0; j < (K^2/2); j++)
{
 while (1)
 {
  for (n = 0; n < (K/2); n++) {
   if (receive AGGR {j}, PORT {n} (a,b,VM(u))){
    if (int[a/(K/2)^2] == int[b/(K/2)^2])
     send AGGR {j}, PORT {int[b/(k/2)]} (a,b,VM(u));
    else
     for (n' = (K/2); n' < K; n'++)
      send AGGR {j}, PORT {n'} (a,b,VM(u));
   }
  }
  for (n' = (K/2); n' < K; n'++) {
   if(receive AGGR {j}, PORT {n'} (a,b,VM(u))){
    send AGGR {j}, PORT {int[b/(k/2)]} (a,b,VM(u));
  }
 }
}
```

– *CORE C_l*:

The upper layer, also known as the Core layer, is the one interconnecting different Pods, so the ports of all its switches are looking downwards providing a full mesh topology among all the existing Pods.

Therefore, all of the switches are waiting to receive a VM through any of its corresponding ports and, when that happens, it is redirected to its directly connected middle layer switch on the Pod holding the destination host.

In this case, the source and destination hosts do not share the same Pod, also known as InterPod, meaning both are three-hops away to each other (see Algorithm 4).

Algorithm 4. CORE( ).

```
for (l = 0; l < (K^2/4); l++)
{
 while (1)
 {
  for (p = 0; p < K; p++) {
   if(receive CORE {l}, PORT {p} (a,b,VM(u))){
    send CORE {l}, PORT {int[b/(k/2)^2]} (a,b,VM(u));
  }
 }
}
```

VI. GETTING ALL DEVICES ON REDUNDANT PATHS

All the above may be used in order to get a list of devices for each of the redundant paths taken from a given source host to a given destination host as it may be seen in Algorithm 5. The nomenclature for the devices is the same one used so far.

– *LIST OF DEVICES*:

Algorithm 5. DeviceList (a, b).

```
DeviceList(a,b){
 items = [];
 items += [ Ha ];
 if (int[a/(K/2)] == int[b/(K/2)])
 {
  items += [ Eint[a/(K/2)] Hb ];
  t = 1;
  topology = "INTRANET";
 }
 else if (int[a/(K/2)^2] == int[b/(K/2)^2])
 {
  items += [ Eint[a/(K/2)] "(" ];
  for (q = (K/2) · int[a/(K/2)^2];
       q < K/2) · int[a/(K/2)^2] + (K/2); q++)
   items += [ Aq ];
  items += [ ")" Eint[b/(K/2)] Hb ];
  t = 2;
  topology = "INTRAPOD";
 }
 else
 {
  items += [ Eint[a/(K/2)] "(" ];
  for (q = (K/2) · int[a/(K/2)^2];
       q < K/2) · int[a/(K/2)^2] + (K/2); q++)
  {
   items += [ Aq "{" ];
   for (r = q · (K/2); r < q · k; r++)
    items += [ Cr ];
   s = (K/2) · int[b/(K/2)^2]  + r mod (K/2) ;
   items += [ "}" As ];
  }
  items += [ ")"Eint[b/(K/2)] Hb ];
  t = 3;
  topology = "INTERPOD";
 }
 print("Topology: %s\n",topology);
 print("Number of Hops: %d\n",t);
 print("Redundant Paths: %d\n",(K/2)^(t-1));
 print("Items: %s\n", items);
}
```

The list presented above may be extended with the corresponding ports used in each link for all redundant paths as it may be seen in Algorithm 6. The ports are going to be expressed within parenthesis attached to its corresponding device with the sign "*" and a link between two ports is going to be expressed by the signs "---" appearing in between both ends of such a link.

– *LIST OF DEVICES AND PORTS*:

Algorithm 6. DeviceAndPorts List (a, b).

```
DeviceAndPortsList(a,b){
 if (int[a/(K/2)] == int[b/(K/2)])
 {
  t = 1;
  topology = "INTRANET";
 }
 else if (int[a/(K/2)^2] == int[b/(K/2)^2])
 {
  t = 2;
  topology = "INTRAPOD";
 }
 else
 {
  t = 3;
  topology = "INTERPOD";
 }
 print("Topology: %s\n",topology);
 print("Number of Hops: %d\n",t);
 print("Redundant Paths: %d\n",(K/2)^(t-1));
 // REDUNDANT PATHS
 if (topology == "INTRANET")
 {
  Path(0) = [ Ha *
            * (0) --- (a mod (K/2))*
            * Eint[a/(K/2)] *
            * (b mod (K/2)) --- (0) *
            * Hb ];
 }
 else if (topology = "INTRAPOD")
 {
  x = 0;
  for (q = (K/2) · int[a/(K/2)^2];
       q < K/2) · int[a/(K/2)^2] + (K/2); q++)
  {
   Path(x) = [ Ha *
             * (0) --- (a mod (K/2))*
             * Eint[a/(K/2)] *
             * ((K/2) + x) --- (int[a/(K/2)])*
             * Aq *
             * (int[b/(K/2)]) --- ((K/2) + x) *
             * Eint[b/(K/2)] *
             * (b mod (K/2)) --- (0) *
             * Hb];
   x++;
 }
 }
 else   // if(topology == "INTERPOD")
 {
  y = 0;
  for (q = (K/2) · int[a/(K/2)^2];
       q < K/2) · int[a/(K/2)^2] + (K/2); q++)
  {
   z = 0;
   for (r = q · (K/2); r < q · K; r++)
   {
    s = (K/2) · int[b/(K/2)^2] + r ·mod (K/2);
    Path(y · (K/2) + z) =
          = [ Ha *
            * (0) --- (a mod (K/2))*
            * Eint[a/(K/2)] *
            * ((K/2) + y) --- (int[a/(K/2)])*
            * Aq *
            * ((K/2) + z mod (K/2)) ---
              --- (int[a/(K/2)^2]) *
            * Cr
            *  (int[b/(K/2)^2]) ---
              --- ((K/2) + z mod (K/2)) *
            * As *
             * (int[b/(K/2)]) --- ((K/2) + y) *
            * Eint[b/(K/2)] *
            * (b mod (K/2)) --- (0) *
            * Hb];
    z++;
   }
   y++;
  }
 }
 // SUMMARY OF PATHS
 for (c = 0; c < (K/2)^(t-1); c++)
  print("Path(%d) = %s\n",c,Path(c));
}
```

Regarding evaluation and verification of the VM migration algorithms proposed, some executions showing all case scenarios may do it, thus considering source and destination Hosts being 1-hop away, 2-hops away or 3-hops away with a generic K. Let us focus on the algorithm DeviceList as the algorithm DeviceAndPortsList is just an extension of the former showing the port identifiers involved for each device.

First of all, let us take a scenario being IntraNet. The model considers both Hosts being 1-hop away with just one path between them as there is just a single Edge switch defining the only path for any pair of Hosts hanging out of it. Therefore, the first conditional sentence in the algorithm is going to hold and the Edge switch is going to be identified.

Then, let us take an IntraPod scenario. The model considers both Hosts being 2-hops away with K/2 redundant paths in between as both Hosts share the same Pod. Each Aggregation switch within that single Pod defines a different path being reached from the same source Edge switch and being redirected to the same destination Edge switch. Thus, the second conditional sentence in the algorithm is going to hold and the components of each path are going to be identified.

Finally, let us take an InterPod scenario. The model considers both Hosts being 3-hops away with $K^2/4$ redundant paths between both as both Hosts stand in different Pods. Hence, each Core switch defines a different path being reached from one of the Aggregation switches in the source Pod and being redirected to one of the Aggregation switches in the destination Pod. Therefore, the else sentence in the conditional sentence is going to hold and all the components of each path are going to be identified.

To round it all up, the use of this algorithm matches what is happening in a real Fat Tree architecture regarding all possible case scenarios. Therefore, the model may be considered as verified.

VII. CONCLUSIONS

In this paper, the VM migration process between a given source host and a given destination host, both being interconnected through a Fat Tree architecture, is studied. First of all, migration types are exposed noting that live VM migration is the most interesting one and, among all its variations, iterative pre-copy is the most efficient regarding the trade-off between the downtime and the total migration time.

Then, Clos networks are presented and a comparison between Leaf and Spine and Fat Tree architectures is introduced leading to the consideration of Fat Tree as being more scalable. That is why it is selected to build an algorithmic model regarding VM migration.

Eventually, each of the three layers of the Fat Tree and an additional layer for the hosts, where VMs are located, are modelled. The behaviour of each layer is expressed in terms of receiving a VM owned by a user coming from a source host and going to a destination host and, in turn, sending it to the optimal path to reach that destination with the minimal possible number of hops. If that number of hops is more than one, those are redundant paths, which are all optimal. So, the model gives them all.

As a sort of the proof of the concept, two algorithms are proposed. The first one lists the devices to be traversed through all redundant paths available from a given source host to a given destination host. The second one is an extension of the former quoting the ports involved in each link through each redundant path.

In conclusion, the algorithmic model proposed, which is based on arithmetic operations, succeeds in expressing the VM migration process from one host to another in a Fat Tree architecture.

REFERENCES

[1] C. S. R. Prabhu, "Overview - fog computing and internet-of-things (IoT)", *EAI Endorsed Transactions on Cloud Systems*, vol. 17, no. 10, pp. 5. DOI: 10.4108/eai.20-12-2017.154378.

[2] M. R. Anawar, S. Wang, M. A. Zia, A. K. Jadoon, U. Akram, and S. Raza, "Fog computing: An overview of big IoT data analytics", *Wireless Communications and Mobile Computing (WCMC)*, vol. 2018, pp. 22, 2018. DOI: 10.1155/2018/7157192.

[3] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments", *ACM Transactions on Internet Technology*, vol. 19, no. 1, pp. 9–21, 2018. DOI: 10.1145/3186592.

[4] F. Al-Doghman, Z. Chaczko, A. R. Ajayan, and R. Klempous, "A review on fog computing technology", in *Proc. 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. DOI: 10.1109/SMC.2016.7844455.

[5] A. Khakimov, A. Muthanna, and M. S. A. Muthanna, "Study of fog computing structure", in *Proc. 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. DOI: 10.1109/EIConRus.2018.8317028.

[6] J. Abdelaziz, M. Adda, and H. MCheick, "An architectural model for fog computing", *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 10. no. 1, pp. 21–25, 2018. DOI: 10.5383/JUSPN.10.01.003.

[7] K. Gilly, S. Filiposka, and A. Mishev, "Supporting location transparent services in a mobile edge computing environment", *Advances in Electrical and Computer Engineering (AECE)*, vol. 18, no. 4, pp. 11–22, 2018. DOI: 10.4316/AECE.2018.04002.

[8] M. Musolesi and C. Mascolo, "Mobility models for systems evaluation", *Mobility Models for Systems Evaluation - A Survey*. Springer, pp. 43–62, 2009. DOI: 10.1007/978-3-540-89707-1_3.

[9] A. Hess, K. A. Hummel, W. N. Gansterer, and G. Haring, "Data-driven human mobility modeling: A survey and engineering guidance for mobile networking", *ACM Computer Surveys (CSUR)*, vol. 48, no. 3, 2016. DOI: 10.1145/2840722.

[10] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research", *Wireless Communications and Mobile Computing (WCMC)*, vol. 2, no. 5, pp. 483–502, 2002. DOI: 10.1002/wcm.72.

[11] P. J. Roig, S. Alcaraz, K. Gilly, and C. Juiz, "Study on mobility and migration in a fog computing environment", in *Proc. 2018 22nd International Conference Electronics*. DOI: 10.1109/ELECTRONICS.2018.8443636.

[12] O. Hesham and G. A. Wainer, "Centroidal particles for interactive crowd simulation", in *Proc. of the Summer Computer Simulation Conference*, 2016.

[13] O. Hesham and G. A. Wainer, "Context-sensitive personal space for dense crowd simulation", in *Proc. of the Symposium on Simulation for Architecture & Urban Design*, Toronto, Canada, 2017. DOI: 10.22360/simaud.2017.simaud.019.

[14] T. Bosse, M. Hoogendoorn, M. Klein, J. Treur, C. van der Wal, and A. van Wissen, "Modelling collective decision making in groups and crowds: Integrating social", *Autonomous Agents and Multi-Agent Systems (AAMAS)*, vol 27, no. 1, pp. 52–84, 2013. DOI: 10.1007/s10458-012-9201-1.

[15] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K. R. Choo, and M. Dlodlo, "From cloud to fog computing: A review and a conceptual live VM migration framework", *IEEE Access*, vol. 5, pp. 8284–8300. DOI: 10.1109/ACCESS.2017.2692960.

[16] P. Kaur and A. Rani, "Virtual machine migration in cloud computing", *International Journal of Grid Distribution Computing*, vol. 8, no. 5, pp. 337–342. DOI: 10.14257/ijgdc.2015.8.5.33.

[17] L. Chaufournier, P. Sharma, F. Le, E. Nahum, P. Shenoy, and D. Towsley, "Fast transparent virtual machine migration in distributed edge clouds", in *Proc. of the Second ACM/IEEE Symposium on Edge Computing*, 2017. DOI: 10.1145/3132211.3134445.

[18] D. Rosário *et al.*, "Service migration from cloud to multi-tier fog nodes for multimedia dissemination with QoE support", *Sensors*, vol. 18, no. 2, p. 329. DOI: 10.3390/s18020329.

[19] M. Forsman, A. Glad, L. Lundberg, and D. Ilie, "Algorithms for automated live migration of virtual machines", *Journal of Systems and Software*, vol. 101, pp. 110–126, 2015. DOI: 10.1016/j.jss.2014.11.044.

[20] Y. Ruan, Z. Cao, and Z. Cui, "Pre-filter-copy: Efficient and self-adaptive live migration of virtual machines", *IEEE Systems Journal*, vol. 10, no. 4, pp. 1459–1469, 2016. DOI: 10.1109/JSYST.2014.2363021.

[21] P. C. Nayak, D. Garg, A. K. Shakva, and P. Saini, "A research paper of existing live VM migration and a hybrid VM migration approach

in cloud computing", in *Proc. of 2nd International Conference on Trends in Electronics and Informatics*, 2018. DOI: 10.1109/icoei.2018.8553741.

[22] C. Clos, "A study of non-blocking switching networks", *The Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[23] A. Singh *et al*., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network", *Communications of the ACM*, vol. 59, no. 9, pp. 88–97. DOI: 10.1145/2975159.

[24] R. Rojas-Cessa and C. Lin, "Scalable two-stage clos-network switch and module-first matching", in *2006 Workshop on High Performance Switching and Routing*, pp. 303–308. DOI: 10.1109/HPSR.2006.1709725.

[25] X. Yuan, "On nonblocking folded-clos networks in computer communication environments", in *Proc. of 2011 IEEE International Parallel & Distributed Processing Symposium*, Anchorage, AK, USA. DOI: 10.1109/IPDPS.2011.27.

[26] F. Hassen and L. Mhamdi, "High-capacity clos-network switch for data center networks", in *Proc. of IEEE ICC 2017 Next Generation Networking and Internet Symposium*, France, Paris. DOI: 10.1109/ICC.2017.7997147.

[27] K. C. Okafor, "Leveraging fog computing for scalable IoT datacenter using spine-leaf network topology", *Journal of Electrical and Computer Engineering (JECE)*, vol. 2017, 2017. DOI: 10.1155/2017/2363240.

[28] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture", in *Proc. of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 63–74. DOI: 10.1145/1402946.1402967.