Formal Algebraic Specification of an IoT/Fog Data Centre for Fat Tree or Leaf and Spine architectures

Conference Paper · June 2020

DOI: 10.1109/ICECCE49384.2020.9179445

CITATIONS
3

READS
75

5 authors, including:

Pedro Juan Roig
Universidad Miguel Hernández de Elche
35 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Salvador Alcaraz
Universidad Miguel Hernández de Elche
53 PUBLICATIONS **137** CITATIONS
SEE PROFILE

Katja Gilly
Universidad Miguel Hernández de Elche
66 PUBLICATIONS **273** CITATIONS
SEE PROFILE

Sonja Filiposka
Ss. Cyril and Methodius University in Skopje
140 PUBLICATIONS **840** CITATIONS
SEE PROFILE

Formal Algebraic Specification of an IoT/Fog Data Centre for Fat Tree or Leaf and Spine architectures

Pedro Juan Roig
Department of Computer Engineering
Miguel Hernández University
Elche (Alicante), Spain
pedro.roig@graduado.umh.es

Salvador Alcaraz
Department of Computer Engineering
Miguel Hernández University
Elche (Alicante), Spain
salcaraz@umh.es

Katja Gilly
Department of Computer Engineering
Miguel Hernández University
Elche (Alicante), Spain
katya@umh.es

Sonja Filiposka
Faculty of Computer Science and Eng.
Ss. Cyril and Methodius University
Skopje, North Macedonia
sonja.filiposka@finki.ukim.mk

Noura Aknin
Faculty of Science
Abdelmalek Essaadi University
Tétouan, Morocco
noura.aknin@uae.ac.ma

***Abstract*—Fog computing is an evolution of cloud computing paradigm, whose key point is the location of computing resources at the edge of the network. Data center facilities at the fog level are smaller than those at the cloud level, but nevertheless, they may share similar topologies, such as fat tree or leaf and spine architectures. In this paper, a formal algebraic specification of an IoT/Fog environment based on each of both architectures is presented, where users may be moving around and their associated computing assets are meant to migrate among hosts in order to follow their respective users so as to be as close as possible to them.**

Keywords—ACP, fog computing, IoT, formal protocol specification, networking.

I. INTRODUCTION

Cloud computing paradigm has been a reference in the last years among distributed computing models due to its advantages against traditional computing. Some of those are cost reduction in device acquisition, configuration and maintenance, the ubiquitous access to information anywhere, anytime and anyhow, the improvement of backup management, and the flexibility in meeting actual computing needs adequately [1].

Nevertheless, this paradigm is not up to all scenarios, such as real-time applications, where latency is a key factor, or when dealing constrained devices, as it happens with IoT devices [2]. In order to cope with it, cloud philosophy has been adapted to such environments by means of getting the computer assets to the edge of the network, thus getting them closer to the end user, hence reducing latency and the need of bandwidth. This new approach has been redefined as fog computing paradigm [3].

Special attention may be paid to the moving IoT devices [4], as their computing assets, such as virtual machines (VMs) or containers, should be as close as possible to them, therefore, those might try to follow their associated users while moving around, in order to minimise latency and the usage of bandwidth. This movement of VMs through different hosts within a fog computing facility is known as migration [5].

In order to study VM migration, it is necessary first to establish the type of architecture in the fog computing data centre (DC). As fog computing is an evolution of cloud computing, fog DCs share some of the topology designs being in use in cloud DCs, such as fat tree [6], leaf and spine [7] or n-hypercube topologies [8].

VM migration among hosts is a complex operation, having some approaches available, such as cold migration, hot migration and life migration [9], that being the most interesting one as it carries out the migration process in the most transparent manner for the user.

Supposing live VM migration is selected, there are some three parameters to rate its performance, such as the downtime, the total migration time and the amount of dirty pages migrated [10], where the best solution is to consider a tradeoff between the first and the second one, as the third one may not be known beforehand.

For that tradeoff to be successful, memory transfer is key, which may be performed with different techniques, but the pre-copy migration seems to be the one giving better performance due to its efficiency [11]. However, the models considered herein will not take into account all of that and will be focused only on algebraic and arithmetic properties.

In this paper, the objective is twofold. First, we will try to achieve a formal algebraic specification of a fat tree architecture with a generic K value in order to model the VM migration process between a source host and a destination host, both being part of a fat tree topology.

Furthermore, that topology is to be changed by a leaf and spine architecture and a brand-new formal algebraic specification is going to be used to model that same VM migration procedure, so as to compare the behaviour of both.

In both case scenarios, it is going to be supposed that VM migration processes take places in a distributed manner, meaning that there is neither an orchestrator nor any other control entity dictating the path to be followed from a host source to a host destination within the DC. Therefore, each step of such a path is going to be worked out by means of arithmetic calculations depending on source and destination.

The organisation of this paper is as follows: first of all, Section 2 presents an ACP overview, then, Section 3 presents a formal algebraic specification for a generic fat tree architecture, next, Section 4 replicates the previous study for a generic leaf and spine architecture, and finally, the Section 5 draws some conclusions about the whole study.

II. ACP OVERVIEW

The formal algebraic specifications proposed are going to be designed by using Algebra of Communicating Processes (ACP), which is a type of process algebra within the family of abstract algebras focused on reasoning about relationships within distributed systems [12].

ACP defines each entity in a concurrent system as a process and each one is described as process terms stating its properties and liaisons with other peers [13]. Those relations are composed by atomic actions, such as send a message d, given by the expression $s_i(d)$, or read a message d, stated by $r_i(d)$, through a communication channel i.

The logical flow in which those atomic actions takes place may be customized by the use of operators [14], these being the main ones: the sequential operator, where one process gets executed and the next one does it after its completion, is stated by the $\cdot$ sign; the alternate operator, where the choice between the execution of two processes is taken, is given by the $+$ sign; the concurrent operator, where two processes get executed at the same time, is cited by the $\|$ sign; the conditional operator, where a decision is taken depending on the delivery of a certain condition, is denoted by the string $TRUE \lhd condition \rhd FALSE$.

Some more extra operators are given by the Expansion Theorem by Bergstra and Klop [15], which expresses the concurrent (merge) operator $\|$ in terms of left merge $\lfloor$ and communication merge $|$ in order to make easier this kind of calculations; the encapsulation operator ∂_H to force atomic actions involved in internal channels to transform into communications or otherwise go to deadlock, and abstraction operator τ_I to hide internal communications, showing only the external behaviour of the model studied.

At this stage, the external behaviour obtained for the model and that of the real system, expressed in ACP syntax and semantics, may be compared, and if both share the same string of actions and also the same branching structure, it may be stated that both are rooted branching bisimilar [16], being a sufficient condition to get the model verified [17].

III. ALBEBRAIC SPECIFICATION FOR FAT TREE

Fat tree is a three-layer architecture derived from Clos networks [18], where crossbar switches employed to switch telephone calls have been substituted for commodity network switches, making it one of the preferred solutions for Data Centre implementations.

Regarding the three layers involved in fat tree [19], the first one is called Edge, the second one is named Aggregation and the third one is labeled Core. It is to be pointed out that all Hosts being part of a fat tree DC hang on the Edge layer switches and the path for a VM to be migrated from any given source host to any given destination host may be one, two or three hops away, depending on the minimum number of layers necessary to establish that path.

Fat tree topology is divided into Pods, defined as a set composed by the same number of switches on Edge and Aggregation layer where there is full mesh connectivity, in a way that there is a connection between any given switch on Edge layer and all of the switches on Aggregation layer. Additionally, each Pod has full mesh connectivity with all the switches situated on the Core layer.

The setup of a fat tree topology gets fully influenced by parameter K, being that a positive even number, as it dictates the layout of all devices involved. Likewise, all connections among different devices are also heavily biased by parameter K, as long as the oversubscription rate is 1:1, because any other rate would not show all expected links.

The aforesaid influence on K parameter may be appreciated in the values shown in Table I for an oversubscription rate 1:1.

TABLE I. FAT TREE MOST RELEVANT VALUES

	K-ary	***K=8***	***K=4***
Number of Pods	k	8	4
Number of switches in a Pod	k	8	4
Number of ports per switch	k	8	4
Number of Hosts per Edge switch	k/2	4	2
Num. Aggregation switches within a Pod	k/2	4	2
Num. Edge switches within a Pod	k/2	4	2
Num. Hosts within a Pod	$(k/2)^2$	16	4
Number of connections in a Pod	$3\cdot(k/2)^2$	48	12
Total Core switches in a topology	$(k/2)^2$	16	4
Total Aggregation switches in a topology	$k^2/2$	32	8
Total Edge switches in a topology	$k^2/2$	32	8
Total switches in a topology	$5\cdot(k/2)^2$	80	20
Total Hosts in a topology	$k^3/4$	128	16
Total connections in a topology	$3\cdot k^3/4$	384	48
Paths between two hosts being 1-hop away	$(k/2)^0$	1	1
Paths between two hosts being 2-hop away	$(k/2)^1$	4	2
Paths between two hosts being 3-hop away	$(k/2)^2$	16	4

Regarding the nomenclature of the devices involved, items at each layer are enumerated from left to right, going from zero to the precedent integer quoted by the previous table. In fact, Table II shows the lower and upper bounds for each type of item present in a certain layer, along with its acronym to be used within the algebraic model proposed.

Each layer is identified by a number, where digits 1 to 3 represent the layers of the fat tree topology in ascending order, and digit 0 represents where the hosts are. Besides, values 1 to 3 identify the minimum number of hops away between any two given hosts attached to the topology.

TABLE II. ACRONYM AND INDEX FOR EACH LAYER IN FAT TREE

#	***Layer where the item lays***	***index***	***Lower bound***	***Upper bound***
0	Host (H)	h (H_h)	0	$(k^3/4) - 1$
1	Edge (E)	i (E_i)	0	$(k^2/2) - 1$
2	Aggregation (A)	j (A_j)	0	$(k^2/2) - 1$
3	Core (C)	l (C_l)	0	$(k^2/4) - 1$

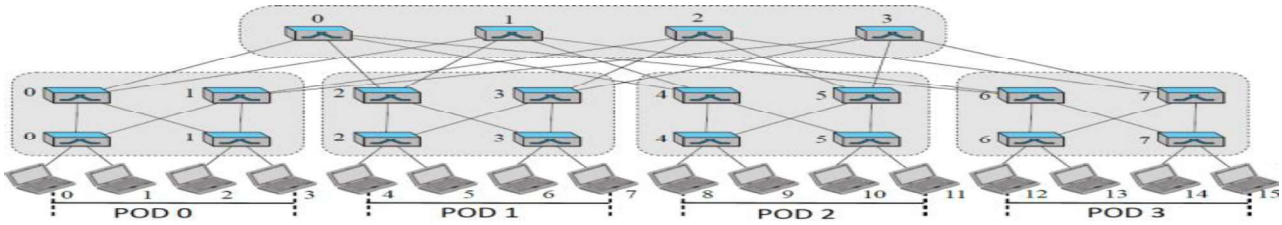

Fig. 1. Fat tree topology for K=4 and oversubscription rate 1:1

Likewise, Pods are identified from left to right, where items belonging to a certain Pod will be spotted in the model by means of the basic arithmetic operators, along with the integer division (*int*) and the modulo operation (*mod*).

As an example, Fig. 1 depicts this nomenclature applied to the specific case where K=4 with oversubscription rate 1:1, thus exhibiting all available links.

With respect to the ports of each of those items, Fig. 2 shows their respective layouts [20]. Basically, items situated at host layer are supposed to be servers with a single link to the proper upper switch, being that link called 0. In order to distinguish the source and destination hosts from the rest of them, the former will be quoted as host a (H_a) and the latter as host b (H_b).

As per the ports in the different kind of switches, each switch standing on layer 1 (Edge) are supposed to have half of their ports looking downwards and the other half looking upwards, hence, those ports are identified from *0* to *(k/2)-1*, going from left to right, on the lower half, and then, from *k/2* to *k-1*, from left to right as well, on the upper half. With respect to switches staying on layer 2 (Aggregation), they have the same layout, so they share the same way to name the ports. Finally, switches being on layer 3 (Core) have all their ports looking downwards, thus, they will be quoted from *0* to *k-1*, going from left to right.

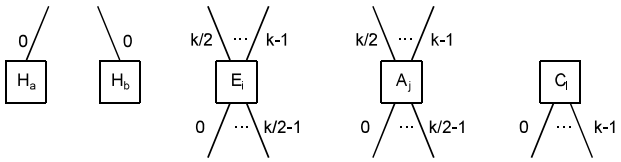

Fig. 2. Ports on each layer in the model for fat tree topology

All devices will be working in a distributed manner, thus doing it concurrently. In the model proposed, it is assumed that no control entity is managing the path from source to destination, but it gets calculated in a hop by hop fashion. To do so, let us assume that the data being sent from a given source host contain not only the migrated VM but also the values identifying source and destination hosts. This way, both numbers may be used in each item along the path to resolve where the next step is going to be, all the way from source to destination.

Regarding the offered traffic, switches do not know which port it is coming from, hence, all switches may be listening through all ports on a regular basis. With respect to the carried traffic, it may be sent over all uplink ports in case of upstream traffic, or otherwise, it may be sent only on the way to destination in case of downstream traffic.

As per the expressions to model the different types of devices, let us start with the hosts, where the source H_a starts the migration process by sending VM across and the destination H_b listens to receive it, whilst the rest of them keep listening but they are idle during this transaction.

Specifically, source host may be described by this recursive expression: $H_a = s_0(d) \cdot H_a$, where destination host may do it by this one: $H_b = r_0(d) \cdot H_b$. Extending this to include all hosts, considering that all hosts may be ready to send at any moment, otherwise they are listening to the channel for possible new arrivals, and taking into account that all hosts work in a concurrent manner, which will be denoted by the symbol $\|$, the overall expression to exhibit the behaviour of all hosts goes like this, where send and receive actions bear two parameters separated by a comma, as the first one indicates the device involved and the second one its port involved in the transaction:

$$H_h = \prod_{h=0}^{k^3/4} \left(s_{Hh,0}(d) + r_{Hh,0}(d) \right) \cdot H_h \tag{1}$$

Applying the same rules, the behaviour of switches standing on layer 1 of the fat tree architecture (Edge), on layer 2 (Aggregation) and on layer 3 (Core) may be modelled according to the following expressions:

$$E_i = \prod_{i=0}^{\frac{k^2}{2}-1} \left(\sum_{m=0}^{\frac{k}{2}-1} \left(r_{Ei,m}(d) \cdot \left(s_{Ei,b \bmod (k/2)}(d) \lhd \mathrm{int}\left(\frac{a}{(k/2)} \right) = \mathrm{int}\left(\frac{b}{(k/2)} \right) \rhd \sum_{m'=k/2}^{k-1} s_{Ei,m'}(d) \right) \right) + \sum_{m'=k/2}^{k-1} r_{Ei,m'}(d) \cdot s_{Ei,b \bmod (k/2)}(d) \right) \cdot E_i \tag{2}$$

$$A_j = \prod_{j=0}^{\frac{k^2}{2}-1} \left(\sum_{n=0}^{\frac{k}{2}-1} \left(r_{Aj,n}(d) \cdot \left(s_{Aj,\mathrm{int}\left(b/(k/2) \right)}(d) \lhd \mathrm{int}\left(\frac{a}{(k/2)^2} \right) = \mathrm{int}\left(\frac{b}{(k/2)^2} \right) \rhd \sum_{n'=k/2}^{k-1} s_{Aj,n'}(d) \right) \right) + \sum_{n'=k/2}^{k-1} r_{Aj,n'}(d) \cdot s_{Aj,\mathrm{int}\left(b/(k/2) \right)}(d) \right) \cdot A_j \tag{3}$$

$$C_l = \prod_{l=0}^{\frac{k^2}{4}-1} \left(\sum_{p=0}^{k-1} \left(r_{Cl,p}(d) \cdot s_{Cl,\mathrm{int}\left(b/(k/2)^2 \right)}(d) \right) \right) \cdot C_l \tag{4}$$

Eventually, the overall recursive model Z for the whole fat tree topology would be specified by executing all items from all layers in a concurrent manner:

$$Z = \left(H_h \parallel E_i \parallel A_j \parallel C_l\right) \cdot Z \tag{5}$$

It is to be noted that the arithmetic condition proposed to find out whether two hosts are connected to the same Edge switch is $\text{int}\left(a/(k/2)\right) = \text{int}\left(b/(k/2)\right)$, whereas the one to know if two hosts are connected to the same Pod is $\text{int}\left(a/(k/2)^2\right) = \text{int}\left(b/(k/2)^2\right)$. With that in mind, the aforementioned expressions modelling the three layers of switches may be easily understood.

Regarding all switches on the Edge layer, all lower ports are waiting to receive a message from a source host, and when that happens, such a message is forwarded on to either only the destination host if it is hanging on the same Edge switch, or otherwise, through all upper ports. Likewise, all upper ports are waiting to receive a message, and when that comes about, it is forwarded on just to the destination host.

With respect to all switches on the Aggregation layer, the behaviour is quite similar, with the difference that all lower ports redirect all incoming messages from a given Edge switch either to the proper Edge switch where the destination host is connected, if it happens to be in the same Pod, or otherwise, through all upper ports. And on the other hand, all upper ports redirect all incoming messages to the Edge switch where the destination host is hanging.

Respecting all switches on Core layer, they just wait to get new messages from a given Aggregation switch to forward them on to the proper Aggregation switch within the Pod where the destination host is located.

The aforementioned model is able to transport data, such as VMs, from source host H_a to destination host H_b, regardless where about in the topology they are situated, thanks to its tree-like structure [21]. As a matter of fact, three case scenarios may appear, according to the way both hosts are interconnected [22], matching real fat tree layout:

1) Through the same Edge switch, so they are one-hop away. They are considered to share the same network, thus having just *1* unique path between them. If this is the case, both hosts may be called neighbors.

2) Through all Aggregation switches within a given Pod, so they are two-hops away. They are considered to share the same Pod, hence having as many paths between them as Aggregation switches in a Pod, this is, *k/2* paths. In such a case, both hosts might be named superneighbors.

3) Through all Core switch around the whole topology, so they are three-hops away. They do not share the same Pod, therefore, there are as many paths between them as the total tally of Core switches, this is, $(k/2)^2$ paths. In such a case, both hosts might be named hyperneighbors.

Regarding the verification of the model, it may be performed in different ways, although it is going to be presented by applying structural induction and also by means of ACP concept of rooted branching bisimilarity.

As per the structural induction, a fat tree topology may well be considered as a set *M* of *m-ary* trees, being *m* any natural number. Hence, the point is to proof that any full and complete *m-ary* tree fulfills the definition of a tree, therefore, it will imply connectivity among any given pair of leaves, as a tree provides a continuous path among all their leaves (those being considered the hosts in a fat tree setup).

In order to do it, let us consider a tree with a single node *r* is inside *M*. In this case, if *r* is a node and $\{T_1, T_2, \ldots, T_M\}$ are disjoint *m-ary* trees, this is, $\{T_1, T_2, \ldots, T_M\} \in M$, then the tree $T = (r, T_1, T_2 \ldots, T_M)$ is also an *m-ary* tree, this is, *T* is inside *M.* And the same applies for a set of *m-ary* trees.

As per the ACP approach, let us apply the encapsulation operator to the final model X presented above to each pair of neighbouring layers presented therein, as non neighbouring layers do not interact with each other, so they will go deadlock and will be discarded.

It is to be noted that the communication actions show both ends of each channel, separated by a double dash, where at each end it is shown the device and its port. Also, port variables *m* and *n* may be single and double quoted in order to point out that they may belong to different switches.

In summary, only internal communications will make it through, along with external inputs and outputs. First, in (6), we are focusing on the links between Host layer and Edge layer, only if there is a shared channel, through port *m* of E_i. Then, in (7), we are looking at the links between Edge layer and Aggregation layer, just if there is a shared channel, through port *n* of A_j. Finally, in (8), we are selecting the links between Aggregation layer and Core layer, just if there is a shared channel, through port *p* of C_l.

$$\partial_H\left(H_h \parallel E_i\right) = \left[\begin{array}{l} c_{Ha,0\text{--}Ei,m}(d) \cdot \left(c_{Ei,b \bmod (k/2)\text{--}Hb,0}(d) \lhd \text{int}\left(\frac{a}{(k/2)}\right) = \text{int}\left(\frac{b}{(k/2)}\right) \rhd \sum_{m'=k/2}^{k-1} s_{Ei,m'}(d) \right) + \\ + \sum_{m'=k/2}^{k-1} r_{Ei,m'}(d) \cdot c_{Ei,b \bmod (k/2)\text{--}Hb,0}(d) \end{array} \right] \cdot \partial_H\left(H_h \parallel E_i\right) \tag{6}$$

$$\partial_H\left(E_i \parallel A_j\right) = \left[c_{Ei,m'\text{--}Aj,n}(d) \cdot \left(\begin{array}{l} c_{Aj,\text{int}\left(b/(k/2)\right)\text{--}Ei',m''}(d) \lhd \text{int}\left(\frac{a}{(k/2)^2}\right) = \text{int}\left(\frac{b}{(k/2)^2}\right) \rhd \sum_{n'=k/2}^{k-1} s_{Aj,n'}(d) + \\ + \sum_{n'=k/2}^{k-1} r_{Aj,n'}(d) \cdot c_{Aj,\text{int}\left(b/(k/2)\right)\text{--}Ei',m''}(d) \end{array} \right) \right] \cdot \partial_H\left(E_i \parallel A_j\right) \tag{7}$$

$$\partial_H\left(A_j \parallel C_l\right) = \left[c_{Aj,n'\text{--}Cl,p}(d) \cdot c_{Cl,\text{int}\left(b/(k/2)^2\right)\text{--}Aj',n''}(d) \right] \cdot \partial_H\left(A_j \parallel C_l\right) \tag{8}$$

At this stage, it is time to apply the abstraction operator, and that would hide all internal communications. At this point, if we consider the model as in (5), where the host layer may be seen as the lower most layer, and above that, all fat tree switching structure is built upon, then all communications may well be seen as internal.

In this case, all communications shown in (6)-(8) will get masked, and also the send and read actions appearing therein will do the same as they represent an action to a different layer than those shown in the corresponding expression. Hence, no external communication will take place and the behaviour of the overall topology might be perceived as a closed system, with no external interaction.

$$\tau_I\left(\partial_H\left(H_h \,||\, E_i \,||\, A_j \,||\, C_l\right)\right) = \varnothing \tag{9}$$

However, if it is considered just the fat tree switching structure, thus taking out of the expression the host layer, then paths from the source host and to the destination host would prevail after applying the abstraction operation, as being external paths, whereas the rest of terms will be hidden. This would make look the overall expression for the switching layers as an open system, with two paths for external interaction.

$$\begin{aligned} &\tau_I\left(\partial_H\left(E_i \,||\, A_j \,||\, C_l\right)\right) = \\ &= r_{Ha,0}(d)\cdot s_{Hb,0}(d)\cdot \tau_I\left(\partial_H\left(E_i \,||\, A_j \,||\, C_l\right)\right) \end{aligned} \tag{10}$$

On the other hand, the external behaviour of a real fat tree switching infrastructure, leaving out the host layer, is just the internal architecture to interconnect the different hosts hanging on it, where a source host transmits some information getting into the system, and a destination host receives such information getting out of the system.

$$X = r_{Ha,0}(d)\cdot s_{Hb,0}\cdot X \tag{11}$$

By comparing both previous expressions, the one obtained for the external behaviour of the model (10) and the one achieved for the external behaviour of the real system (11), it is obvious that both are recursive expressions with the same terms being multiplied. Therefore, it is clear that both recursive variables may match in this context:

$$X \leftrightarrow \tau_I\left(\partial_H\left(H_h \,||\, E_i \,||\, A_j \,||\, C_l\right)\right) \tag{12}$$

Hence, it may be concluded that both expressions are rooted branching bisimilar, and that makes the model proposed gets verified.

IV. Albebraic Specification for Leaf and Spine

Leaf and spine is a two-tier architecture, derived from Clos networks, whose main characteristic is that all switches in the first layer, called leaf, have full mesh connectivity with all switches in the second layer, called spine [23]. This provides better performance and redundancy for hosts, them hanging on leaf switches, although scalability issues may arise as the number of necessary switches grows [24].

Leaf and spine designs are not influenced by any special parameter, as it happened with fat tree, so there is some degree of freedom when dealing with models [25]. In this paper, it is going to be considered that the number of leaf switches is given by variable p, whereas the number of spine ones is stated by q. This may be appreciated in Fig. 3.

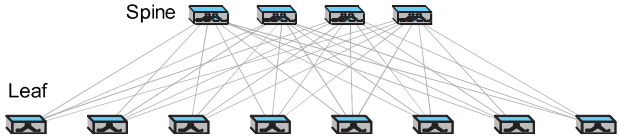

Fig. 3. Leaf and spine topology for p=8 and q=4

In order to model each layer, being leaf, spine and host, Fig. 4 presents them along with their list of ports [26].

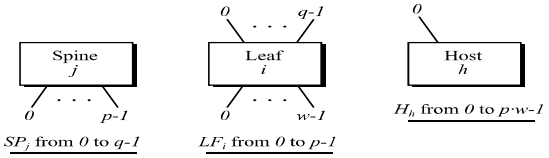

Fig. 4. Ports on each layer in the model for leaf and spine topology

The study of this model is pretty similar to that carried out for the fat tree architecture, in a way that a distributed system is considered, hence, no central entity is governing how the switches organise the path from a source host to a destination host, but it is done on a hop by hop basis, under the criteria of choosing the shortest paths.

Following the same nomenclature presented in the previous case, where H_a is the source host and H_b is the destination host, here it comes the model showing the behaviour of all hosts for the leaf and spine topology:

$$H_h = \sum_{h=0}^{p\cdot w-1}\left(s_{Hh,0}(d) + r_{Hh,0}(d)\right)\cdot H_h \tag{13}$$

Regarding the lower layer of switches, namely the Leaf, it is to be taken into consideration that, the arithmetic condition proposed to find out whether two hosts are connected to the same Leaf switch is $\text{int}(a/w) = \text{int}(b/w)$.

Comparing this rule to the one expressed for the fat tree topology, it is clear that parameter K plays a key role in the layout of the hosts, whereas leaf and spine topology leaves some degree of freedom to the network designers, as variable w may be chosen arbitrarily.

This variable w accounts for the number of physical hosts linking to each Leaf switch (considering the same number in all of them), and depending on the bandwidth of those ports, the load being supported by the network may increase over the limit, affecting the network performance.

Anyway, applying the same rules as in the previous section, the behaviour of switches standing on lower layer (Leaf) and on upper layer (Spine) may be modelled according to the following expressions:

$$LF_i = \sum_{i=0}^{p-1}\left(\sum_{m=0}^{w-1}\left(r_{LFi,m}(d)\cdot\left(s_{LFi,b \bmod w}(d) \lhd \text{int}\left(\frac{a}{w}\right) = \text{int}\left(\frac{b}{w}\right) \rhd \sum_{m'=0}^{q-1} s_{LFi,m'}(d)\right)\right) + \sum_{m'=0}^{q-1} r_{LFi,m'}(d)\cdot s_{LFi,b \bmod w}(d)\right)\cdot LF_i \tag{14}$$

$$SP_j = \sum_{j=0}^{q-1}\left(\sum_{n=0}^{p-1} r_{SPj,n}(d)\cdot s_{SPj,\text{int}(b/w)}(d)\right)\cdot SP_j \tag{15}$$

Eventually, the overall model for leaf and spine would be given by running all devices concurrently:

$$Z = \left(H_h \parallel LF_i \parallel SP_j\right) \cdot Z \tag{16}$$

In this case, it happens the same as in the previous section, in a way that the whole architecture including the hosts may be perceived as a closed system from the point of view of its external behaviour, as no interaction will be appreciated externally:

$$\tau_I\left(\partial_H\left(H_h \parallel LF_i \parallel SP_j\right)\right) = \varnothing \tag{17}$$

However, if it is considered just the leaf and switching structure, the switching architecture may be seen as an open system from the point of view of its external behaviour, with two paths for external interaction:

$$\tau_I\left(\partial_H\left(LF_i \parallel SP_j\right)\right) = r_{Ha,0}(d) \cdot s_{Hb,0}(d) \cdot \tau_I\left(\partial_H\left(LF_i \parallel SP_j\right)\right) \tag{18}$$

As per the external behaviour of real leaf and spine switching infrastructure, it may be seen as if the whole system is ready to receive a message from a given host, and in turn, it will forward it on to another given host, regardless of whether they both are hanging on the same switch or not.

$$X = r_{Ha,0}(d) \cdot s_{Hb,0} \cdot X \tag{19}$$

By comparing both expressions (18) and (19), it is clear that they are both rooted branching bisimilar, as they show one recursive variable being multiplied by the same factors.

$$X \leftrightarrow \tau_I\left(\partial_H\left(H_h \parallel LF_i \parallel SP_j\right)\right) \tag{20}$$

Therefore, it may be concluded that the model proposed has been verified.

V. Conclusions

In this paper, formal algebraic models for fat tree and leaf and spine architectures have been presented, according to the behaviour of each of its layers.

Regarding the topology designs, fat tree is more rigid as it depends heavily on parameter *K*, which constraints all aspects of its design, whereas leaf and spine is looser as it depends on variable *w*, which is chosen at design time. But small differences in design do not make a big deal in order to apply the ACP set of axioms to study them.

First, fat tree topology has been introduced and its key points have been explained, and in turn, arithmetic has been used to describe the behaviour of each of its layers. Later on, a process algebra called ACP has been used to extract the external behaviour of the model, and then, by comparing it to the external behaviour of the real system, it has been proved that both of them run the same string of actions and have the same branching structure, thus the model has been considered as being verified, according to ACP.

On the other hand, leaf and spine topology has been brought in and its key ideas have been exposed, and the same procedure has been applied to it in order to get it verified.

In conclusion, both models have been carefully studied and they both have been verified. As a future work, some other models for Data Center topologies may be built up following the same criteria so as to see whether they also get verified, such as n-hypercube or more complex structures.

References

[1] A. Botta, W. de Donato, V. Persico and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," Future Generation Computer Systems, 2016, Vol. 56, pp. 684-700.

[2] P. Sethi and S.R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," in Journal of Electrical and Computer Engineering, 2017, article ID 9324035, pp. 1-25.

[3] C.S.R. Prabhu, "Overview - Fog Computing and Internet-of-Things (IoT)," in EAI Endorsed Transactions on Cloud Systems, 2017, Issue 10, Article 5, pp. 1-24.

[4] L. Bittencourt et al., "The Internet of Things, Fog and Cloud continuum: Integration and challenges," in Internet of Things, 2018, Vol. 3-4, pp. 134-155.

[5] M.R. Anawar et al, "Fog Computing: An Overview of Big IoT Data Analytics," WCMC'2018, Article ID 7157192, pp. 1-22.

[6] M. Al-Fares, A. Loukissas and A. Vahdat, "A scalable, commodity data center network architecture," in SIGCOMM'2008, pp. 63-74.

[7] M. Alizadeh and T. Edsall, "On the Data Path Performance of Leaf-Spine Datacenter Fabrics," in IEEE 21st Annual Symposium on High-Performance Interconnects, 2013, Vol 1, pp. 71-74.

[8] Z.A. Khan, J. Siddiqui and A. Samad, "Topological Evaluation of Variants Hypercube Network," in Asian Journal of Computer Science and Information Technology, 2013, Vol 3(9), pp. 125-128.

[9] C. Puliafito et al, "Container Migration in the Fog: A Performance Evaluation," in Sensors - Middleware Solutions for Wireless Internet of Things, 2019, Vol. 19(7), Article 1488, pp. 1-22.

[10] O. Osanaiye et al, "From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework," in IEEE Access, 2017, Vol. 5, pp. 8284-8300.

[11] M. Forsman, A. Glad, L. Lundberg and D. Ilie, "Algorithms for automated live migration of virtual machines," in Journal of Systems and Software, 2015, Vol. 101, pp. 110-126.

[12] D.A. Padua, "Encyclopedia of Parallel Computing," Springer, 2011.

[13] L. Lockefeer, D.M. Williams and W. Fokkink, "Formal specification and verification of TCP extended with the Window Scale Option," in Science of Computing Programming, 2016, Vol. 118, No. 1, pp. 3-23.

[14] M. Gazda, W. Fokkink and V. Massaro, "Congruence from the operator's point of view," in Acta Informatica, 2019, pp. 1-23.

[15] J.A. Bergstra and J.W. Klop, "Algebra of communicating processes with abstraction," in T.Computer Science, 1985, Vol. 37, pp. 77-121.

[16] J.F. Groote and M.R. Mousavi, "Modelling and Analysis of Communicating Systems," MIT Press, 2014.

[17] W. Fokkink, "Introduction to Process Algebra," Springer, 2007.

[18] C. Clos, "A study of non-blocking switching networks," in The Bell System Technical Journal, 1953, Vol. 32, Issue 2, pp. 406-424.

[19] Y.H. Lo, Y. Zhang, Yi Chen, H.L. Fu and W.S. Wong, "The Global Packing Number of a Fat-Tree Network," in IEEE Transactions on Information Theory, 2017, Vol. 63, Issue 8, pp. 5327-5335.

[20] P.J. Roig, S. Alcaraz, K. Gilly and C. Juiz, "Modelling VM Migration in a Fog Computing Environment," in Elektronika Ir Elektrotechnika, 2019, Vol. 25, Issue 5, pp. 75-81.

[21] V.P. Bakoev, "Algorithmic approach to counting of certain types m-ary partitions," in in Discrete Mathematics, 2004, Vol. 275, Issues 1-3, pp. 17-41.

[22] D. Li et al., "Scalable and Cost-Effective Interconnection of Data-Center Servers Using Dual Server Ports," in IEEE/ACM Transactions on Networking, 2011, Vol. 19(1), pp. 102-114.

[23] K.C. Okafor, I.E. Achumba, G.A. Chukwudebe and G.C. Ononiwu, "Leveraging Fog Computing for scalable IoT datacenter using Spine-Leaf network topology," in Journal of Electrical and Computer Engineering, Vol. 2017, Article 2363240, pp. 1-11.

[24] M. Alizadeh and T. Edsall, On the Data Path Performance of Leaf-Spine Datacenter Fabrics, in IEEE 21st Annual Symposium on High-Performance Interconnects, 2013, Vol. 1, pp. 71-74.

[25] X. Li, C.H. Lung and S. Majumdar, "Green spine switch management for datacenter networks," in Journal of Cloud Computing, 2016, Vol. 5, Art. 9, pp. 1-19.

[26] P.J. Roig, S. Alcaraz, K. Gilly and C. Juiz, "Modelling a Leaf and Spine Topology for VM Migration in Fog Computing," in Elektronika Ir Elektrotechnika, 2020, in press.