Multi-Layer-Mesh: A Novel Topology and SDN-Based Path Switching for Big Data Cluster Networks

Leandro Batista de Almeida, Damien Magoni, Philip Perry, Eduardo Cunha de Almeida, John Murphy, Anthony Ventresque

Multi-Layer-Mesh: A Novel Topology and SDN-based Path Switching for Big Data Cluster Networks

Leandro Batista de Almeida*†, Damien Magoni‡, Philip Perry*,
Eduardo Cunha de Almeida§, John Murphy*, Anthony Ventresque*
*Lero, School of Computer Science, University College Dublin, Ireland
†UTFPR, Federal Technological University of Parana, Brazil
‡LaBRI, University of Bordeaux, France
§UFPR, Federal University of Parana, Brazil

***Abstract*—Big Data technologies and tools have being used for the past decade to solve several scientific and industry problems, with Hadoop/YARN becoming the "de facto" standard for these applications, although other technologies run on top of it. As any other distributed application, those big data technologies rely heavily on the network infrastructure to read and move data from hundreds or thousands of cluster nodes. Although these technologies are based on reliable and efficient distributed algorithms, there are scenarios and conditions that can generate bottlenecks and inefficiencies, i.e., when a high number of concurrent users creates data access contention. In this paper, we propose a novel network topology called Multi-Layer-Mesh and a path switching algorithm based on SDN, that can increase the performance of a big data cluster while reducing the amount of utilized resources (network equipment), in turn reducing the energy and cooling consumption. A thorough simulation-based evaluation of our algorithms shows an average improvement in performance of 31.77% and an average decrease in resource utilization of 36.03% compared to a traditional Spine-Leaf topology, in the selected test scenarios.**

***Index Terms*—Big data, Hadoop, network topology, SDN.**

I. Introduction

The last decades have seen an exponential growth in the volume of data to be analyzed by computational systems. From Data Warehousing to IoT and Machine Learning systems, the processing needs for large volumes of information has consistently grown. To meet that demand, Big Data systems were developed such as Google File System, MapReduce, Hadoop, and several other tools in the following years. The Big Data ecosystem is composed of several tools and frameworks, that cooperate in order to provide the most appropriate solution for each particular problem, creating a multi-product environment.

There are several components involved in the performance metrics of a Big Data job execution, and the network is one of them. As stated by its name, the data volume is big, beyond the capabilities of a single server, or even a small cluster. For this reason, almost all Big Data systems use some form of distributed file system, in order to cope with both the volume of data, and the need of reading the data at fast pace. The Hadoop Distributed File System is generally used not only by Hadoop, but also by other systems, such as Spark and Impala. The usual Hadoop stack is composed of HDFS at the bottom, the job engine (like Hadoop YARN or MapReduce) over it, and the application layer on the top, with Hive, Spark or MapReduce programs solving the user queries. This stack is shown in Figure 1. In these huge systems, there is a constant pursuit for performance, since the execution time for jobs can take several hours, even days [1]. In this environment, even a minor improvement in performance can save a considerable amount of execution time. The energy consumption and cooling costs are also directly influenced by CPU processing required for running the jobs [2]. Although Hadoop tries to run jobs locally (by reading data from local storage),the total workload of concurrent jobs will eventually have the jobs require chunks of data spread out on several servers in different racks. This will create a bottleneck if the interconnection network throughput is not able to handle the large data transfers needed for job completion.

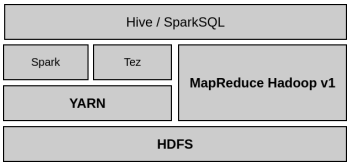

Fig. 1. Common Hadoop stack.

To address this issue, we have developed a novel network architecture, based on a Multi-Layer-Mesh topology, and a SDN-driven forwarding algorithm that can use information about the job execution from the Hadoop main server to organize the network traffic for optimizing the use of available network paths. Our proposal can decrease the number of network equipment used, and increase the network performance for Big Data job execution. Thus, it reduces costs both in equipment acquisition and maintenance, and in energy consumption and cooling. The switches used in our topology must be SDN-enabled and are configured by a SDN controller.

The rest of the paper is organized as follows. Section II presents the state of the art in Hadoop clusters, DCN

topologies and Hadoop simulators. Section III details our novel network topology proposal. Section IV details the experimentation that we have carried out for validating our proposal.

II. Related Work

A. Data Center Topologies and Hadoop Clusters

A Data Center (DC) concentrates a large amount of servers in one physical place. Those servers need to be optimally interconnected to ensure proper communications both intra-DC and inter-DC. There exists more than 40 different types of topologies for setting up a DC network (DCN) [2], [3]. These topologies are usually characterized by the type of switches used (i.e., electronic, optical or wireless) and whether they are switch-centric (i.e., many links between switches) or server-centric (i.e., many links between servers). They are usually statically implemented in hardware. In 2017, a system for implementing a convertible DCN architecture has been proposed in [4] to overcome this issue. It can dynamically change the network topology in order to setup architectures optimized for diverse workloads. It requires the use of small port-count converter-switches. Hybrid electro-optical systems have also been recently proposed to improve performances, such as the Hybrid Fat Tree (HFT) [5]. By introducing wavelength-division multiplexing (WDM) and optical switches in Fat-Tree like topologies, the authors find that these technologies reduce the required number of switches by 45% in their minimum hybrid networks. Inter-rack optical rings have also been proposed for improving the overall throughput of the DCN, by using flexible spectrum allocation with ROADMs [6].

Software-Defined Networking (SDN) is increasingly being deployed in modern DCs and can benefit big data applications in many ways [7], including big data processing, data delivery, programming at runtime for optimizing big data applications, scientific big data architectures, and more specifically for Hadoop scheduling as shown in [8]. By measuring and analyzing the DC traffic, such as presented in [9], topologies and flows can be optimized through dynamic network reconfiguration, thanks to SDN technology. A Hadoop cluster is a computational cluster designed for storing and analyzing huge amounts of unstructured data. These clusters are usually hosted in data centers, and thus have similar network topologies, although their traffic patterns may be different from other applications. Several solutions based on SDN technology have been recently proposed to improve Hadoop performance. Narayan et al. use OpenFlow to provide better link bandwidth utilization for traffic exchanged during the shuffle phase of MapReduce [10]. This, in turn, decreases the time that Reducers have to wait to gather data from Mappers, and therefore shorten the completion time of Hadoop jobs. Similarly, Nejad and Majma have proposed a method based on alpha-beta filter to improve the efficiency of MapReduce in Hadoop clusters by leveraging SDN [11]. Ghalwash et al. investigate the throughput and execution time of Hadoop read/write and sorting operations [12]. They evaluate different network sizes of a Fat-tree topology of OpenFlow switches and they observe improvements for some of their topologies when using OpenFlow rather than regular L2 switching. Our work is different from these works as we propose a new physical topology as well as a SDN-driven forwarding algorithm over it. We aim at reducing the number of network components as much as improving the Hadoop job completion times.

B. Big Data Simulators

Most of the Big Data software and frameworks are designed to run in clusters with hundreds or even thousands of nodes. Testing new technologies can be a difficult and time costly process, and usually those resources are not available, or their use could be more expensive than the available budget. This is the case of this research, when we need to test how the data movement will behave in a multi-rack cluster environment, with different cluster sizes, and with a novel network topology and switching protocol.

And in order to execute our experiments and test the presented hypothesis, we need set several parameters for the scenarios to be simulated, and the simulator should be capable of providing data about the network behavior in the jobs' reading phase, and the related actions, like the distribution of blocks and task placement.

Based on these premises, we developed a simulator to help on researching on Big Data systems performance, the BigDataNetSim [13], tested against physical cluster to assess the accuracy, with published results. This simulator was used for the experiments on this research.

III. The Multi-Layer-Mesh Topology

The main contribution of this research is a novel network topology based on a collection of layer-1 (L1) full meshes interconnected via a layer-2 (L2) mesh. Theoretically, we could have any number of layers, but in this paper we limit the topology to two layers. All switches are located at layer 1. They are partitioned in m groups of n switches. In each group, the n switches form a full mesh: each switch is connected to all the others in the group.The links used inside this full mesh are called **intralinks**. Then, meshes are interconnected by using links connecting some or all switches from one mesh to some or all switches in all other meshes. These links are called **interlinks** and they form the layer-2 mesh. Their number is a parameter of the topology. There are no switches at this second layer. We could add a third layer by partitioning the layer-1 groups into p layer-2 supergroups of q groups. Layer-2 interlinks would interconnect groups in a supergroup, while layer-3 interlinks would interconnect the supergroups themselves. This topology is oriented to Big Data clusters with data intensive applications, such as data analysis and machine learning.

On top of the physical topology, a SDN enabled forward algorithm is used, obtaining information about the big data application running in the cluster and setting up the most appropriate paths for each communication link between the nodes.

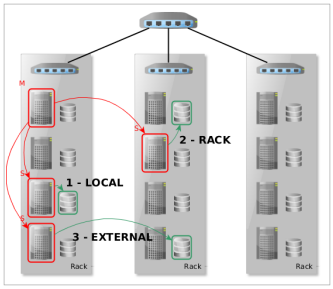

Fig. 2. Example of local (1), same rack (2) and external (3) tasks.

A. Problem Definition

Nowadays, the general scenario for Big Data clusters shows clusters being used for several applications at the same time, due to the involved costs to build and maintain such a large installation. It is common to have several tools and frameworks sharing the same cluster infrastructure, in a multi-tenant configuration. The data sources are also shared by many applications, and the job results are used as inputs for new jobs as well [1]. This causes a high level of data reuse, creating zones of the data space labeled as ”hot data”, shared by many possible concurrent jobs.

In a general way, the data is stored in a distributed fashion in a big data cluster. As the data volume is large, the information is divided into smaller portions and stored in a large array of nodes. This is how the Hadoop Distributed File System (HDFS) works [14], by dividing the files in blocks of 128 MB (default value), and by distributing the blocks in the cluster. Besides, HDFS creates replicas of each block (3 replicas, by default) to ensure data availability and fault recovery. When some job requests a file, the HDFS master node sends to the Application Master, the location of all blocks and replicas of that particular file, and the processing manager decides which block/replica to read.

Hadoop v1 used MapReduce as the main processing manager, and starting with Hadoop v2, YARN (Yet Another Resource Negotiator) is used to manage how a job will be executed, how a job will be divided into smaller tasks, and how to read information from the distributed file system. In general, YARN tries to start a task (a subdivision of a job) on the same node where the information about to be read is located, avoiding any loss of time due to network transfers. As the number of possible tasks running in each node is limited (based on how many concurrent threads can be started, related to the number of CPU cores), this strategy works well when the cluster has a lower number of users and concurrent jobs.

When YARN needs to read a particular block of information to process the data, and all the available cores of all nodes containing that data are depleted, the system starts the task on another node, and reads the information across the network. Initially, YARN tries to start those tasks at least in the same rack where the information is stored, but eventually - as the load increases - tasks will be started in another rack, increasing the network transfer delay. This is shown in Figure 2, with a Local task (reading from the local storage) is shown, compared with a Rack task (reading from the storage of another node in the same rack) and a External task (reading from another node in another rack), with the increased delays for each step.

In the end, as the load increases, the network is put under pressure, and regardless of the used application framework (Spark, MapReduce, Hive, Impala), the completion time for the jobs will increase. The worst case scenario is when the cluster receives multiple concurrent jobs accessing the same data sources (hot data), thereby increasing the chances of reading tasks having to use the network to read the files.

There are several network topologies designed for data center and cluster environments [3], and some of these topologies are adequate for clusters, like Fat Tree or Spine-Leaf, however, those topologies don’t scale well as the number of nodes increases. In Big Data clusters, the number of nodes can easily reach a few thousands of nodes for some scenarios. Then, the related network infrastructure must be designed properly. For a hypothetical cluster with 1000 nodes divided in 50 racks containing 20 nodes each, a regular Spine-Leaf topology requires a number of spine switches up to half the number of racks, in order to maintain the traffic at fair levels, even under high load conditions.

B. The Multi-Layer-Mesh Topology

To solve the described problem, a novel network topology is proposed - the Multi-Layer-Mesh (MLM) topology - along with a switching protocol that uses information from the application (big data application manager). The main goal is to reduce the amount of resources used (equipment and energy) and increase performance.

As described earlier, the physical network topology is based on a partitioning of all the switches in the network into a given number of L1 full meshes. Additional links, called interlinks, interconnecting those L1 meshes together form a global L2 mesh.It is a full mesh for the L1 meshes in the sense that each L1 mesh has at least one link connecting it to each other L1 mesh. The actual racks with the cluster nodes are connected to switches belonging to L1 meshes. An example network can be seen in Figure 3 with every L1 mesh containing 5 switches. In this case, only one interlink between each L1 mesh pair is presented, for the sake of clarity.

There are two parameters to be configured, in order to increase the number of available paths and reduce the bottlenecks: (1) the number of switches in each L1 mesh, and (2) the total number of interlinks between any pair of L1 meshes.

On this example network, the number of intralinks in each L1 mesh is 10 and the total number of interlinks from each mesh to each other is 6, effectively creating a full mesh of meshes. It should be noted that the number of interlinks between a given pair of L1 meshes can be much more than 1 thus leading to more interlinks than just $n(n-1)/2$ (with n the number of L1 meshes). As links are cheaper than equipment - considering copper connections - this topology reduces the amount of used resources, while keeping the traffic

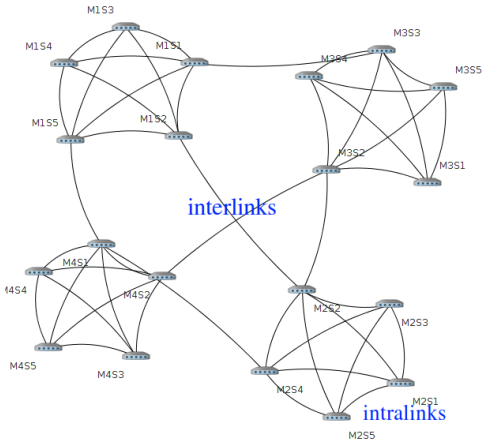

Fig. 3. Example of a Multi-Layer-Mesh topology with full L1 meshes interconnected by interlinks.

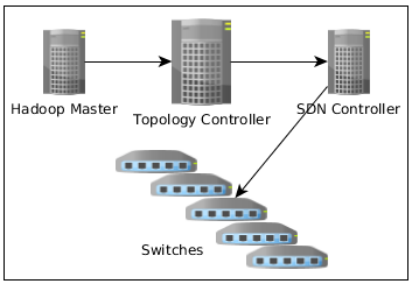

Fig. 4. System architecture overview.

at acceptable levels, eventually surpassing a regular Spine-Leaf topology.

To determine the paths to be used, a switching protocol was designed, based on information from the big data cluster controller, in order to prepare the traffic for the upcoming transfer tasks. The topology is independent from the big data underlying technology, by the use of drivers designed to cope with the singular properties of each used technology. In the experiments of this research, the system was connected to a Hadoop installation, using HDFS as storage, and YARN as the application manager. With the proper driver developed, this system is able to use different storage technologies (Cassandra, S3) and application managers (Spark, Tez) as well. The proposed system reads information about the jobs submitted to the Hadoop cluster, and prepare the paths accordingly. This architecture is shown in Figure 4.

C. Proposed Construction Algorithm

The path table construction algorithm is based on Dijkstra's shortest path algorithm, in a tracing algorithm to generate unique and balanced paths between a given pair of nodes, using one of the available interconnections between the layer-1 meshes. This is based on the fact that racks connected to the same mesh have always a shortest path of distance 1, and the maximum path distance in interconnections is 3, and the minimum is 1. The algorithm goal is the keep the actual distance as close as possible to the minimum, based on the features of a particular set of parameters. There are two algorithms used in the research, one to create the paths table, and another to select the proper path for each communication flow. Those algorithms are described below.

1) Path Table Creation Algorithm: The initial flow table will be created by the Topology Controller based on the shortest paths between each pair of switches in the cluster. As the maximum path is 3, there are potentially more than one path with cost over 1 for a particular pair of switches. For this situation, a list with all the paths with same cost for each pair is generated to help in the path selection algorithm.

A list with all possible hops in the paths is also created. This list contains all possible segments in the network, and counters to keep track of the traffic in each hop.

2) Path Selection Algorithm: When the jobs starts to run, the Topology controller obtains the data about each task in the job (which node is processing the data and which node is storing the data) from the Hadoop controller, and orders the SDN controller to setup a flow for that particular communication pair using the most appropriate available path. Although this setup process take some time to happen, the start of a Hadoop task is also time consuming, so, when the actual task is starting, the corresponding flow is already configured.

The algorithm is based on the following steps, as shown in Algorithm 1. As the local and rack tasks are treated by the same switch or no switch at all, the algorithm focus only on the external tasks, when communication between the racks is needed. For each new external reading task from a Hadoop job, obtain the information about the processing node and the data node from Hadoop controller. Verify if there is a path with cost 1 between the racks containing the two nodes, and select it. If there is no direct connection between the two switches, select the least used least cost path from the list of unique paths between the top of rack switches containing the pair of nodes.

IV. Experiments

Using our simulator, three cluster sizes were tested, using a configuration commonly found in real clusters.

A. Usage of BigDataNetSim Simulator

The common use case for our simulator in this research is to generate a cluster of a particular size, with a particular network topology, create the file system in it, populate the file system with files of a specific size, submit a given number of concurrent jobs, generate the metrics for the network usage and generate the reports about the simulation.

This research focuses on the reading phase of Big Data jobs because it is a time consuming one, and it is generally the only common phase regarding the Big Data technology running on top of it (e.g., Hadoop, Hive, Spark, Impala, etc). The subsequent phases of data processing could vary depending upon the chosen framework and make the simulation task complex or not feasible. In addition, the reading phase is the

Algorithm 1: Path Selection Algorithm

```
// considering i,j as processNode and
   dataNode
input : jobTasks: List< task >, uniquePaths_{i,j}:
        List< path >, steps: List< path_{i,j} >
output: selectedPaths_{task,i,j}: List< steps >
for task ∈ jobTasks do
    // get processing and data node from task
    processNode, dataNode ← getNodePair(task)
    if cost(uniquePaths_{processNode,dataNode}) = 1 then
        // get the path between the nodes using
           switches inside same layer-1 mesh or
           with direct connection
        path ← directPath(processNode, dataNode)
    else
        // find path with less cost
        cost ← minimunCost(processNode, dataNode)
         path ←
         lessUsedPath(processNode, dataNode, cost)
    // add select path to list of used paths
    selectedPaths ← path // increases the usage
       counter of each hop in path
    hops ← increaseHopCount(path)
return selectedPaths
```

most demanding in terms of network usage, as the subsequent phases generally only transmit results (partial or full), with much less volume. As the simulator supports different network topologies and protocols, it is possible to obtain the network metrics from the reading phase in a faster way.

The main parameters used on this research are:

- Number of layer-1 meshes: number of first level fully meshed groups of access switches in the topology.
- Number of access switches per layer-1 mesh: number of Top of Rack (ToR) switches, each one connected to the full mesh (there is one access switch per rack).
- Number of nodes per rack: number of nodes (i.e., servers) per rack.
- Number of cores: number of available processing slots (CPU cores) in each node, this number represents how many simultaneous tasks a node can run.
- Number of links between two layer-1 meshes: how many inter- (layer-1) mesh connections (i.e., interlinks) from switches in one mesh to the switches in another mesh.
- Spine-Leaf network parameters: how many spine switches are in the spine-leaf topology.

B. Experimental Setup

In the experiments, we used the simulator to assemble 3 different cluster configurations, and executed the tests considering the file distribution and number of concurrent jobs for each case. We are considering that some portion of the input data is used more often than the rest. This is the common scenario in big data processing [15], where most of the time, the jobs are reusing the same data in several executions, or reusing output from previous jobs. In the experiments, this is a configurable parameter, and the jobs concentrate in reading from the hot data. For the majority of concurrent jobs in a multi-tenant cluster [15], this is the common situation, and our topology and algorithms are designed to improve the performance and decrease the delay for this type of scenario. All the experiments used the HDFS default values of 128 MB for the block size and 3 for the replication factor. Those are the values usually found in real clusters, and used for general data.

The network connecting the nodes and the racks is a Gigabit Ethernet, with delays of 1 ms per switch, frame size of 1518 bytes and not considering the delays caused by the TCP receiving buffers (usually with 64 KB) due to the dynamic buffering option found in modern Linux operating system kernels.

Three sizes of clusters were used in the tests:

- 500 nodes (25 racks with 20 nodes each)
- 1000 nodes (50 racks with 20 nodes each)
- 2000 nodes (100 racks with 20 nodes each)

For these clusters, the following graph configurations were used in the tests:

- 500 nodes: 5 layer-1 meshes with 5 racks in each.
- 1000 nodes: 5 layer-1 meshes with 10 racks in each.
- 2000 nodes: 5 layer-1 meshes with 20 racks each.

The number of nodes per rack were chosen based on the regular size of a network cabinet and the conventional number of ports found in Ethernet switches.

The maximum number of concurrent jobs was set to 64 for the tests, with values set to 1, 16, 32, 48 and 64 for the tests. The one single running job case was considered, as specific clusters are sometimes used to solve one job at a time, using all the available resources.

In the spine-leaf topology, the number of switches in the spine is set to half of the number of available racks, ensuring enough parallel paths to keep traffic in adequate conditions. In Multi-Layer-Mesh topology, the number of switches is equal to the number of access switches used in spine-leaf topology. There are no core or spine switches, then, the number of required equipments is less than a regular spine-leaf.

All the results are average values from 100 consecutive runs, to avoid bias due to the utilisation of random variables used to define data placement and so on.

The dataset size used in the experiments is calculated to use all the available resources in the cluster, and is equal to the block size multiplied by the total number of available processing slots in all the nodes. In that way, all the cluster processing resources are used in all the tests, with resulting dataset sizes of approximately 0.5 GB (500 nodes), 1 TB (1000 nodes) and 2 TB (2000 nodes) per concurrent job.

C. Results

The results are shown in Figures 5, 6 and 7 for the three cluster configurations. The plots show the comparison between

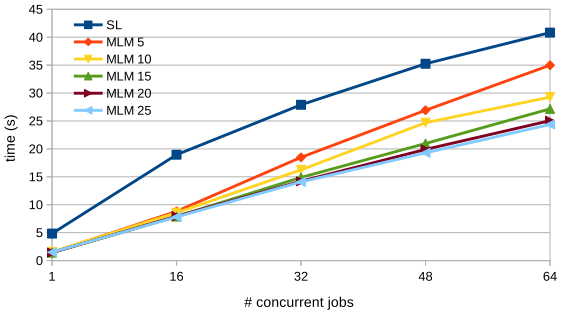

Fig. 5. Jobs completion time (500-node cluster).

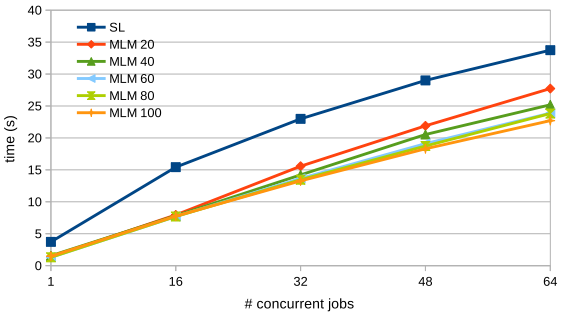

Fig. 6. Jobs completion time (1000-node cluster).

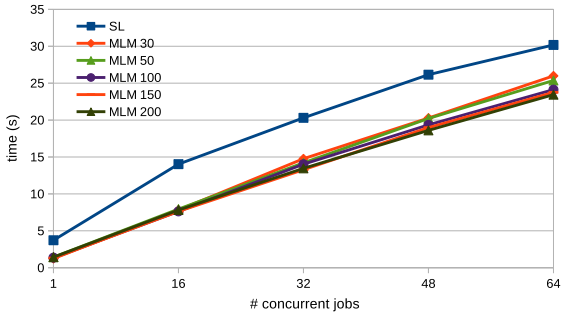

Fig. 7. Jobs completion time (2000-node cluster).

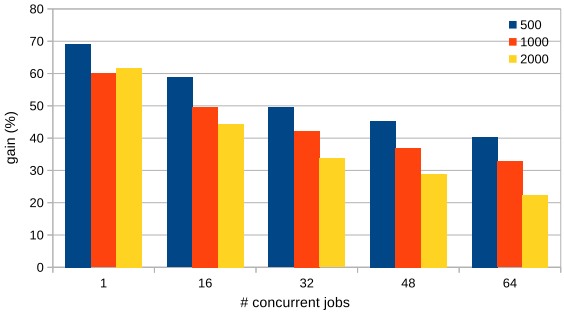

Fig. 8. Performance gain of the MLM topology over the Spine-Leaf topology.

the average reading phase duration of the baseline (Spine-Leaf, represented by SL, in Blue) with the different number of interlinks of Multi-Layer-Mesh topology (represented by MLM, in color). The number of interlinks (links between each pair of layer-1 mesh) is shown in the legends.

The observed gain in performance is related to the number of interlinks, and this is also related to the overall cost to assemble the particular topology. This can be also viewed in Figure 8, that compares the performance gain over Spine-Leaf baseline, using the maximum number of interlinks in all cases.

For the 500-node cluster, this gain is from 14.31% for 5 interlinks, up to 40.29% with 25 interlinks, for 64 concurrent tasks. The maximum gain (with maximum number of interlinks) values are 32.72% for the 1000-node cluster, and 22.32% for the 2000-node cluster. The maximum number of interlinks was kept below the limit imposed by the number of ports per switch. It's worth noting that for just one running task, the performance gain is around 63% for all MLM configurations, showing that our proposed topology can be quite useful for this particular use case.

Figure 9 shows the number of switches and links used. The first two bars show the number of switches in Spine-Leaf and Multi-Layer-Mesh topology for a given number of nodes, and the last two bars show the number of links needed for the same topologies (SL and MLM). The number of links are shown divided by ten, for comparison purposes. We can observe that MLM uses less switches than SL, but more links in the network. As the cost of links are usually cheap (especially copper ones), the overall costs are lower for MLM.

The estimated monetary costs are shown in Figure 10. For this estimate, it was considered a monetary value (in February, 2019) of €2600 per switch [1], and €60 [2] per copper link. The switch model was chosen because it is a conventional one, but still supporting advanced features (like SDN), and the cable link was chosen because CAT5E is still very common in data centers and suits most of the existing architectures.

The chart compares the estimated cost for a SL topology to an equivalent MLM topology that has the same performance, and another with the maximum performance as shown in Figure 8. The average reduction in costs for MLM is 39.02% for the same performance of a SL installation, and 22.91% for maximum performance.

The relationship between gain in performance and reduction in cost is shown in Figure 11, for a 500-node cluster. In this chart, as the gain in performance increases, the reduction in cost decreases. The same trend occurs in the other cluster configurations.

V. Conclusion

As shown in the results section, our novel Multi-Layer-Mesh topology and SDN-based path switching protocol can achieve the same level of performance of an Spine-Leaf topology with an average of 36.03% reduction in network infrastructure costs, or increase performances by an average of 31.77% (from 22.32% up to 40.29%) using the maximum level of

[1] 48-port SDN switch, in https://www.fs.com/products/69226.html

[2] Stranded copper F/UTP CAT5E 100m cable, in https://www.maison-du-cable.com/prix/cable-multibrins-f-utp-cat5e-16291.html

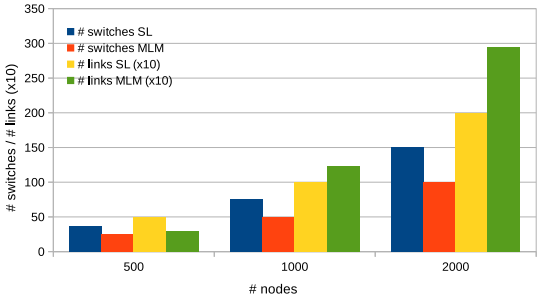

Fig. 9. Number of switches and links *vs* cluster topology and size.

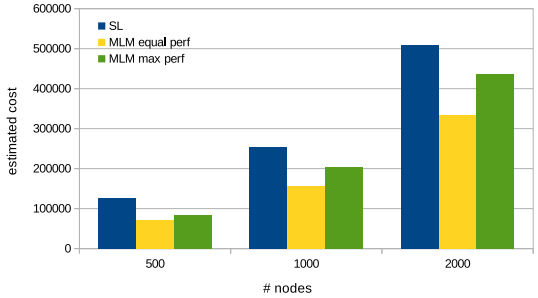

Fig. 10. Estimated cost (€) *vs* cluster topology and size.

interlinks. It should be noted however that the savings in equipment are as high as the performance gains, and although the number of connections is increased compared to a Spine-Leaf topology, connections (especially copper ones) are much cheaper than equipment, thus the overall cost is still reduced. Even more significant is the potential cut in costs, related with the energy consumption from the switches and the related cooling systems. If a system achieves the same level of performance with less equipment used, the energy and cooling for those equipments are not needed anymore. Based on the data from a survey [2], network devices are responsible for 20%–30% of the energy consumption of a data center and for each watt consumed by a network device or a server, one watt is consumed for cooling the same equipments. In short, either from the viewpoint of performance gain, or energy and cost savings, the Multi-Layer-Mesh topology and path switching protocol shows positive preliminary results.

For future work, we plan to integrate the topology with data and task placement strategies, improving the level of information that the network has about running applications, in order to better configure the communications' flows. As this research was conducted assuming that regular switches were used, we will adapt the topology and simulator to test optical switches and connections as well. We also plan to improve the way the path/traffic analysis is executed, allowing the controller to react to changes in the network during the jobs' execution.

Acknowledgments

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

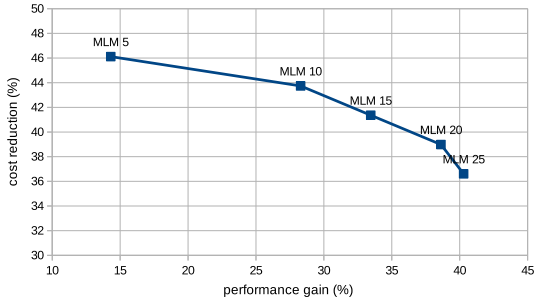

Fig. 11. Cost *vs* performance (500-node cluster with 64 concurrent jobs).

References

[1] F. G. Villanustre, "Big data trends and evolution: A human perspective," in *Proceedings of the 3rd Annual Conference on Research in Information Technology*, 2014, pp. 1–2.

[2] A. Hammadi and L. Mhamdi, "A survey on architectures and energy efficiency in data center networks," *Computer Communications*, vol. 40, pp. 1 – 21, 2014.

[3] T. Chen, X. Gao, and G. Chen, "The features, hardware, and architectures of data center networks: A survey," *Journal of Parallel and Distributed Computing*, vol. 96, pp. 45 – 74, 2016.

[4] Y. Xia, X. S. Sun, S. Dzinamarira, D. Wu, X. S. Huang, and T. S. E. Ng, "A tale of two topologies: Exploring convertible data center network architectures with flat-tree," in *Proceedings of the Conference of the ACM SIG on Data Communications*, 2017, pp. 295–308.

[5] G. Guelbenzu, N. Calabretta, and O. Raz, "Hybrid fat-tree: Extending fat-tree to exploit optical switch transparency with wdm," *Optical Fiber Technology*, 2018.

[6] Z. Zhang, W. hu, W. Sun, L. Zhao, and K. Zhang, "Elastic optical ring with flexible spectrum roadms: An optical switching architecture for future data center networks," *Optical Switching and Networking*, vol. 19, pp. 1 – 9, 2016.

[7] L. Cui, R. Yu, and Q. Yan, "When big data meets software-defined networking: Sdn for big data and big data for sdn," *IEEE Network*, vol. 30, pp. 58–65, January 2016.

[8] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data," *IEEE Systems Journal*, vol. 11, pp. 2337 – 2344, 2017.

[9] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *9th ACM Internet Measurement Conference*, 2009, pp. 202–208.

[10] S. Narayan, S. Bailey, and A. Daga, "Hadoop acceleration in an openflow-based cluster," in *SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 535–538.

[11] E. S. Nejad and M. R. Majma, "A modern method to improve efficiency of hadoop and mapreduce cluster using software-defined networks technology," in *2017 Iranian Conference on Electrical Engineering (ICEE)*, 2017, pp. 1497–1502.

[12] H. Ghalwash and C. H. Huang, "Software-defined extreme scale networks for bigdata applications," in *IEEE High Performance Extreme Computing Conference*, 2017, pp. 1–7.

[13] L. B. de Almeida, E. C. de Almeida, J. Murphy, R. de Grande, and A. Ventresque, "Bigdatanetsim: A simulator for data and process placement in large big data platforms," in *IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications*, 2018.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.

[15] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1802–1813, 2012.