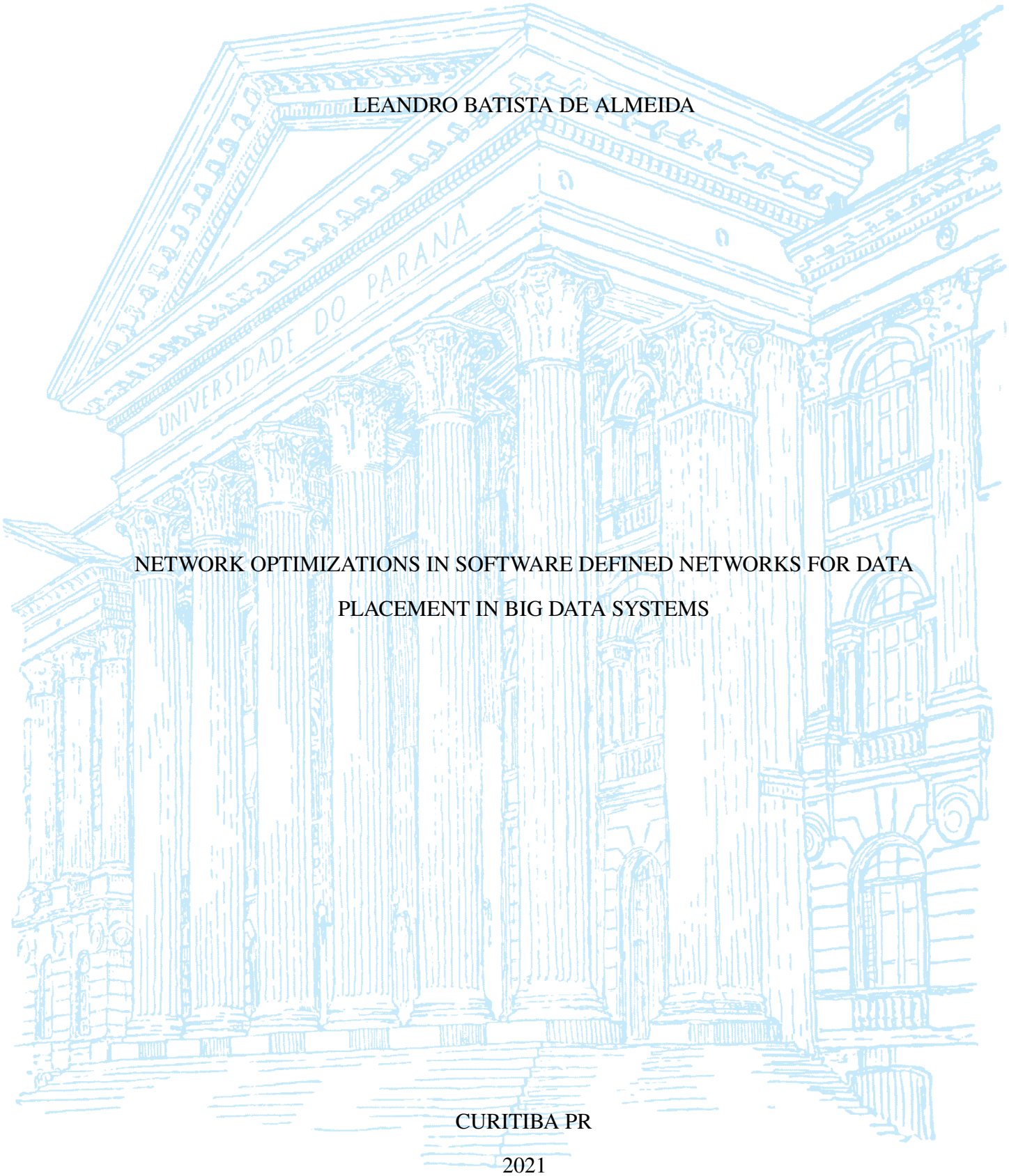
UNIVERSIDADE FEDERAL DO PARANÁ

LEANDRO BATISTA DE ALMEIDA



NETWORK OPTIMIZATIONS IN SOFTWARE DEFINED NETWORKS FOR DATA PLACEMENT IN BIG DATA SYSTEMS

CURITIBA PR

2021

LEANDRO BATISTA DE ALMEIDA

NETWORK OPTIMIZATIONS IN SOFTWARE DEFINED NETWORKS FOR DATA PLACEMENT IN BIG DATA SYSTEMS

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Cunha de Almeida.

CURITIBA PR

2021

CATALOGAÇÃO NA FONTE – SIBI/UFPR

A447n Almeida, Leandro Batista de

Network optimizations in software defined networks for data placement in big data systems [recurso eletrônico]/ Leandro Batista de Almeida - Curitiba, 2021.

Tese apresentada no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná. Área de concentração: Ciência da Computação.

Orientador: Eduardo Cunha de Almeida.

1. Big data 2. Redes de computadores. 3. Informática. I. Almeida, Eduardo Cunha de. II. Universidade Federal do Paraná. III. Título.

CDD 004.66

Bibliotecária: Vilma Machado CRB9/1563

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **LEANDRO BATISTA DE ALMEIDA** intitulada: **Network Optimizations in Software Defined Networks for Data Placement in Big Data Systems**, sob orientação do Prof. Dr. EDUARDO CUNHA DE ALMEIDA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 22 de Fevereiro de 2021.

Assinatura Eletrônica
22/02/2021 15:16:24.0
EDUARDO CUNHA DE ALMEIDA
Presidente da Banca Examinadora

Assinatura Eletrônica
22/02/2021 18:06:03.0
ANTHONY VENTRESQUE
Avaliador Externo (UNIVERSITY COLLEGE DUBLIN)

Assinatura Eletrônica
22/02/2021 17:35:15.0
LUIZ CARLOS PESSOA ALBINI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
22/02/2021 14:40:32.0
DAMIEN MAGONI
Avaliador Externo (UNIVERSITÉ DE BORDEAUX)

To Maria Leopoldina and Estanislau,
teachers for a lifetime.
To Andrea and Pedro,
my new teachers.

ACKNOWLEDGEMENTS

During a period of intense work as a PhD, it is very important to surround yourself with people who can help you, support you and allow you to do your best. I am very grateful to have had this opportunity and to be surrounded by a great group of people who allowed me to make this effort.

I am deeply grateful to my advisor, Professor Eduardo Cunha de Almeida, for the trust placed in me, for the guidance and patience over the years, for the brilliant outbursts in the course of the work, and for the companionship.

I also want to express my deepest gratitude to thesis committee members for the suggestions and reflections shared in a welcoming and rigorous environment, on behalf of Professors Luiz Carlos Pessoa Albini, Anthony Ventresque and Damien Magoni, to whom I register my admiration for their affable and committed posture in the academic environment.

I would also like to thank all colleagues of the Database Lab at the Federal University of Paraná, and all the colleagues of the Computer Science Department in UCD.

I am deeply grateful for the support and encouragement of my friends, both in Brazil and those I was privileged to make in Ireland.

And as with almost any aspect of life, the importance of the family is enormous. Without the support and love of my family, this thesis would not have been possible.

RESUMO

Sistemas de Big Data estão disponíveis por mais de uma década, com Hadoop/YARN se tornando a plataforma padrão "de facto"para esses sistemas, sendo usados em áreas como análise de dados, aprendizado de máquinas e IoT.

Esses sistemas são baseados principalmente em dados distribuídos em clusters de máquinas, que dependem fortemente da infraestrutura de rede para mover dados entre centenas ou milhares de nós. Considerando a movimentação de dados em um cluster de máquinas, existem cenários e condições em que o desempenho geral pode ser reduzido, devido à contenção de acesso a dados e gargalos de rede. A disposição de blocos no sistema de arquivos distribuído (Hadoop Distributed File Sytem, HDFS) que suporta a maioria dos sistemas de BigData, também tem um impacto significativo no desempenho dos sistemas e da rede.

A questão de pesquisa principal a ser respondida por esta tese é: *podemos melhorar o desempenho de um sistema de Big Data modificando como os dados serão movidos e posicionados no cluster de acordo com a utilização desses mesmos dados pelo sistema?*

Nossa hipótese é que isso pode ser realizado por um esforço combinado, usando uma estratégia que permite que a rede entregue de forma mais eficiente os blocos de dados aos nós que precisam deles (com uma nova topologia de rede baseada em SDN - Software Defined Networks) e também fazendo com que os dados estejam disponíveis em nós que minimizem os custos de acesso (com um algoritmo de disposição de dados).

Para fazer isso, propomos uma implementação de uma nova estratégia de distribuição de dados sobre uma nova topologia de rede chamada Multi-Layer-Mesh e um algoritmo de comutação de caminhos que utiliza Software Defined Networks. O Multi-Layer-Mesh permite selecionar caminhos de rede, orientando a disposição de dados distribuídos de arquivos mais utilizados e indexando máquinas, que, quando executados em conjunto, podem reduzir a movimentação de dados que pode ter um impacto importante no desempenho de sistemas de Big Data.

O algoritmo de disposição de dados proposto difere do algoritmo HDFS clássico por direcionar blocos dos arquivos mais usados (dados quentes) para os nós com mais slots de processamento disponíveis, aumentando assim a chance de utilizar dados locais (localidade dos dados) e diminuindo o uso da rede.

A topologia de rede Multi-Layer-Mesh e o algoritmo de distribuição de dados são as principais contribuições desta tese, assim como um simulador de sistemas de Big Data, o BigDataNetSim, desenvolvido para simular a movimentação de dados em um cluster HDFS.

Uma avaliação extensiva baseada em simulação de nossos algoritmos mostra uma melhora média no desempenho de 42% e uma diminuição média na utilização de recursos de 36,03% em comparação com uma topologia Spine-Leaf tradicional e o algoritmo de distribuição de dados HDFS clássico, em cenários selecionados de teste.

Palavras-chave: Big Data. SDN. topologias de rede. disposição de dados

ABSTRACT

Big Data systems have been around for more than a decade now, with Hadoop/YARN becoming the “de facto” standard platform for these systems, being used in areas like data analysis, machine learning and IoT.

These systems are mainly based on data distributed in a cluster of machines, that rely heavily on the network infrastructure to move data around hundreds or thousands of distributed nodes. Considering the movement of data in the distributed nodes, there are scenarios and conditions when the overall performance can be reduced, due to data access contention and network bottlenecks. The placement of blocks in the distributed file system (Hadoop Distributed File System, HDFS) that supports most platforms, also has an important impact on the performance of the system and the network.

The main research question to be answered by this thesis is: *can we improve the performance of Big Data systems by modifying how the data will be moved and placed in the cluster according to the usage of data by the system?*

Our hypothesis is that this can be accomplished by a combined effort, using a strategy that allows the network to more efficiently deliver the data blocks to the nodes that need them (with a novel network topology based on SDN - Software Defined Networks) and also making the data available in nodes that minimize the access costs (with a data placement algorithm).

In order to do that, we propose an implementation of a novel data distribution strategy over a novel network topology called Multi-Layer-Mesh and a path switching algorithm leveraging Software-Defined Networking. The Multi-Layer-Mesh allows selecting network paths, guiding distributed data placement of hot files and indexing machines that, when running altogether, can reduce data movement which could have an important impact on performance of big data systems. The proposed data placement algorithm differs from the classical HDFS algorithm by directing blocks of the most used files (hot data) to the nodes having more available processing slots, therefore increasing the data locality and decreasing network usage.

The Multi-Layer-Mesh network topology and the data distribution algorithm are the main contributions of this thesis, as well as the Big Data system simulator, called BigDataNetSim, developed to simulate the data movement in a HDFS cluster.

A thorough simulation-based evaluation of our algorithms shows an average improvement in performance of 42% and an average decrease in resource utilization of 36.03% compared to a traditional Spine-Leaf topology and the classical HDFS data distribution algorithm, in the selected test scenarios.

Keywords: Big Data. SDN. network topologies. data placement

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
UTFPR	Universidade Tecnologica Federal do Paraná
HDFS	Hadoop Distributed File System
RMON	Remote Monitoring Protocol
SDN	Software Defined Networks
SNMP	Simple Network Management Protocol
TOR	Top Of Rack switch

CONTENTS

1 INTRODUCTION

1.1 BIG DATA SCENARIO

In the last decades, we have observed an important growth in the volume of data generated by information systems. The amount of collected data is reaching unprecedented levels in history that is expected to be more than 5× the available storage by the year of 2025 (Milo, 2020). Considering data from the Internet, Data Warehousing and Business Intelligence systems, IoT and Machine Learning data flows, the storage and processing needs for large volumes of data (hence, Big Data) has consistently grown (Milo, 2020).

In order to meet that demand, technologies and tools were developed and adopted both in industry and academia. Since the publication of the seminal papers on the Google File System (Ghemawat et al., 2003) and MapReduce (Dean and Ghemawat, 2008) by Google, and the start of Hadoop project in 2006, we observe a huge growth in the Big Data market, with several other tools, frameworks and methodologies being developed to help solve the problems at hand (Villanustre, 2014).

Considering the volume of data, and the impossibility of storing it in a single computer - as big as it can be - the Big Data systems usually rely on some form of distributed file system. From the beginning, the Hadoop Distributed File System (HDFS) (Shvachko et al., 2010) is used in many of these systems to properly store and retrieve the data in an organised and efficient way. Due to its features, HDFS is generally used not only by Hadoop, but also by other systems, such as Spark, Impala, TEZ, Flink and Presto. This leads to the scenario where HDFS becomes the basis of the Big Data solution stack, with almost all the software solutions relying on HDFS to store both the input data and the results of the jobs. This architecture is shown in Figure 1.1, presenting the usual Hadoop stack, 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.

1.2 PERFORMANCE IN BIG DATA SYSTEMS

This foundation role performed by HDFS makes it an optimal target for performance improvements, as any improvement in performance here, can impact the entire software stack depending upon it.

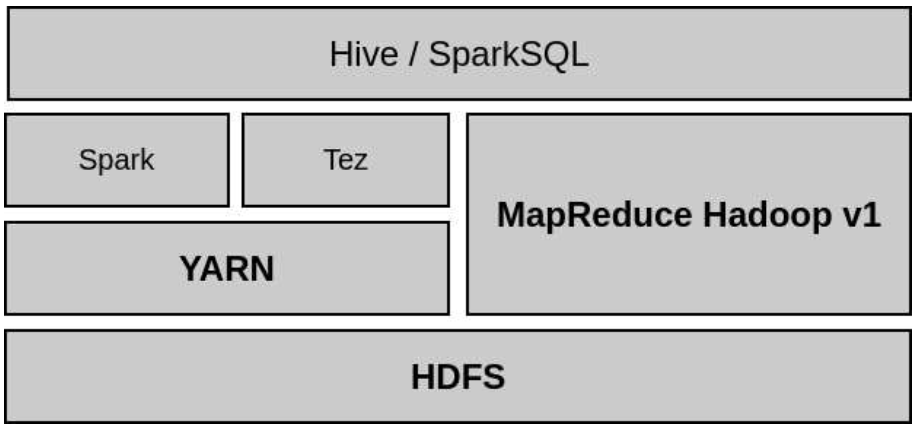


Figure 1.1: Common Hadoop stack.

Thus, 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 and data transfer across distributed machine nodes[1] required for running the jobs (Hammadi and Mhamdi, 2014; Filho et al., 2019).

As a distributed system, Big Data systems have several components that can affect the performance of job execution. In this sense, two variables can be of importance to increase performance, namely the data distribution strategy and the underlying network infrastructure. The distributed nature of Big Data systems based on HDFS makes it sensitive to where the data will be processed and how and when the data will be transferred around the nodes of a cluster.

Hadoop (and other Big Data systems) tries to process the jobs locally, by reading data from the local node storage, when possible. However, if there is no available processing slot in the nodes where the data is located, data will be transferred over the network from any other cluster node that contains the appropriate block of data.

The data distribution strategy can help to increase the number of data blocks processed locally, and, in the cases when the data must be transferred from another node, the network infrastructure can avoid bottlenecks and increase the overall throughput in the system.

Moreover, we have a trend in Big Data clusters showing several concurrent analysis platforms running in the same cluster (Chen et al., 2012), like simultaneous Hive queries, Spark jobs and MapReduce jobs. The current clusters are probably multi-tenant ones, where several different software layers must co-exist, due to the costs involved in assembling and managing a large cluster. This translates into a greater burden for the network infrastructure, which is already a critical component in distributed systems.

To address the network performance issue, we have developed a novel network architecture, based on a Multi-Layer-Mesh (MLM) topology, and an SDN-driven forwarding algorithm that can use information about the job execution from the Hadoop main server to organise the network traffic for optimizing the use of available network paths (de Almeida et al., 2019). Our proposal can decrease the number of network equipment used, and increase the network performance for Big Data job execution, reducing costs both in equipment acquisition and maintenance, and in energy consumption and cooling.

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.

In the course of this research, we also have designed a novel data placement algorithm for an HDFS-based Hadoop system, which is using the Multi-Layer-Mesh topology (de Almeida et al., 2019) as a network transportation system. Our goal is to investigate how the combination of our novel data placement algorithm with the Multi-Layer-Mesh topology can improve the overall performance of the system.

In order to execute the experiments of this research, we have implemented BigDataNetSim to simulate the data movement in hundreds or thousands of nodes. This was done as the available simulators did not meet the requirements needed to reproduce thousands of nodes moving data around, like the collecting of network statistics and using different data placement strategies. The accuracy and performance of the simulator was assessed and the results were published in (De Almeida et al., 2018). We have carried out our experiments by using our BigDataNetSim simulator, and have tested several scenarios, designed to reproduce common situations in Big Data systems. Our algorithms show an average improvement in performance of 42% and an average decrease in resource utilization of 36.03% compared to a traditional Spine-Leaf topology

[1]We refer to a "node" as a computing machine that can be connected to a set of machines to form a cluster.

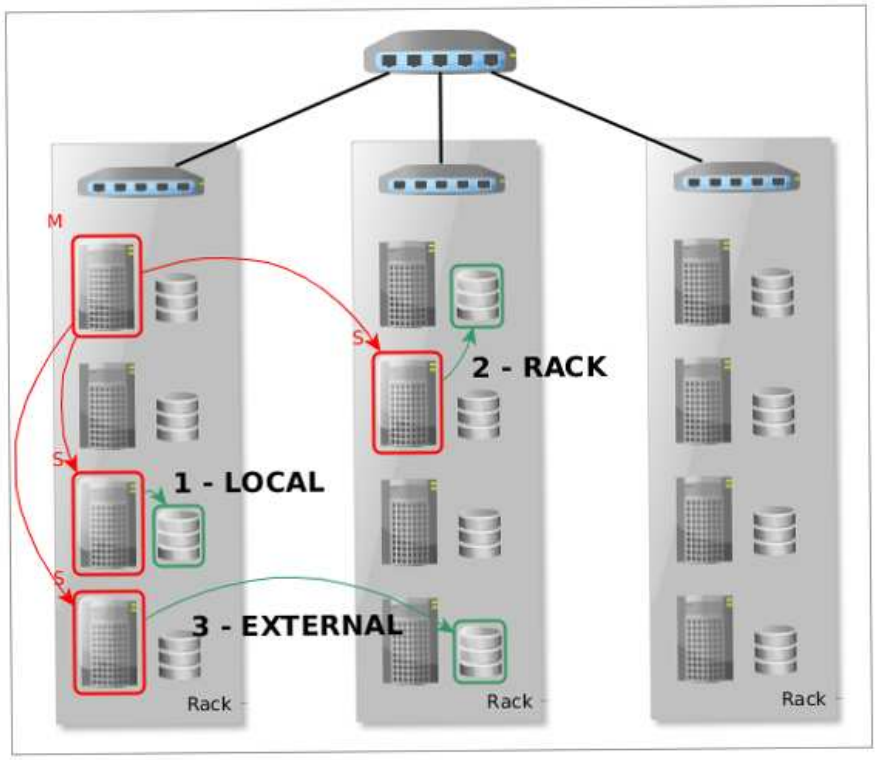


Figure 1.2: Example of local (1), same rack (2) and external (3) tasks.

and the classical HDFS data distribution algorithm, in the selected test scenarios, with an initial pre-computation yielding a negligible overhead on CPU and memory usage.

1.3 CONTEXT AND PROBLEM STATEMENT

Nowadays, the general scenario for Big Data systems 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 (Villanustre, 2014). 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 (Shvachko et al., 2010), 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. 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 required block of information about to be read is located, avoiding delays 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 5.1, 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 application framework used by the Big Data system (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.

In Hadoop v3, a new file system organisation was made available, using Erasure Coding based in Reed-Solomon form (Plank et al., 2009) for some particular scenarios in HDFS. When the files are not often accessed (warm and cold datasets), with relatively low I/O activities, in not likely that the replicas will be frequently accessed. In these cases, the use of Erasure Coding will reduce the storage usage by 50% when compared with the usual replication strategy. However, EC will increase the CPU and network usages, not being suitable for high traffic situations (HDF, c).

To sum it up, the problem to be solved is how to improve or at least maintain performance while decreasing the amount of resources needed to run a big data cluster under high load with concurrent access to hot data.

1.4 IMPROVING THE PERFORMANCE OF BIG DATA SYSTEMS

In recent years, some new technologies became available to help in the performance issues of big data systems.

First, the concept of Software Defined Networks (SDN) brings the agility of virtual machines to the network infrastructure, allowing the management of communication links, traffic and routing rules and security in a software level, independent from the hardware manufacturer. In this new architecture, instead of complex routing/switching equipment distributed in the network, we can use simple and affordable packet passing equipment that reports to a central server in order to process and get the rules dictating the network behaviour. This brings the opportunity to observe the network traffic and the applications running on top of it, and make decisions based on the current demands, in effect, making the network easier to react to application needs and traffic changes.

For big data systems, this means collecting information about the running frameworks and datasets used, and configuring the network accordingly on the fly. There are several research works in this sense, as we will see in the Literature Review chapter, and we will propose a novel strategy for this integration.

There are several network topologies designed for data center and cluster environments (Chen et al., 2016), and some of these topologies are adequate for clusters, like Fat Tree or Spine-Leaf, however, those topologies do not 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.

If the number of core switches could be reduced, this would lower the cost of acquisition and maintenance, considering that those switches are probably of high-end class, and decrease the associated costs in energy consumption and cooling systems.

In the same sense, trying to place the blocks of data in a way that the HDFS algorithms can better use the network resources will also increase the performance and possibly decrease the usage of network resources.

In order to improve the performance of these Big Data systems, this research is exploring novel network topologies and routing algorithms and data placement to bring together the application level data and the network organisation techniques.

The main research question to be answered by this thesis is: *can we improve the performance of Big Data systems by modifying how the data will be moved and placed in the cluster according to the usage of data by the system?*

Our hypothesis is that this can be accomplished by a combined effort, using a strategy that allows the network to more efficiently deliver the data blocks to the nodes that need them (with a novel network topology based on SDN - Software Defined Networks) and also making the data available in nodes that minimizes the access costs (with a data placement algorithm).

1.5 DOCUMENT ORGANISATION

This document is organised as following:

Chapter 2 shows the Literature Review, covering aspects about SDN and optimizations based on it, Data placement in HDFS clusters, data center topologies and related work on Big Data simulators.

In Chapter 3, the BigDataNetSim is shown, describing its requirements and why we needed our own simulator for the research, as well its architecture, internal operation and the executed evaluation tests.

Chapter 4 addresses optimisations in Big Data systems, particularly the concept of "data locality", explored with experiments on the replication factor of HDFS, and a novel data placement algorithm, that considers the data locality and processing slots in the cluster. The results of the corresponding experiments are also shown.

The main contributions of this research are shown in Chapter 5, namely the Multi-Layer-Mesh network topology, designed to improve the performance of large scale Big Data clusters, and the implementation of our novel data placement algorithm using MLM as the network topology. The conducted experiments are described in the chapter, as well its results.

Chapter 6 presents the conclusion and future work based on this research.

2 LITERATURE REVIEW

The literature review was based on the concepts needed for the research, and the previous works in the same area, that show improvement possibilities.

2.1 DATA CENTRE TOPOLOGIES AND HADOOP CLUSTERS

A Data Centre (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) (Hammadi and Mhamdi, 2014; Chen et al., 2016). 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 was proposed in (Xia et al., 2017) to overcome this issue. It can dynamically change the network topology in order to set up architectures optimised 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) (Guelbenzu et al., 2018). 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 (Zhang et al., 2016b).

Software-Defined Networking (SDN) is increasingly being deployed in modern DCs and can benefit big data applications in many ways (Cui et al., 2016), 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 (Qin et al., 2017a). By measuring and analyzing the DC traffic, such as presented in (Kandula et al., 2009), 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 (Narayan et al., 2012a). 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 (Nejad and Majma, 2017). Ghalwash et al. investigate the throughput and execution time of Hadoop read/write and sorting operations (Ghalwash and Huang, 2017b). 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.

Regarding the network topology and routing algorithms (described in Section 5.1), our research is different from these previous 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.

2.2 BLOCK PLACEMENT ALGORITHMS FOR HDFS

As mentioned in Section 1,3, the common ground for several Big Data systems is the use of the distributed file system, Hadoop Distributed File System (HDFS), to store the source data and the results for the jobs. Despite several of these technologies could potentially use other processing engines apart from Hadoop YARN (like Hive with TEZ and Spark), all of them read the data from nodes in a HDFS cluster.

HDFS provides a way of distributed processing access the distributed data in a local fashion, i.e., the code is transferred to a node where the data resides, in order to reduce the network traffic. As the pieces of code are significant smaller than the pieces of data (a jar file have an common size of dozens of KB, and a block of data is at least 128 MB), it is always more efficient to transfer the code and not the data, due to the respective sizes (Santos et al., 2018). This leads to the concept of "data locality", explained in Section 3.1.

Observing how the HDFS works, particularly when we have several concurrent jobs using the same part of the data (so called “hot data”), we can observe that the number of tasks using the network connections increases as the number of concurrent jobs also increases. This is due to the fact that when the available processing slots (virtual cores) are depleted for a particular node, the standard Hadoop algorithm selects a different node to run the task, and the data needs to be transferred through the network, as stated in Section 1.3.

This behavior leads to the fact that where the blocks of data are placed could have an impact on how fast these blocks will be processed by the different nodes where the tasks will be allocated. The HDFS standard data placement algorithm considers only the fault tolerance aspects when deciding where to store the blocks of a particular file, treating the distribution in an agnostic manner. Considering this, and the potential impact of the data localization, there are efforts to improve the performance of HDFS using different data placement algorithms.

In this research, our requirements are focused on the data movement in the cluster and how tuning network features in order to improve the performance. The processing slots in each node must also be considered, as this will help to determine when the task will run locally (reading data from the local storage) and when the data will be transferred through the network. Then, a data placement used in the research must consider these factors in order to be helpful in this context. For our literature review, several research works were studied, as follows.

In "A Novel Blocks Placement Strategy for Hadoop", the authors propose a load balancing oriented block placement, in order to avoid disk space waste and improve the overall performance of Hadoop (Ye et al., 2012). Although the performance is a goal in this data placement strategy, it is mainly oriented towards disk usage, trying to balance the load among the nodes in the cluster. The processing slots and the task behavior are not considered here, neither any other network aspect, such as the rack disposition or network topology, although there is some information about this provided by Hadoop itself. As we need to interact with the network infrastructure in our research, this data placement strategy does not fit our requirements.

In "A Load-Balancing Algorithm for Hadoop Distributed File System", the authors propose a system to re-balance the blocks stored in an HDFS cluster using an external node as the balance manager (Lin and Lin, 2015). The main idea is to execute the re-balancing from time to time to keep the storage usage even in the cluster. The intention is similar to the previous paper mentioned in this list, and again, the focus is only related to the storage usage, although the performance can be improved as a side effect, due to a better distribution. This architecture also needs an additional node to control the data flow, besides the fact that the re-balancing process is potentially slow. Similarly to the previous paper, there is no concern about the processing slots in the nodes, so it is not applicable to our research.

In "Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster", the authors describe a strategy to better place the data for video processing near the node about to consume that information (Zhang et al., 2016a). This data placement is heavily focused in a particular use case (video processing), making its utilisation or adaptation difficult. As the video processing must be executed in a sequential order, the files need to be aggregated to avoid the overhead in the network. Although this work shows how flexible the HDFS can be (being adapted for very specific cases), it is not a general use of the average user in a Hadoop environment. There is no interaction with the network layer, also.

In "RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems" (He et al., 2011) the authors study different data organization (horizontal row-store, vertical column-store, hybrid PAX store) as a data placement strategy, and propose a hybrid and compressed one (RCFile) to store data. However, this method requires a different file organization, not applicable for other applications or file organization systems. Therefore, the solution is not generally applicable, and it is not useful for our purposes.

In "Intentional Data Placement Optimization for Distributed Data Warehouses" (Arres et al., 2015) the authors propose a customized data warehouse placement policy, based on an unsupervised machine learning cluster algorithm (k-means). However, the policy uses relational data characteristics into account, not being applicable to other types of data organization. Although very useful to data warehouse systems (in a particular organisation), it is not applicable to general solutions on the Hadoop stack.

In a general way, there is an interest in improving the HDFS standard data placement algorithm, mainly because the generic nature of HDFS allows for this type of improvement proposal. As a distributed file system that aims to treat any data in the same way, HDFS compromises part of the potential performance in order to accomplish that.

In the same way, several of the data placement algorithms reviewed also compromise part of HDFS generic nature to achieve performance improvements, being specific to some use cases.

Moreover, none of the previous works analyse the network behavior regarding different network features, like physical topology or link layer protocol, a desired feature for this research. Also, the node capacity (processing slots) is not considered, and as this is an important feature in our scenario - especially when combined with the network features - we developed our own data placement algorithm to better address these features.

2.3 SOFTWARE DEFINED NETWORKS

With the increasing demand in network traffic originated from Big Data, IoT and Machine Learning systems, the complexity in network infrastructure increased in the same level, creating problems in the network management and in the applications running on top of those networks. Several approaches were tried to solve these problems (like MPLS and Provider Bridging Network), but when the seminal papers from Martin Casado and Nick McKeown (Casado et al., 2007) (McKeown et al., 2008) described the concept of Software-Defined Networks (SDN) and the OpenFlow protocol, industry and research areas shown an increased level of interest. In our current situation, all the main commercial providers offer solutions based on SDN (Casado et al., 2014) and there is a thrilling research area (Kreutz et al., 2015).

SDN is based on two simple ideas: decouple the software that controls the network from the devices that implement it and make the network hardware (switches, routers) general, providing a standardised set of packet-processing functions, in opposition to proprietary vendor features (Casado et al., 2014). The separation of the controlling software is based on an external

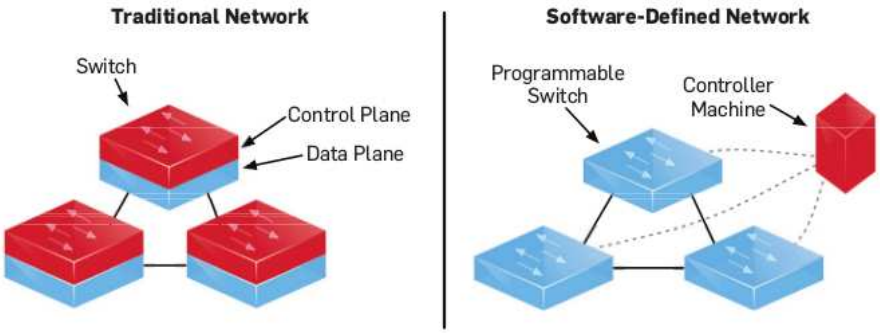


Figure 2.1: Comparison of traditional and a SDN based networks (Casado et al., 2014).

controller that receives information from the network hardware, executes the switching and routing functions, and sends to the network equipment the directions about how to proceed with the passing packets. In this structure, one or more controller nodes (general purpose computers), acting as the controllers, can run the programs responsible for the network organisation, and communicate with other programs running in the network, in order to better determine the changes in the network. This architecture creates the two "planes" of SDN, the Data Plane, where the actual packets are processed and transmitted, and the Control Plane, where the messages concerning the network control are transmitted.

With this architecture, is is possible to change and evolve the network infrastructure without having to change the underlying network hardware (and without being linked to a particular vendor) and enables expressing network algorithms in terms of appropriate abstractions for particular applications (Casado et al., 2014) (Casado et al., 2012).

Several major vendors already provided software control for their network hardware, but usually in a proprietary way, or using open protocols in a limited fashion. In SDN, instead of having a complex and powerful switching equipment in every network segment, a manager can deploy simpler hardware (and cheaper) that communicates to the controller to receive the instructions about how to proceed. This architectural contrast is shown in Figure 2.1, showing the SDN controller executing a general-purpose program that responds to the network events.

According with Martin Casado (Casado et al., 2014), the most important points in SDN are:

- The software that controls the network devices runs in a controller, that is a general-purpose computer, running general-purpose programs (written in Java or Python language, for instance);
- As all the network devices send to the controller its status, there is a global visibility of the network state, simplifying how the network algorithms runs;
- In the majority of the cases, the network functionalities can evolve without changing the underlying hardware.

A protocol specification called OpenFlow (McKeown et al., 2008) was introduced to define how the network devices communicate with the controller (a standard interface) and a standard collection of features switches must provide. Using the implementations of this specification, a programmer can create a software to run in a controller, acquiring network information and sending orders or configurations for the switches.

Since the beginning of SDN implementation by the major vendors, new resources and APIs have become available for the network and application managers. From that, several tools,

algorithms and models were implemented, and several application areas started to experiment integration techniques with the network.

2.4 OPTIMISING BIG DATA FRAMEWORKS USING SOFTWARE DEFINED NETWORKS

There are several research projects in the literature that use some SDN capabilities to improve Big Data systems, but we select the projects closest to our goals. It is interesting to mention that none of the presented works are similar to our proposal.

In "An SDN App for Hadoop Clusters" (Lin and Liao, 2015), the authors use a program in Node.JS to include rules in the SDN switches, in order to increase the performance of the shuffle phase of MapReduce jobs. This can improve the performance of Big Data systems in general, particularly in cloud systems, the paper goal. The authors developed an Android application, designed to be used by the system administrators, to monitor and manage the flow rules in the SDN control plane. It is an interesting work to show that SDN can improve the performance of a Big Data system, even if these rules are focused only in one phase of the processing, and even being just queue precedence rules. However, the paper only focuses on a particular phase of a particular framework (MapReduce), limiting the usage in other scenarios.

In "Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data" (Qin et al., 2017b), the authors describe an heuristic bandwidth-aware task scheduler (BASS) using SDN. The strategy is based on using SDN to read the available bandwidth in every network link in the cluster, while also collecting information about the running Hadoop tasks. Based on that, BASS can estimate the execution time for the tasks, classifying by type, (map, reduce, sort, shuffle, etc). With this data, for every new task, BASS can decide if it is better to run the task with local data (and eventually wait for a processing slot being available), or start the task in a remote node, transferring the data from another node in the cluster. The paper presents a mathematical formalisation of the problem, and the results show a noticeable improvement in performance. While the strategy is interesting, the paper only uses SDN to collect the network activity data, a task that can be performed by other (simpler) protocols, like SNMP and RMON. SDN is not used in the proper active role in this tool.

In "Hadoop Acceleration in an OpenFlow-based Cluster" (Narayan et al., 2012b), the authors study how OpenFlow switches can be used to provide better link bandwidth for shuffle traffic. The basic idea is a simple and effective one, give priority for traffic from Hadoop daemons, hoping that these packets get more bandwidth and the task performance increases. They create two queues in the switches, the first queue providing higher priority to all Hadoop traffic, and the second one, giving shuffle traffic even more priority. They conducted sets of experiments with variations of these two basic queues, showing performance increase in all scenarios, going up to more than 20% in the best case. The study is relevant for showing that only with basic precedence queue, Hadoop performance can be increased. However, the same results can be obtained using other strategies, simpler than SDN, specially because the queues are static, do not demand dynamic setup, one of the most important features in SDN. Moreover, the study only focuses on MapReduce, and in a particular phase of the framework.

Almost the same can be said about "Software-Defined Extreme Scale Networks for Bigdata Applications", (Ghalwash and Huang, 2017a), when the authors investigate the use of SDN switches with rules to improve MapReduce phases (as read/write and sorting), measuring throughput and execution time. This study compares normal switches and ODL (OpenDayLight) switches for Hadoop applications. Again, the implemented rules are only the basic ones, and static. The study, as the previous one, shows performance improvement, even using only static rules and focusing only in a particular framework.

As some of the previous works, "Advanced Control Distributed Processing Architecture (ACDPA) using SDN and Hadoop for Identifying the Flow Characteristics and Setting the Quality of Service(QoS) in the Network" (Desai and S, 2015) studies the data capture in SDN. The paper shows how an OpenDayLight controller can obtain data from the network traffic, and analyse it, in order to propose QoS rules based on the classification of this traffic. The work is important to reinforce the interest in capturing network information through SDN switches, and to show that it is possible to benefit from the fact that a SDN switch is always collecting data from the passing packets, therefore, taking a next step to process this data is not a difficult task. This data processing can be done in the controller (or controllers) or in a computer node near the switch. Apart from that, this study focuses on analysing the network traffic and proposing the implementation of rules.

Some works show promising studies using SDN, Hadoop and Machine Learning techniques, which may point to the design of integrated solutions in the future. The survey "Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview" (Latah and Toker, 2018) provides more details of current efforts and recent advancements to include AI in SDN-based networks.

In "An Approach To Efficient Network Design And Characterization Using SDN and Hadoop" (Desai et al., 2016), the authors are concerned with the security aspects of a Hadoop cluster, and the capacity planning for the resources in the cluster. The SDN features are used to obtain data from the running network, in order to feed machine learning algorithms to classify the traffic and the flows. The authors use an artificial neural network to initially classify the packets, and later some variations from K-means, a cluster algorithm, to put each flow in the appropriate class, in order to plan the resources needed for a certain scenario. It is an interesting work due to the data collecting process, and the use of machine learning algorithms to find the desired information, although the paper does not have the same goals as our research. Another interesting feature, is that the machine learning algorithms are used during the processing, showing that, at least for this particular scenario, the algorithm performance is adequate.

In "Application-Awareness in SDN", (Qazi et al., 2013), the authors used a mixed set of techniques to classify network traffic and apply rules to improve the performance of a particular application. The goal is to identify a particular network application based on the passing packets, in order to provide more resources to that application. The application is a VoIP one, running in Android smartphones. This is hampered by the fact that the communication is encrypted, and the solution needs to improve just one application, not a protocol as a whole. The classification is executed by a Machine Learning algorithm, based on information obtained from crowdsourcing, when part of the users are running an agent on their devices, warning the controller when they start to use the application on focus. Based on the first N packages in the communication channel, the Machine Learning algorithm is able to determine if the data flow is being generated from the specified application, even when the packets contents are encrypted. This study is very interesting for using novel techniques to help SDN to find the proper rules to function, although the goals and scenarios are not exactly the same as in our research.

And as we move to optical architectures in the data centre, with the consequent influence on Hadoop, the study "Optical Switch in the Middle (OSM) Architecture for DCNs with Hadoop Adaptations" (Wang et al., 2017) become relevant, for showing part of this new environment. In this paper, the authors propose modifications for Hadoop to improve heavy shuffle applications, by transferring the results from Mapper tasks to storage devices connected to the TOR (Top Of Rack) switches, and then using the optical links to transfer to the Reducer tasks. The study is interesting for showing how the optical links can improve the performance of a MapReduce

application, while using SDN to implement queues for reservation. However, as only MapReduce is considered, the study is limited to applications developed with this particular framework.

2.5 RELATED WORK ON BIGDATA SIMULATORS

We have investigated several previous works that could fulfil the need of a simulator that can provide an environment for precisely representing large-scale Big Data scenarios. The main research questions, in place, were related to how data and task placement algorithms, combined with network topologies and protocols, affect the data movement; thus, these features guided our search.

Many of the simulators we find in the literature focus on prototyping Big Data jobs, such as MRSim (Hammoud et al., 2010), SimMR (Verma et al., 2011), MRSG (Kolberg et al., 2013), and Mumak (Mumak, 2008). This category of simulators is interesting in itself as designing and implementing Big Data jobs is not an easy task; however, they do not address the critical "distributed" nature of these systems.

Moreover, when the deployment of jobs is considered in their design, the Big Data platforms' simulators often focus on one single element of the large scale distributed systems. This aspect is either (i) job placement/scheduling, such as BS-YARN (Lin et al., 2016), SimMapReduce (Teng et al., 2011)), and YARN Scheduler Load Simulator (Apache, 2013)); (ii) VM provisioning, such as MR-CloudSim (Jung and Kim, 2012) and HSim (Liu et al., 2013)); or (iii) optimisation of deployed systems – through a model based and/or performance oriented approach, such as Starfish (Herodotou et al., 2011), Doopnet (Qiao et al., 2016), DAGSim (Ardagna et al., 2018), and Camp (Panda et al., 2018)).

Some of the projects, like Mumak and WaxElephant (Ren et al., 2012), only considered Hadoop v1, based solely on MapReduce engine, without considering YARN as a processing engine. As the current big data frameworks use YARN over Hadoop as a basic infrastructure, these later simulation tools are not applicable since we would not be able to test new frameworks.

Doopnet and Hadoop MiniCluster (Clusters, 2016) were considered for executing code from Hadoop and starting the Hadoop process in a target computer. However, these approaches lack scalability, accommodating only a few dozens or a few hundred nodes. Thus, this limitation has become an impediment as our research requires simulation of thousands of nodes.

A few of the projects were just abandoned or not active, without any evolution in several years, making them not useful for newer versions of processing engines and libraries, like Mumak and MRPerf.

In our search for open source projects in which we could build upon our functionalities for the required scope of our simulations, we could not find any simulator general enough. Unfortunately, as each simulator was developed with a specific set of features and objectives, this proved to be a more complex path than developing the required features from scratch.

As discussed, the main requirements for a simulator that satisfies our research are related to how data and task placement algorithms, in a cluster with certain network topologies and protocols, affect the data movement, requiring information about the network traffic in those particular conditions. Other requirements related to the underlying network are also important, as the switching protocol used and the supported cluster size. These requirements are listed in Table 2.1 along with the studied simulators.

As none of the cited simulators meet all the requirements needed, we developed a new simulator - described in Chapter 3 - to generate the data analysis based on the proposed parameters, such as data and task placement and network features.

		Requirements				
		Data placement	Task placement	Large clusters	Network topology	Switching protocol
Big Data Simulators	MRSim	✓				
	SimMR	✓	✓	✓		
	MRSG	✓				
	Mumak	✓	✓	✓		
	BS-YARN	✓	✓	✓		
	SimMapReduce	✓	✓	✓		
	Yarn Sched Load Sim	✓	✓	✓		
	MR-CloudSim			✓		
	Doopnet	✓	✓		✓	✓
	WaxElephant	✓	✓	✓		
	HadoopMiniCluster	✓	✓			
	MRPerf	✓	✓			

Table 2.1: Comparison of Big Data Simulators

2.6 FINAL REMARKS

The related work shows a very promising research area for use of network topologies and SDN in Big Data frameworks. Even when only basic and static rules are applied, performance improvement is perceived, so we can expect better results with more dynamic techniques. It can be improved when combined with data placement algorithms and related techniques.

This clearly shows a gap to be fulfilled, with researchers using different techniques to bring together the flexibility of SDN and the large scale processing of Big Data systems.

3 BIG DATA NETWORK SIMULATOR

In the course of this research, the need for a proper simulator was found, based on our options to evaluate our proposals. When researching performance improvements in Big Data systems, it is only natural to test the proposals in a prepared environment, in order to properly measure the gains, and study the behavior of the system. When testing Big Data clusters, there are some unique situations:

- A real Big Data cluster is composed of several hundreds or thousands of nodes, a hard and expensive setup for testing. We have a 21 nodes Hadoop cluster to test, but this size is insufficient for part of the research.
- The use of virtual machines is limited by the possible number of VMs running in a given host, usually far below some hundreds.
- Some dockerized setups, like Doopnet (Qiao et al., 2016), using Docker and Mininet, are also limited by the host capabilities, and do not reach the appropriate number for testing.

There are also simulators designed specifically for Big Data systems, but as we will see in a future section, not oriented to the needs of our current research.

In this sense, we were inclined to develop a new simulator that complies with the needs of the research, and allows the desired modifications and protocol implementations. This simulator was published in the DS-RT conference (De Almeida et al., 2018), and is being used in the research for almost one year.

Simulating Big Data platforms, such as Apache Hadoop, plays a fundamental role in enabling and evaluating the feasibility of novel Big Data processing approaches. Due to its importance, the simulation of such platforms has driven a lot of attention and interest in the past decade. Big Data systems contain several parameters that have to be tailored to particular infrastructures, data sets, and algorithms; also, big data jobs are, some extent, difficult – read: "not intuitive" – to engineer. The complexity of these systems seems well suited for the use of simulators, which can help developers of Big Data platforms, as well as Big Data engineers and Data scientists, to tune the various parameters, choose the correct algorithms, and do the proper resource allocation.

Distributed systems are inherently complex as they are an intricate collection of interconnected machines that interact at various levels of their hardware, networking, and software stacks. As such, setting up a Big Data platform can often be seen as a work of art, with capital allocators, data engineers, and data scientists working hand-in-hand to tailor their systems to their needs. For instance, large multinational companies regularly describe their approaches, strategies, and "standards" (Shvachko et al., 2010; Szyperski et al., 2016), detailing the scale and complexity of their systems. Such resources are made available to publicly demonstrate their interests and investments in showing the potential of their solutions to be adopted as *de facto* standards. Survey studies exemplify the current motivators in the area: the multinational HR consulting firm Randstad (Randstad, 2017) has recently stated that Big Data engineer was the "best in-demand job for 2017".

Besides the industry-oriented challenges in defining systems and platforms, algorithms to process Big Data sets are also skill- and labour-intensive (Stonebraker et al., 2007). To address this challenge, special data centres/clusters have been built with the exclusive purpose of running

these intricate algorithms, as well as variants, – often built on non-linear data structures – to analyse complex data sets.

In short, managing data at this scale/speed consists of two major challenges. First, the infrastructure challenge involves the assessment of data storage and access Data Centres. Second, the process challenge includes the efficiency that can be achieved when running parallel and distributed jobs over data sets. From these two factors, there exist a plethora of sophisticated Big Data platforms, each designed to address a type of data structure, data set, or algorithm, with multiple parameters.

Hence, the research challenge we address in this paper is the possibility of building a simulator that has the ability to (i) model real Big Data platforms with a high degree of accuracy, (ii) capture the data movement/migration, like when data is transferred between nodes in the Big Data platform, with a high degree of accuracy, and (iii) scale up to large clusters of Big Data platforms.

The focus on the data movement/migration was chosen due to the related performance improvements, and the data and task placement will be better explained in the next sections. As HDFS has a strong random component in the data placement, and task placement must follow, this problem poses as a good one for complex simulations. When combined with the potential complexity of different network topologies/switching algorithms, the simulation can become even more complex.

As a motivating example, consider a group of Big Data engineers or architects who need to set up a cluster of machines for graph processing using Apache Giraph (Giraph, 2012). Apache Giraph is a Big Data platform for non linear data structures based on the Hadoop ecosystem and the Bulk Synchronous Parallel concept introduced first by Pregel (Malewicz et al., 2010). Selecting the best infrastructure matching the processing and storage needs is the first issue faced in setting up such a system, defining the number of racks, machines per rack, and networking interfaces. Then, the following setup issue comprises defining the proper data distribution strategy, such as partitioning algorithm and the job/task distribution when running data crunching algorithms, involving node selection and task assignment problems. Resource provisioning is a recurring problem in large-scale graph processing (Cela et al., 2018).

Many of these questions receive default answers by the different elements of the Big Data stack: there is a default partitioning algorithm in Giraph, there is a default replication factor in HDFS, and there is a default job/task scheduling algorithm in Hadoop. However, the proper selection of parameters and options in these systems can significantly impact performance. As a result, tools, such as Starfish (Herodotou et al., 2011), attempt to model the performance of the different system elements and to adapt the parameters of Big Data platforms. The adaptation is realised to help users of such systems, tackling the problem at many levels, from the infrastructure to the various available scheduling and placement algorithms.

However, what developers, as well as advanced users, of Big Data platforms could greatly benefit from a simulator that tells them what to expect with the various algorithms implemented in the Big Data platforms, especially *at scale*. For instance, these developers would likely use a simulator that shows them what is the best data placement or task placement algorithm for a certain configuration, or which network topology can provide more performance.

Therefore, we propose BigDataNetSim, a fully functional simulator of data and process placements, as well as networking/data transfer, for large scale Big Data platforms. BigDataNetSim is able to simulate most of the components of the Hadoop ecosystem. The current implementation of BigDataNetSim already realises HDFS, YARN, and MapReduce and focuses on simplicity of adding new components. Our proposed simulator has demonstrated, through

experiments, great performance, scaling to large realistic clusters, flexible adaptability, being configured with different network topologies and placement algorithms.

To achieve such objectives, we present the following major contributions:

- We propose a novel simulator for Big Data platforms, capable of running Big Data jobs on Big Data platforms and to analyse the performance, such as execution time/throughput, of the modeled platforms and jobs;
- We perform a thorough evaluation of BigDataNetSim, using a real cluster of 20 machines in 2 racks. We compare the data placement of HDFS – the data management component of the Hadoop stack – and the job scheduling/placement of YARN – the process management component. We also show the high accuracy of BigDataNetSim model for real systems. Results have displayed a difference of between 2.9% and 9.6% on average for data placement, depending on the file size, and 4.4% difference on average for job placement. We have also evaluated the scalability of BigDataNetSim and showed that we can easily simulate various classical Hadoop jobs.

3.1 BIG DATA NETWORK MODELING AND SIMULATION

BigDataNetSim is a simulator designed to execute analysis of data placement, task placement, and network parameters of Big Data clusters based on Hadoop (HDFS/YARN) and data processing engines running on top of it, like MapReduce, Hive, and Spark. BigDataNetSim is intended to simulate large cluster configurations, with several thousands nodes, allowing the analysis of network and data processing engine parameters on those large clusters. This reflects the evolution of Hadoop itself, considering that Hadoop v1 had limitations over 4000 nodes in a cluster, Hadoop v2 was designed for clusters up to 10000 nodes, and Hadoop v3 was designed to surpass 10000 nodes.

The simulator development was based on Java language, making available command line and web-based interfaces. Simulation parameters describe the cluster, the environment, and the network connecting the nodes. The following sections describe the requirements for the simulator and its architecture, internal operation, and usage.

3.1.1 Requirements for BigDataNetSim

BigDataNetSim was developed to support particular research tasks, regarding data and task placement, as well as network topologies and protocols. Consequently, its requirements are based on the type of analysis that the research needs to perform, which scenarios and tests are executed, and which data is needed as a result from the simulations.

In general, the simulator requires the following:

- Distributed File System: information about the files stored in the file system, the location of the file blocks among the nodes, number of replicas;
- Executing Tasks: number of tasks each job generates, where the tasks are running, from where each task is reading the data, as well as the classification of a task based on the location of data to read – local, rack, external.
- Network Infrastructure: amount of data that is passing through the network, the influence of both network topology and switching protocol working together, the time a particular quantity of data takes to be transferred over the network segments.

These needs will be explained with greater details in the next sections.

3.1.1.1 Simulation Scope

The typical scenarios envisioned for this simulation tool must reflect real clusters where 1000s of machines is not unusual (Villanustre, 2014). For the research on hand, we need to simulate large clusters of different sizes (500, 1000, 2000, 3000 nodes and so on) with different network organisations and strategies for the placement of data and tasks. In terms of network organisation, Hadoop configuration only registers in which rack each node is. In addition to that, in the simulations, we need to select which topology is used on the cluster, such as hierarchical, FatTree based, and new proposed topologies, and which switching protocol is used among the switches in the network, such as Spanning Tree, Shortest Path Bridging, and SDN based protocols. These features are complex and expensive to implement in a real cluster, so the simulator is crucial to help in the selection of the most appropriate and viable set of features, in a simulated environment, and later progressing for the physical tests with the best solutions.

As mentioned before, the reading part of a Big Data job is the only constant among the different tools and frameworks, so the research is focused on this particular phase, as it could be costly to simulate all these frameworks. Still, as the reading part is a costly one in terms of resources and time, optimising just this part can result in significant gains for the job processing, becoming a generic way to improve any available tool on top of HDFS. In this reading phase, the focus is on the data movement, affected for the data and task placement, and the underlying network structure, including the topology, protocols and network features. As a result of this focus on the reading phase, this simulator is not required to model the remaining phases of a Big Data job.

3.1.1.2 Remote Reading in Big Data Platforms

The simulation model assumes that data is distributed in a Big Data cluster, so is the task of reading those data, considering that HDFS is a common base for several big data tools and frameworks (Villanustre, 2014) and several cluster processing engines can be used over HDFS. YARN is a very common one and supports several frameworks, like MapReduce, Hive, and Spark. Besides that, YARN is the default processing engine available in Hadoop, making it easy for other tools to use it.

In general, Big Data platforms are based on the "move the code, not the data" concept. This means that, when possible, it is desirable to process data in a local way, transferring a piece of code – very small compared to the volume of data – to a node and then reading and processing data blocks stored on that particular node. Each node has limited resources to start containers to execute the code (Hadoop, 2008), thus, there is also a limit of the number of simultaneous tasks that can be executed at a given time in a specific node.

In real clusters, when the number of jobs and tasks is large, it is likely that processes cannot always be executed on the machines that are storing the data blocks. Hadoop does not have a very sophisticated algorithm for task placement as decisions have to be taken quickly and optimising task placement is a difficult (NP Hard) problem, which makes the issue even more prevalent. Eventually, there is a divide between data blocks and tasks that create a large overhead on the Big Data platforms' performance.

When the cluster is running a higher number of concurrent jobs, there is a chance that a task cannot find a suitable node with the data to be processed stored and, at the same time, an available core for local processing. In the end, only part of the tasks will be executed reading data from the local storage. If the local reading is not possible, Hadoop will try to find a node in the same rack to read the data from. If no node is available even in the same rack, Hadoop can read the data from any other node in any other rack.

Hadoop attempts to exploit the concept of "data locality" (Hadoop, 2008) as much as possible for defining placements. Following this concept, the tasks can be classified as "local", when reading data from the local storage, "rack", when accessing data from a different node in the same rack, and "external", when reading data from another node in another rack, as shown in Figure 5.1. As expected, the network latency increases when data locality decreases.

3.1.2 Architecture of the Simulation System

The architecture is described showing the simulator main internal components, as well as the main configuration points, and the main sources of abstraction to implement new algorithms. A description of the simulated object is also provided to help to understand the simulator features.

3.1.2.1 Simulated Object

The object to be simulated is the network traffic of the reading phase of a set of big data jobs, in a Hadoop (HDFS/YARN) cluster. This is affected by the data and task placement strategies, as these algorithms determine where the data is located in the cluster, and in which nodes the applications execute, and from which nodes the data are read. Network features, like the network topology, link layer features, and switching/routing protocols also play a role in how data will move in the cluster.

3.1.2.2 Main Components

Figure 3.1 shows the simulator main components and its relationship. At the top, we have the main configuration components, cluster and network. All the configuration parameters for cluster nodes, cluster infrastructure and network link layer are configured there, before the simulation begins. The parameters are then passed to the Simulation Engine.

The Data and Task placement strategy are shown at the left side, and these components determine how the data is stored in the distributed file system, and how the running jobs access these data, as previously explained. These two components are interfaces (abstract classes) in the programming architecture, allowing the simulator to use any implementation to simulate different scenarios and algorithms. The simulator provides a default implementation of these two interfaces, implementing HDFS current data placement algorithm and YARN based MapReduce task placement algorithm. In the course of this research, other data placement algorithms were successfully implemented and tested. The interface abstraction feature provides a convenient and inexpensive way to quickly test new proposed algorithms.

Then, the network processing, based on a graph representing the network structure, calculates the amount of traffic for each path in the network, generating data for analysis and reporting.

3.2 BIGDATANETSIM

Our simulator models a Big Data platform as $P = (V, E)$, where $V = \{v_i, \ldots, v_n\}$, $n \in \mathbb{N}$, is a set of nodes representing computing and network devices, such as switches, and $E = \{e_1, \ldots, e_m\}$, $m \in \mathbb{N}$, $\forall e_i, \exists (v_j, v_k), e_i = (v_j, v_k)$ is a set of connections representing network links with their own characteristics, such as bandwidth and delay. The simulator is designed to accommodate different switching protocols, like STP and SBP (Tsao and Hou, 2015). In general terms, these protocols are defined as policies that direct the behaviour of Algorithm 1, being major elements on the decision-making for selecting which (v_j, v_k) and e_i in a communication path – adding traffic

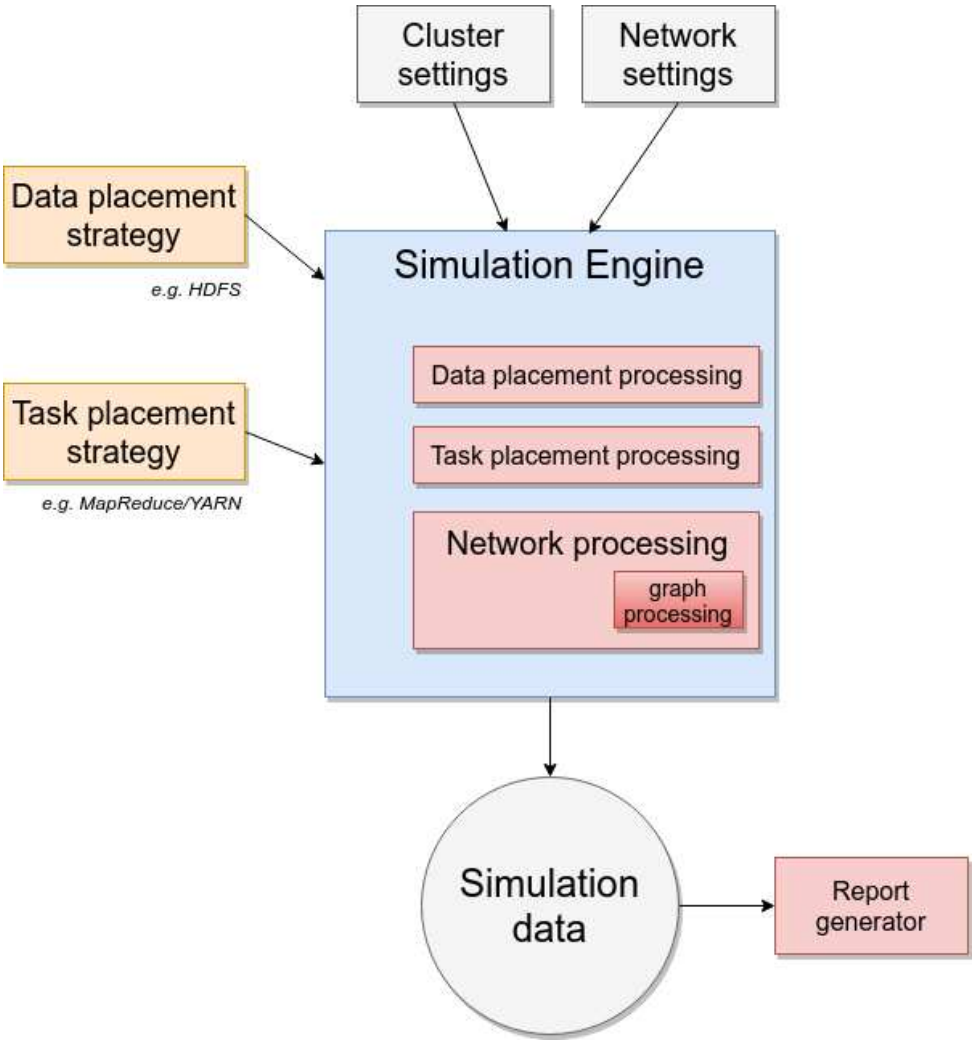


Figure 3.1: BigDataNetSim Architecture.

in the network. BigDataNetSim also models the classical Hadoop/Big Data algorithms, such as HDFS for data placement and YARN/MapReduce for job/task placement. These placement algorithms define how Algorithm 1 picks the $v_i \in V$ that are involved in the data/job placement and job processing towards a performance optimisation policy.

Algorithm 1 uses Shortest Path techniques to find the most appropriate path between the communicating nodes, with several degrees of complexity, depending upon the network topology. This can be simple in a standard hierarchical topology and more complex in a custom topology, like the one shown in Figure 3.2.

The details of how the simulator accomplish that are described in the following sections.

3.2.0.1 Parameters for the Simulations

We selected a set of parameters that best represented the real scenario, with the appropriate level of simplification, in the sense that not all the parameters available for a Hadoop cluster make a significant difference to the results we need.

In a general way, the parameters can be divided into three main categories, the cluster structure (including distributed file system), network structure and frameworks structure, as described below. In addition to that, there is also a set of parameters that describe a test execution, as described at the end of the section.

Cluster configuration: Number of nodes in the cluster, number of nodes per rack in the cluster, number of available processing slots in each node, nodes with different configuration in the number of processing nodes, allowing the simulation of homogeneous or heterogeneous clusters. The simulated file system can configure the size of a file block, the level of replication of each block and the read rate for the storage, including a value for overhead. All these parameters are discrete values, and can be set for each execution. For HDFS, the block placement policy can be configured, either using the default one, or a custom one, implemented using the simulator structure.

Network configuration: Network link bandwidth (classified by the topology layers), link delay, switch delay, network overhead, frame size, MSS (maximum segment size). Based on these parameters, the effective link bandwidth can be calculated. Apart from these discrete values, the simulator can configure other behaviours in the network: switching protocol (STP, SPB, SDN oriented, custom) and the topology. The topology can be set as a regular hierarchical topology, like a FatTree, or as a custom topology, defined using a graph. In either way, it is possible to configure the number of switches in each level or place and the connections between the switches, including redundant and loop connections.

Frameworks and policies/algorithms: These parameters control how jobs execute on the simulator, and have a direct impact on how the tasks will be distributed among the nodes, hence their importance.

Job processing engine: The simulator has one processing engine modeled: MapReduce over YARN. Additional engines are envisioned and in project to development, like Spark over YARN and Spark over HDFS. For YARN, the capacity scheduler has been modeled. For the MapReduce engine, some details are modeled, like turning on and off the speculative execution.

Test execution parameters: Number of concurrent jobs/users, including the range of concurrent jobs, from 1 to 256, for instance, and the step used, from 8 to 8, for instance. Number of tries/rounds for each test, to diminish any deviation caused for the several random aspects of the tests, this parameter is usually configured for at least a few dozens.

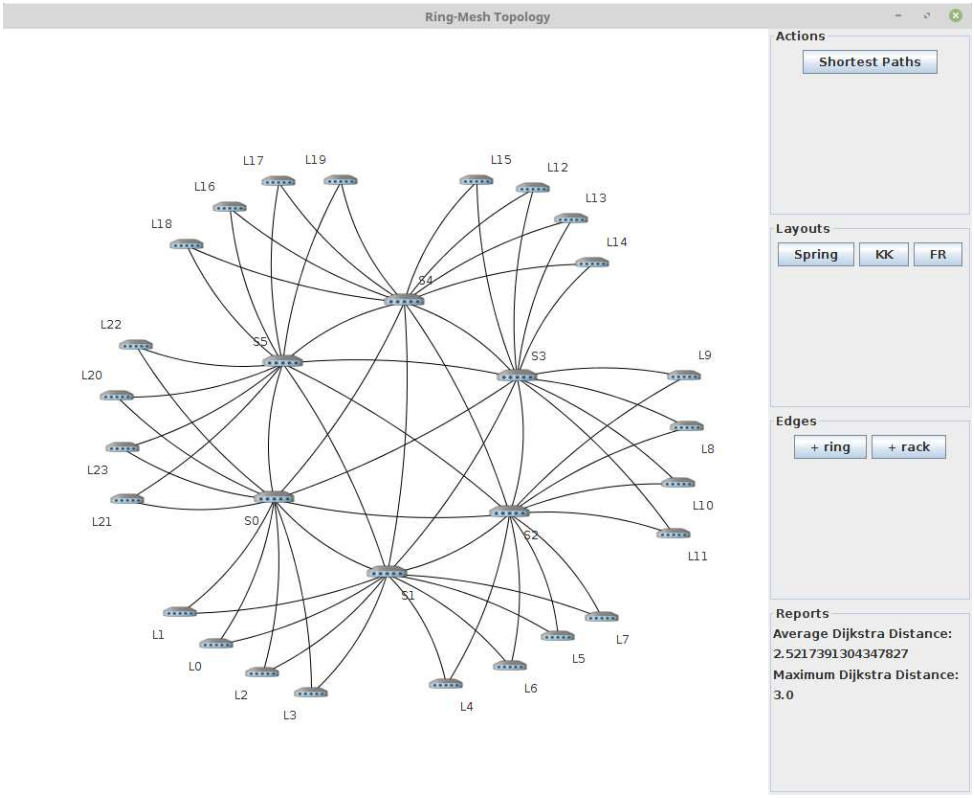


Figure 3.2: Network structure design.

3.2.0.2 Simulation Interfaces

The amount of available parameters and the number of generated reports can make the simulator operation complex, so user interfaces were developed to help each major use case to fulfil the objectives with the minimum overhead in operation.

The simulator was designed to be used mainly in two ways, as an exploratory tool, executing quick tests in different scenarios with different configurations, and as a batch tool, to execute a massive amount of tests for a particular configuration. Those two kinds of usage led to two main ways of using the simulator, through a Web Interface, and a CLI (Command Line Interface).

About the exploratory tests, these are used to quickly test a given configuration, to check if that particular setup is reasonable or not for a longer simulation. Those tests are often executed with reduced parameters and point to more extensive tests, once the proper configuration is decided. For this kind of test, the web interface was developed, with a predefined set of parameters and an intuitive interface to show quick results.

Command line interface: The main interface for using the simulator is the command line, and the parameters can be set either as a JSON configuration file, or a list in the command line itself. This allows the simulator to be included in larger batches of scenarios to be executed as a whole, in a script. The results are shown in standard output and saved to CSV files, if requested.

GUI helper interfaces for network design: The network topology design is a particular complex parameter in the simulations. Besides the conventional hierarchical topology, the simulator allows any custom designed topology to be used as the base infrastructure for the cluster network. For this reason, the simulator can export a graph file in a GEXF format (Bastian et al., 2009) to allow the analysis of a particularly complex topology in graph analysis tools, like Gephi, as shown in Figure 3.3. A GUI interface was developed that shows the resulting

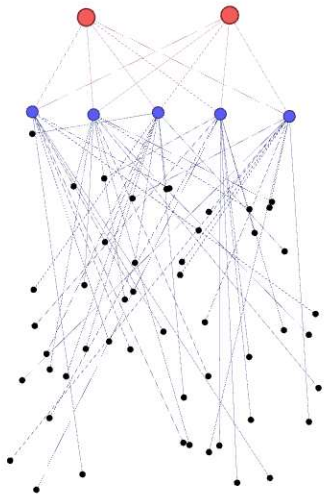


Figure 3.3: Network structure view in Gephi.

graph, allowing minor changes in the topology, like adding more connections between the Top of the Racks switches and the core switches, and increasing the density of links between the core switches, as shown in Figure 3.2. This GUI tool needs to be customised for a particular kind of network topology.

Web interface: As mentioned, the web interface was developed to allow quick exploratory tests, with a subset of parameters, in order to find proper configurations for more extensive tests. It was also developed to allow the simulator to be used by a larger team and take advantage of more processing power available in a proper server, rather than the user's own computer. In this case, the results are displayed as web reports, in addition to the CSV files. The web interface is shown in Figure 3.4.

3.2.1 Phases of a Simulation

To better understand how a simulation is conducted in the simulator, it is interesting to detail the steps of a simulation task, and how these steps are fed with the parameters and relate with each other.

In a general way, a simulation follows a workflow with this main phases, as shown in Figure 3.5:

- Cluster configuration: when the parameters for the file system, network topology and cluster are set, preparing the environment for the execution. Data placement strategy is also configured in this phase.
- File system generation: generation of the files, according with size and number parameters, and using information from the cluster and network configuration to apply the data placement strategy.
- Job configuration: parameters for the job execution are set on this phase, including the number of concurrent jobs, which file (s) will be used as input, which processing framework and its parameters and the strategy for task placement and scheduling.
- Simulation execution: execution of the simulation according to the configured parameters while storing the intermediate results.

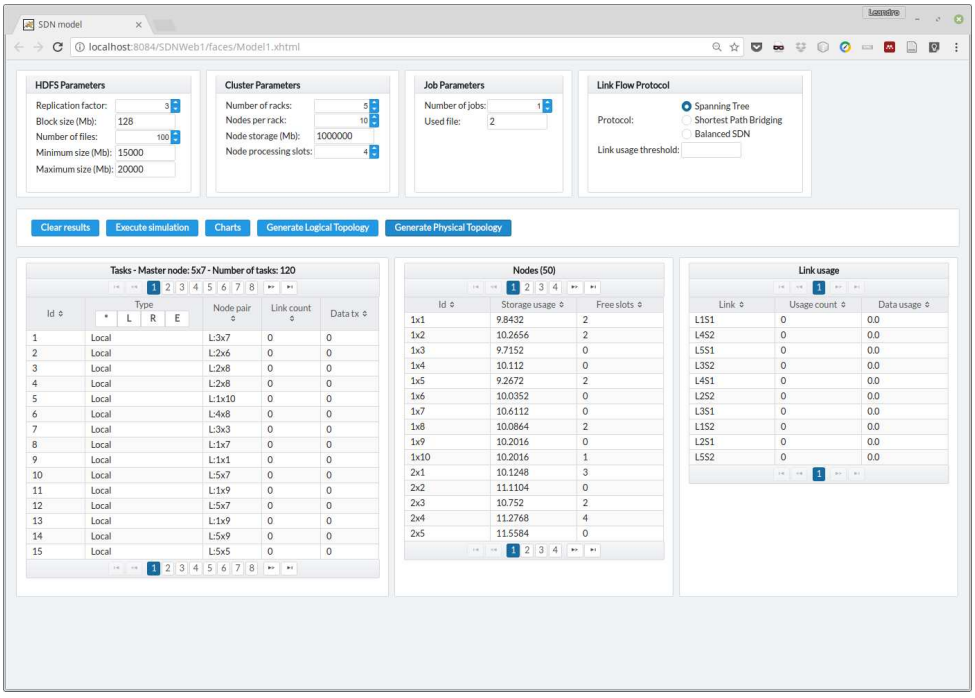


Figure 3.4: Simulator Web Interface.

- Report generation: aggregation of results and generation of the requested reports.

In the workflow, a critical phase is the process to calculate the amount of network traffic for the jobs execution, as described in the Algorithm 1. These values are particularly important for the research that started the simulator development, as network optimisations are the goal for that. In this sense, the information needed is the amount of data transferred between the nodes, and it's the impact on the network infrastructure, regarding the topology and protocols in place.

For this calculation, some information must be available:

- list of tasks of all running jobs and its locations in the cluster;
- list of all data blocks to being read for each running task, and its locations in the cluster;
- list of all network paths in the cluster, following the network topology;
- list of all nodes in the cluster and its locations in the network topology, in which rack each node is;
- network features (bandwidth, latency, etc).

As shown in the algorithm, the list of tasks is evaluated, and the external and rack tasks are selected for the network traffic calculation. For each task, the paths used between the processing node and the data node are obtained from the network topology graph, according to the switching protocol in place. All the traffic flows in all the paths are accumulated by the algorithm, and placed in a list. For this, the underlying network features are taken into account, like the frame size, payload size and overhead. The method to obtain the network path also considers the protocols in place, considering multiple paths and load balancing when available and requested. The analysis is executed later based on this list, following the options available in the simulator. The traffic can be considered as a simple accumulation or average, but can be also considered as a step list, when the network flow pairs are solved in order, regarding the available bandwidth in each network segment.

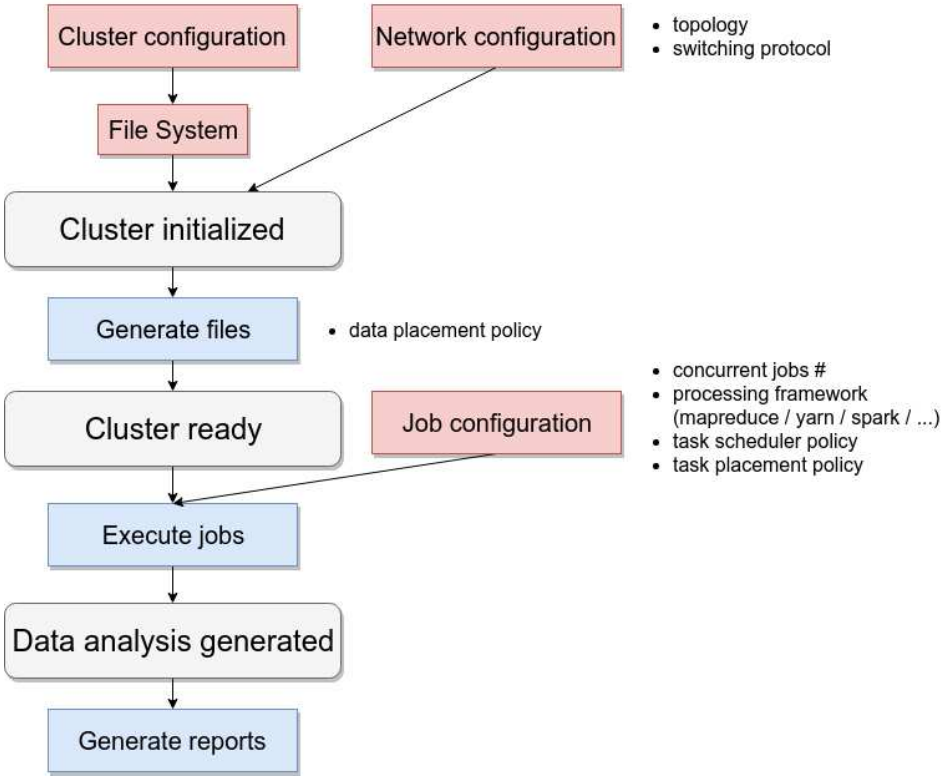


Figure 3.5: Simulation execution workflow.

Algorithm 1: Network Traffic Calculation Algorithm

```
// considering i,j as processingNode and dataNode
input  : jobTasks: List< task >, dataBlocks: List< block >, steps: List< path_{i,j} >
output  segmentList: List< segments >
:
for task ∈ jobTasks do
    if task.type ≠ LOCAL then
        // get processing and data node from task
        processingNode, dataNode ← getNodePair(task) // get network
            path between the nodes
        taskNetSegments ← getNetPath(processingNode, dataNode)
        // get the data blocks for the task
        dataBlocks ← getDataBlocks(task)
        for segment ∈ taskNetSegments do
            // aggregate traffic in list
            increaseTrafficSegment(segment, dataBlocks, segmentList)
return segmentList
```

3.2.2 Simulation Usage Examples

Based on the available features, there are some test scenarios better suited for the simulator, as follows:

3.2.2.1 Evaluation of data placement strategies on HDFS

These strategies can have an impact in how the tasks are executed, due to the need to read and aggregate the source data. The simulator provides a set of Java interfaces to implement the main methods that control the data placement, with the intention of making this implementation simpler.

3.2.2.2 Evaluation of task placement strategies or scheduling algorithms

These strategies and algorithms can also have an impact in performance and network usage, so the simulator was designed to test these as well. In the current state, the simulator can simulate MapReduce API over YARN, using the standard scheduler for tasks. As the class structure was prepared using the same level of abstraction of data placement strategies, more options will be implemented in the future (like Spark), besides custom strategies as well.

3.2.2.3 Evaluation of network topologies

The simulator has a conventional hierarchical topology implemented, with a Spine-Leaf strategy, and new ones can be implemented using graphs, which include complex topologies. At least one complex topology, based on rings and parallel paths, was implemented as an evaluation of data centre networks, using the simulator graph capabilities.

3.2.2.4 Evaluation of network protocols and routing/switching strategies

The protocols STP (Spanning Tree Protocol) and SPB (Shortest Path Bridging) are already implemented, and any cluster and topology can be tested using them. Other protocols or routing/switching strategies can be implemented in a simple way, using interfaces to order behaviour.

It is important to remember that the simulator implements the reading phase of big data job execution, not being suitable for simulations that involve other phases than this one.

3.3 EVALUATION SET-UP

We aim at answering the following three questions:

- **RQ 1**: Is BigDataNetSim's data placement model accurate?
- **RQ 2**: Is BigDataNetSim's job placement model accurate?
- **RQ 3**: How does BigDataNetSim scale to large clusters?

We used a real cluster of 20 machines (+ 1 master nodes) to evaluate the accuracy of our models (RQ 1 & 2). Installed version of Hadoop is 2.7.6 in a single and dual rack network configuration. The nodes are regular PCs, with Intel Core i5 processors, 8 GB of RAM and SATA hard drives with 1 TB each and the master is a Core i7 with 32 GB of RAM. The nodes are connected by regular Ethernet switches, with 1 Gb ports.

As the simulation must follow the features of the existing real cluster, we simulated a homogeneous set of nodes for the test bed. However, the simulator can handle heterogeneous clusters, configuring sets of hosts or each host with different parameters, as shown in Section IV. In the same way, in the tests, we simulated the same hierarchical topology present in the real cluster, although the simulator is capable of testing different topologies (as shown in Figure 3.2).

3.3.1 Jobs executed

As the simulator is designed to mainly generate data from the reading phase of big data jobs (the reading part of Map phase, in MapReduce jobs, for instance), we focused on the kind of tasks where this particular phase is prominent. We need jobs with significant amounts of data to read, in order to force a heavy network traffic in high level of concurrency situations, and as the remaining phases of the frameworks are not covered by the simulator, the tests will not cover the additional phases. The amount of data is also related with the size of the cluster, as the intention of the simulations is to use most or all of the network resources available. In this sense, we need to occupy the highest possible number of processing slots available in the cluster.

For these reasons, we selected 3 job types to test the simulator:

- TestDFSIO: Test job distributed with Hadoop that generates configurable amounts of data to write and read from HDFS using MapReduce jobs.
- WordCount: An standard job that counts the occurrences of words in a text data input.
- TPC-H query 6: A query based on a large reading portion and little processing running on a structure dataset.

As the TPC group is well-known for the standardized tests, most of our tests were conducted using this last one. As shown in the Listing 1, TPC-H Query 6 is based on a single table, thus a single CSV file in HDFS, and goes through the entire file to select part of the dataset to execute an aggregation operation.

Listing 3.1: TPC-H Query 6

```
SELECT
   sum(l_extendedprice * l_discount) as revenue
FROM
   lineitem
WHERE
   l_shipdate >= date '1994-01-01'
   AND l_shipdate < date '1994-01-01' + interval '1' year
   AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01
   AND l_quantity < 24;
```

As the reduce phase is limited to an aggregation function and a simple arithmetic operation, the reading part is the majority of the time consumed. This restriction allows to keep the tests focused on the results the simulator can obtain and not spend time in operations not covered by the simulator, like the shuffle and reduce phases.

3.3.2 Metrics

The simulator accuracy is based on two metrics, the degree of data distribution in the distributed file system and the distribution of tasks among the local, rack, and external classes.

For the data distribution metric, we measured the average number of blocks per node in the cluster and the standard deviation of this value. As most of the choices made by the frameworks rely upon the data locality, an accurate distribution of data in the simulated file system is important in order to have the most approximate results for the job execution.

For the task placement, the classification among local, rack or external shows how much data is transmitted over the network, and between which computers, and this can be modeled for different topologies and network features. This metric is important not only for being a measure of the overall performance of a job since data transmitted over the network poses as a bottleneck. The metric is also important for allowing a realistic analysis of the network paths used in different network topologies, using different network protocols.

Besides these two functional metrics, a performance analysis was conducted, to show how much time each simulation run can take in order to determine the expected time to complete a longer simulation.

3.4 EVALUATION RESULTS

We have conducted tests using the simulator and the described test bed. The results show that the simulator has reasonable accuracy when compared to a real setup. The tests in the physical cluster are the result of 30 consecutive executions, and in the simulated cluster, of 100 executions.

We presented three ways to compare and assess the accuracy and efficiency of the simulator: data distribution, task distribution in job executions, and a performance measurement of execution.

3.4.1 Data distribution in HDFS

The data distribution reports show how the blocks of a particular file are distributed among the cluster nodes. The simulation takes into account HDFS replication factor and the data placement policy.

For these tests, were considered the HDFS standard replication factor of 3, and the default block placement strategy. The tests were executed in files with 1, 3, 5, 10, 20, 30, 40 and 50 GB in the physical cluster, and in a simulation. Both clusters have 20 nodes and 2 racks. The charts showed the block distribution among the nodes, and the standard deviation for the files.

In Figure 3.6, we can observe the distribution of file blocks among the cluster nodes for a 50GB file. The values are ordered in descending order, to better compare the results. The distribution of blocks is similar, with a maximum difference of 7.7% and an average difference less than 1%.

Figure 3.7 shows the comparison of standard deviation for number of blocks per node for each file size. The standard deviation shows how even is the distribution of blocks among the nodes, and this can have an impact on how the processing engines locate the data. The standard deviation increases as the file size also increases, for both the physical and the simulated cluster, being slightly bigger in the simulations, but the progression follows the same pattern, with the difference decreasing as the file size increases. For instance, in the 1GB file, the difference in standard deviation between the physical cluster and the simulated one is around 19%, in the 50GB file is around 8%.

Judging by these results, we can conclude that the file system simulation is adequate when compared to a real testbed. It does not present differences that could affect the results of the simulations executed on the file system.

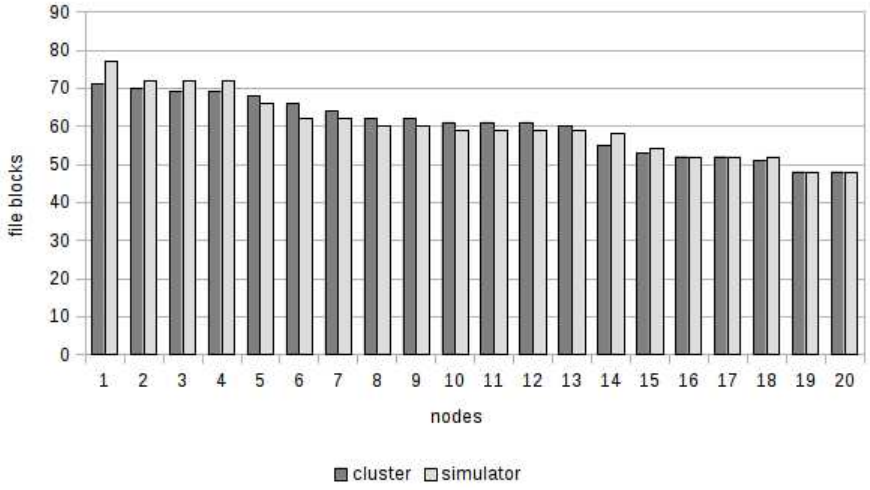


Figure 3.6: Block distribution - 50GB.

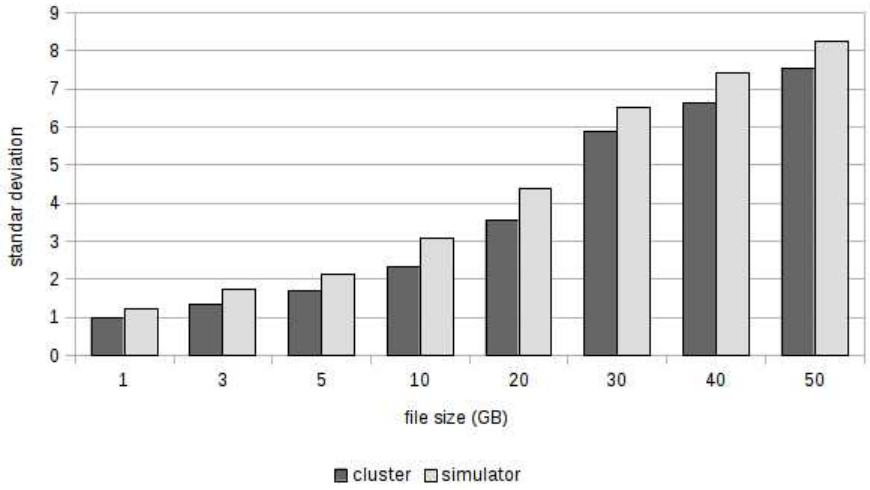


Figure 3.7: Data distribution comparison.

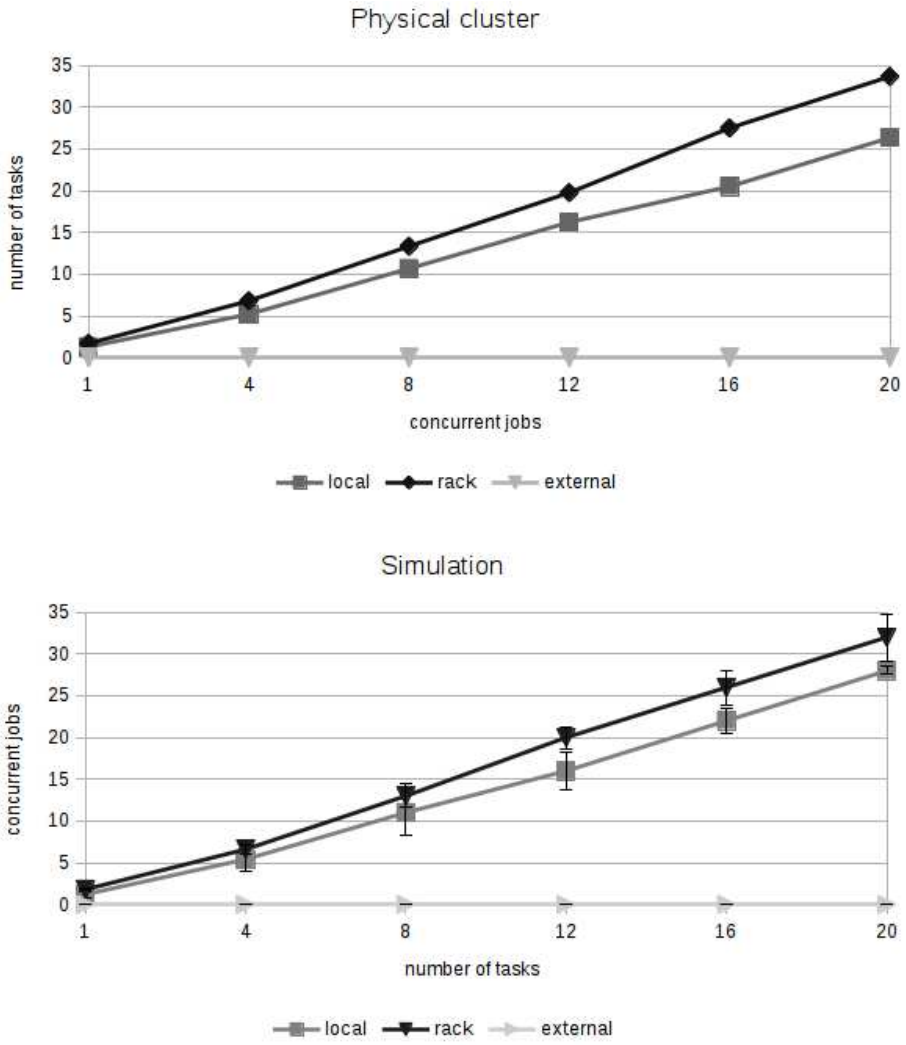


Figure 3.8: Task distribution comparison.

3.4.2 Task distribution in YARN/MapReduce

As mentioned earlier, the classification of tasks between local, rack and external is important to better measure the amount of data movement in the network.

For this, the tested scenario was planned to use all the available processing slots in the cluster, determined by the configured number of vCores in each node. The tests were conducted with concurrent jobs varying from 1 to 20, in steps of 4 (1, 4, 8, 12, 16 and 20). For a physical cluster of 20 nodes with 4 vCores per node, this is enough to occupy all the slots.

Both tests (simulator and physical cluster) were configured with the Speculative Map Execution as off, and with the yarn.scheduler.capacity.node-locality-delay turned on, using double the number of nodes as metric. Those parameters affect how many tasks are launched to satisfy a given map function, and how much time YARN will wait for an ideal node with data locality guaranteed (Hadoop, 2008).

Figure 3.8 shows the comparison between the results in the cluster and in the simulator, showing similar lines. The confidence interval is shown in the simulation chart. It is worth noting that the number of external tasks is low due to the cluster size. The average difference is 4.4%, with the difference being bigger with a small number of concurrent jobs, when the random parts of the algorithms, combined with an underutilised cluster, prevail.

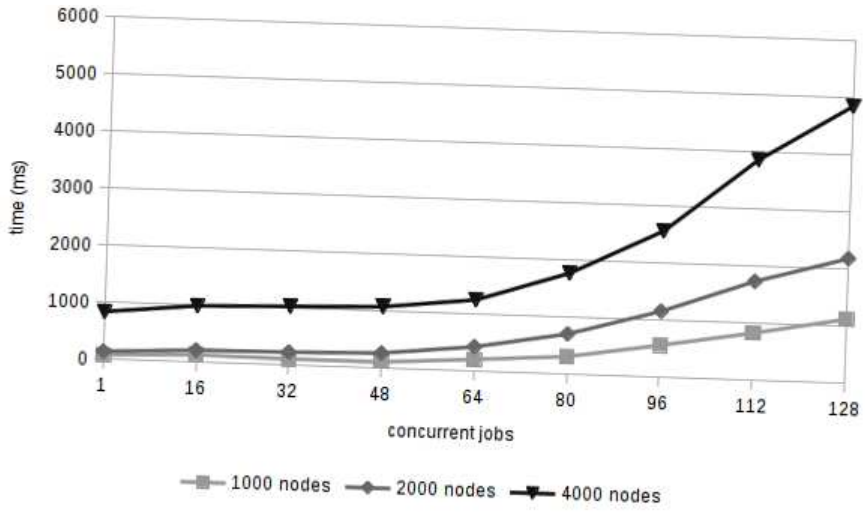


Figure 3.9: Scalability analysis.

3.4.3 Scalability and performance on simulations

Another important metric is the performance of the simulator itself. The amount of time spent by the simulator to run a particular test case was measured, and the results show that even for larger clusters, the performance is reasonable, even for a conventional desktop computer.

The tests were conducted in a desktop computer equipped with a Intel Core i7 processor (4 cores at 3.40GHz) with 32 GB of RAM memory, running Linux Operating System (LinuxMint v18.02) and Oracle JDK 8. All the tests results are from only one execution, but in real tests, it's expected to run multiple tries (up to 100) in order to avoid any statistical errors.

Usually, the simulator is used to execute a set of tests for a particular cluster and job configuration, running from 1 concurrent job to 128 or 256 concurrent jobs, in steps of 16 or 32. Each step is executed 100 times, and the results are aggregated. For some configurations, a test set like this can take more than 30 minutes to execute.

Figure 3.9 shows the results for 1000, 2000 and 4000 nodes clusters, running a number of concurrent jobs.

3.5 FINAL REMARKS

As shown by the experiment results, we can answer the research questions with some degree of confidence. BigDataNetSim is capable of simulating with accuracy the data placement algorithm of HDFS in order to provide accurate results for the simulations on top of it. Using the abstraction in its implementation, it is possible to implement and test novel data placement strategies and study their impact on the overall operation of a big data framework.

BigDataNetSim is also capable of accurately simulating the number of local, rack, and external tasks, which is important to provide accurate data for simulations involving network topologies and protocols, besides the task placement strategy itself.

Even not demonstrated in the experimental results, the provided graphical user interfaces showed very useful for the design, parameter setup, and visualization of novel network topologies, facilitating analyses simpler and easier to evolve.

In what regards its expected usage, the simulator presents some limitations. As the research that originated the simulator is focused on network information and reading phase, this focus limits the expected results and the coverage of the simulations. The simulations are not

considering subsequent phases of processing engines, like the combiner, sort or reduce phase of MapReduce. This decision in the modeling of the Big Data scenarios comes from the fact that these phases transfer less data through the network when compared to the reading phase. Also, these phases can vary in several ways apart from MapReduce when considering Spark, Impala, Tez, and others, making the simulation more complex. The impact is reduced by the more intense usage of the reading phase compared with the subsequent ones, especially in network transfer.

Ultimately, the simulator is useful for analysis and development of data and task placement strategies and network topologies and algorithms, for the impact on the network usage, and performance estimates.

3.5.1 Future Work

As future developments, we will implement new job processing frameworks, like Spark, and new scheduling algorithms. Some features of Hadoop 3, like the new HDFS Erasure Coding will be added as configuration options in the simulator, as these features can impact future research.

The use of an external tool like Gephi to help in the design of complex network topology will be studied. Currently, Gephi is used only to visualize the network topology, but this can be extended in the future, as well as a Mininet script generator based on Gephi files.

4 OPTIMIZATIONS IN BIG DATA SYSTEMS

This chapter presents two contributions of this research, one of them already published, in the paper Investigation of Replication Factor for Performance Enhancement in the Hadoop Distributed File System (Ciritoglu et al., 2018). The first contribution is about the concept of optimising performance of HDFS by increasing the level of block replication of the most used files, and the second one is about increasing the performance of HDFS by better placing the blocks of the frequently used files.

4.1 MODIFIED REPLICATION FACTOR

4.1.1 Replica Management

When data is put on to the Hadoop cluster, the default block placement of HDFS creates exactly 3 copies of each data block. HDFS places the first replica on one computer in the local rack, the second copy is put on another node in the same rack, and the third copy is placed on a different computer that is on a different rack. This placement strategy provides rack awareness and reduces traffic between racks. Since, data placement is important for data locality problems in Hadoop. There has been a lot of research on data placement(Xie et al., 2010; Jin et al., 2012; Eltabakh et al., 2011; Porter, 2010).

As the file blocks are replicated, they are distributed all over the cluster. When a job is submitted, the information must be first located before it is processed. Each part of a file used in a job is called a “split", and can be composed of one or more blocks. It is always more efficient to process on a local (data stored on the same node) rather than a remote node, and this is even more important when the data size is huge, as in Big Data. In this way, Hadoop will always try to process the data already found on the node instead of transferring the information. This is done by the principle that “Moving Computation is Cheaper than Moving Data" (HDF, a), where Hadoop will copy the executable code of a job to the nodes where the data is, since it is much more expensive to move the data.

4.1.2 Data Locality

There are 3 modes of task execution, related to data locality that is shown in Figure 4.1.

- **Local access:** when the split of data is stored in the same node running the task, e.g., in Figure 4.1, Node 1 processes block A.
- **Same rack access:** when the split of data is stored in another node, but inside the same rack (probably with better access through the network) as the node running the task, e.g., in Figure 4.1, when node 3 needs to process the block B, it requests the data from node 2 (which is in the same rack)
- **Off rack access:** when the split of data is stored in another node and in another rack, probably with a slower path through the network, e.g., in Figure 4.1, when node 4 needs to process the block C, it requests the data from node 8.

Hadoop will always try to execute in local mode, if possible. To ensure that, the YARN subsystem could even delay the start of a task with no local access to data, if there is a possibility of a free slot becoming available in a short period of time (White, 2015).

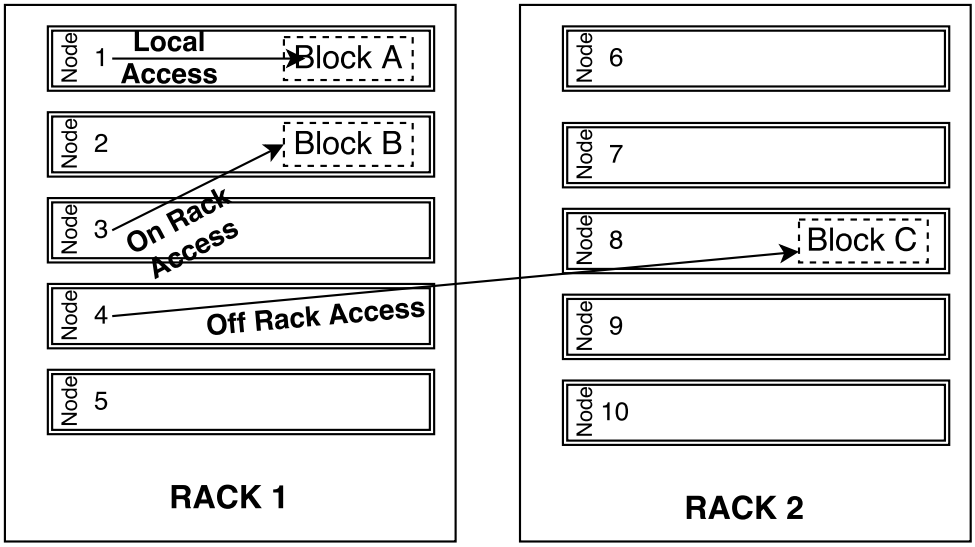


Figure 4.1: Data locality in Hadoop jobs

4.1.3 Data Replication

Data replication techniques can improve the data locality. In this study, performance tests are conducted to investigate the impact of replication factor on the data reading and overall system performance. Increasing replication factor comes with a price, which is the disk space and time to copy the data into the system. However, replicating only a certain portion of data (hot data) will reduce the overhead. We measured the data loading performance as the overhead of this approach. By increasing the replication factor, we increase the availability of the data and ultimately, job locality. So, the resource manager is much more likely to find a node which stores the data. Since the allocated node manager stores the related blocks, it can process these blocks locally rather than requesting them from another node and waiting for the data to be transferred. Our proposed solution increases the local accesses of jobs; consequently, we will see performance improvements. Furthermore, since data distribution is critical for the efficiency of the distributed system, we also investigated the data distribution of HDFS when the replication factor is modified.

It is worth noting that the design assumptions of HDFS assumes that a file once created, does not need to be changed, except for possible appends and truncates. The access model for files is the write-once-read-many, so we don't need to deal with replica synchronization (HDF, b).

4.1.4 Experiments

The experiments were conducted in an isolated laboratory cluster, which was not shared with any other processes. A physical cluster was used over a virtual cluster in order to avoid possible deviations in measurements. For example, a virtualised environment would be subject to additional latency for disk reads due to contention between VMs.

4.1.4.1 Methodology

The test methodology used for the experiments is described in this section. Each test was executed 10 times for statistical soundness, and the average results are presented. Before starting the tests, the data is loaded on to the Hadoop cluster, and then the replication factor is increased incrementally. After the test is finished, the data is deleted from the cluster. It is important to

note that we did not need to decrease the replication factor for future test runs. It was observed in early exploratory tests that decreasing the replication factor from 10 to 3 can cause a significant imbalance of data in the cluster.

The tests will execute using TestDFSIO, a test MapReduce job included in Apache Hadoop distribution, the NOAA dataset, with weather data, and TPC-H dataset, a well-know dataset used in database performance metrics.

We executed throughput tests in NOAA and TPC-H, testing concurrent users, in order to evaluate the performance gains when multiple users try to access the same data set. This reflects the usual cluster behaviour observed in industry, as the majority of the installed clusters are multi-tenant, supporting several applications and users at same time. The number of users for each test run was defined as ***users = {32, 64, 128}***. Each user is concurrently querying the data with the same query; there is a 1-second interval between the start of each query.

4.1.5 Results

The replication factor has an impact on the data availability, which can be related to reading time, and ultimately, execution time. In this study, we evaluated the performance of Hadoop using an increasing replication factor for hot data.

4.1.5.1 TestDFSIO

We assessed the reading throughput by using TestDFSIO with an increasing replication factor. Figure 4.2 shows that the replication factor can improve reading throughput significantly. The X-axis shows the replication factor related with the cluster size with 10 nodes. Having more copies of data increases the data locality, thus improving the reading time, and ultimately improving the job execution time. Two different sets of this data were created. Additionally, two different test beds, one with 10 nodes and another with 20 nodes, were created in order to understand the job's behaviour on different data set sizes. Despite the different volume of data, we see a linear reading throughput performance improvement when the replication factor is increased.

4.1.5.2 NOAA

Figure 4.3 plots the results of the experiment on the NOAA data set. Average execution time is reduced from 30s to 23s for 32 users, from 60s to 44s for 64 users and from 133s to 97s for 128 users by changing replication factor from 1 to 10. Considering only the gains when the replication factor for hot data is increased from 3 to 10, we can see a performance gain of approximately 13%. This gain is increased to 16% for 64 users and 17% for 128 users. The decrease in the job execution time can be attributed to an increase in the number of nodes (and the processing slots in each node) available with the data blocks needed for the mapper tasks. This increased availability supports an increased parallelism, and thus, a decrease in job execution. Also, we can see better performance improvements when data is accessed intensively, i.e., when the concurrency is higher.

4.1.5.3 TPC-H

Figure 4.4 plots the results of the experiment on TPC-H on the 20 node cluster using the 1st query. The improvement in job execution time is approximately 11% for 32 users, 13% for 64 users, and 15% for 128 users as the number of replicas were increased from default replication factor for hot data.

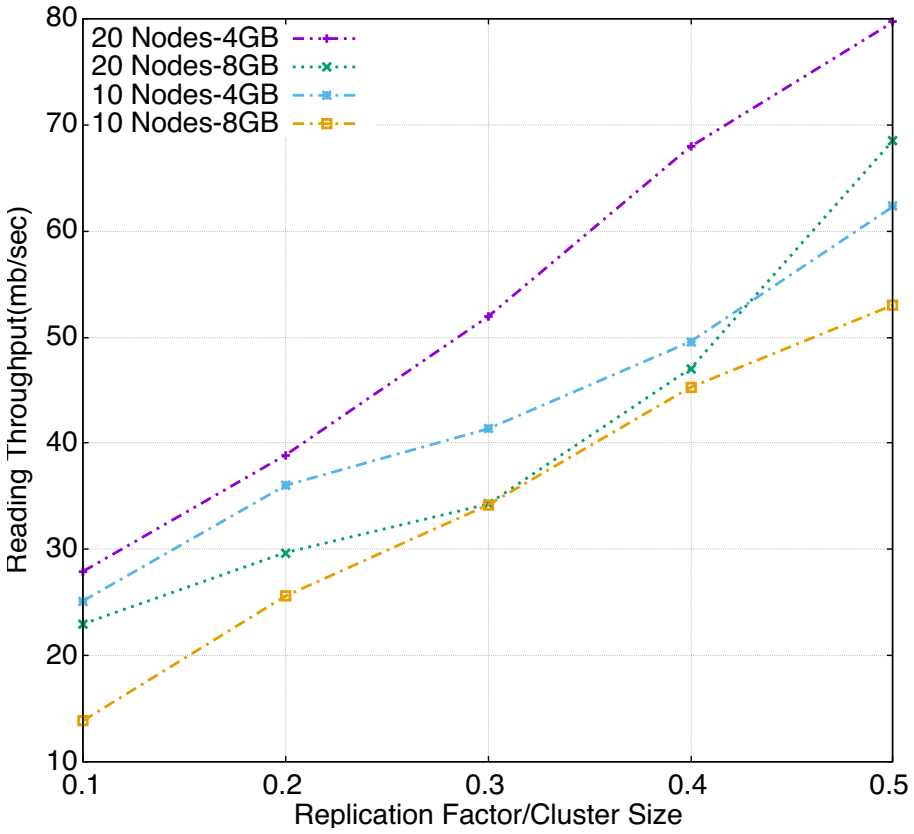


Figure 4.2: TestDFSIO - Reading Throughput on 10/20 Nodes

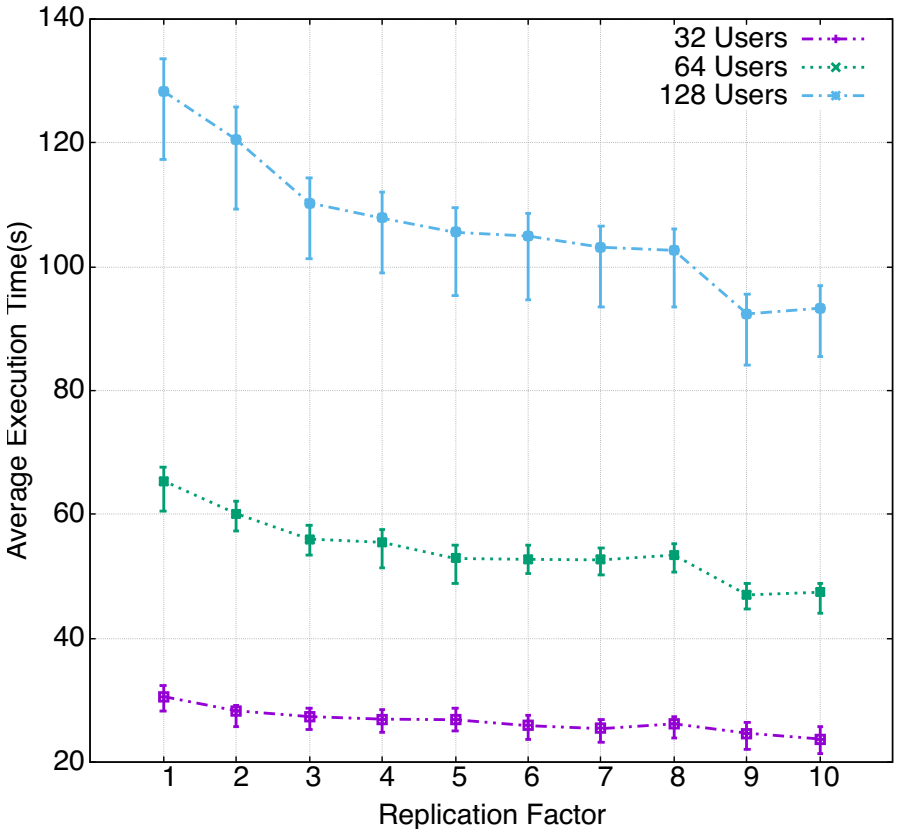


Figure 4.3: Replication factor test with concurrent user on NOAA on 20 nodes test bed

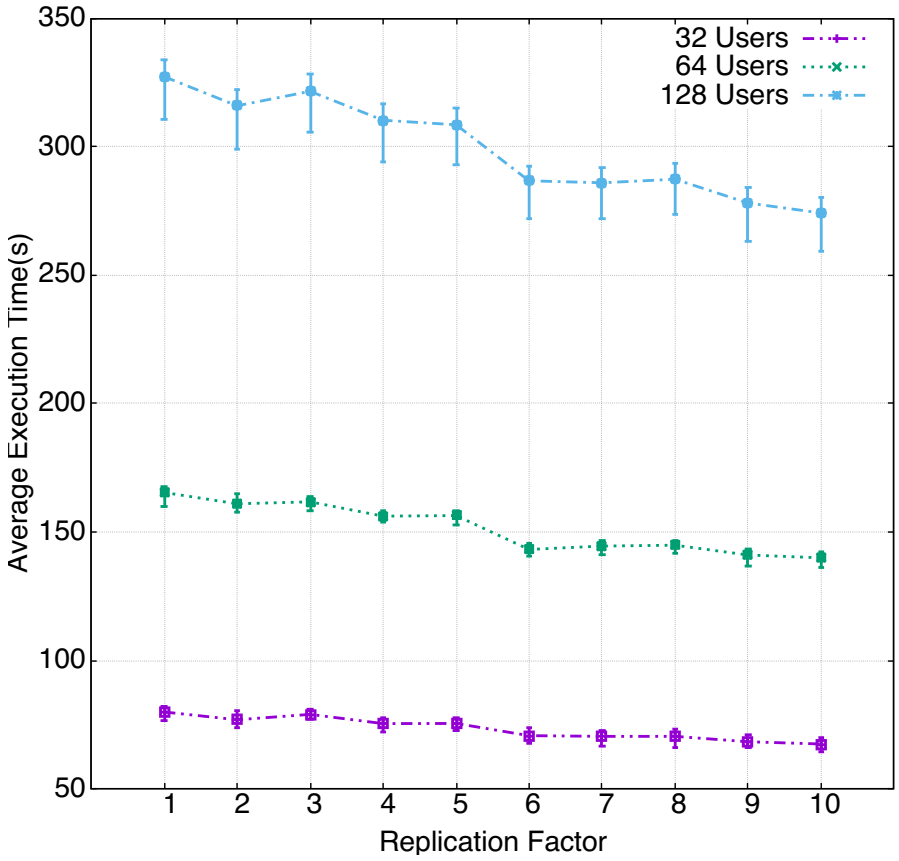


Figure 4.4: TPC-H Q1 - Replication factor test with concurrent user on 20 nodes test bed

Figure 4.5 plots the results of the experiment on TPC-H on the 20 node cluster using the 6th query. It can be seen that the job execution time is reduced by approximately 20% for the test with 128 concurrent users and 17% with 32 and 64 concurrent users when the replication factor were increased from 3.

In both experiments improvements can be seen until the maximum replication factor tested. It's important to note that, when the data is heavily accessed and requested by concurrent queries, the performance improvement is more compelling. Additionally, the performance gain trend is similar in both experiments.

Similar to the results in Figure 4.3, the job execution time plateaus near the maximum replication factor tested with concurrent users. However, if the data is not concurrently queried or there are only a few jobs running at the same time on the cluster, the performance gain becomes marginal at a replication factor of 5 - in this test - showing a relationship between the replication-based performance increase limit and the size of the cluster used. The results show that when the hot data is frequently and concurrently queried or the cluster processing utilisation is very dense, replication will play a key role in the system performance. Therefore, increasing the replication factor for hot data improves the system performance notably.

Experiment results show that the performance of Hadoop is tightly coupled with the data availability, access latency and executing mappers locally. Although, increasing the replication factor can create a storage overhead, increasing the replication factor for the hot data only will minimise this overhead.

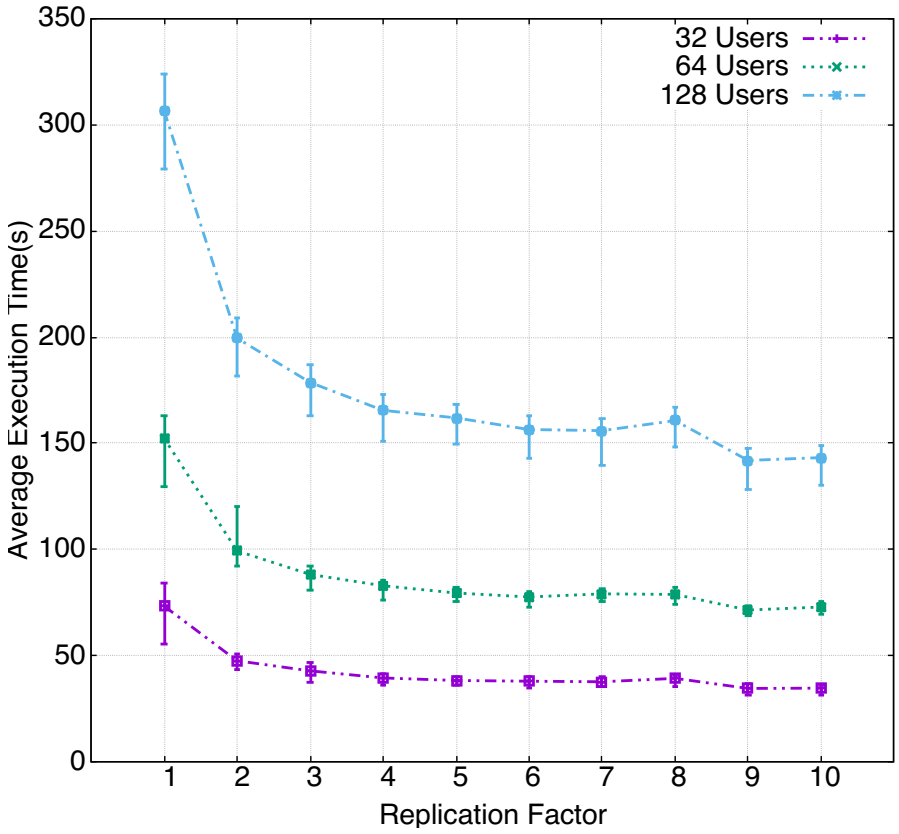


Figure 4.5: TPC-H Q6 - Replication factor test with concurrent user on 20 nodes test bed

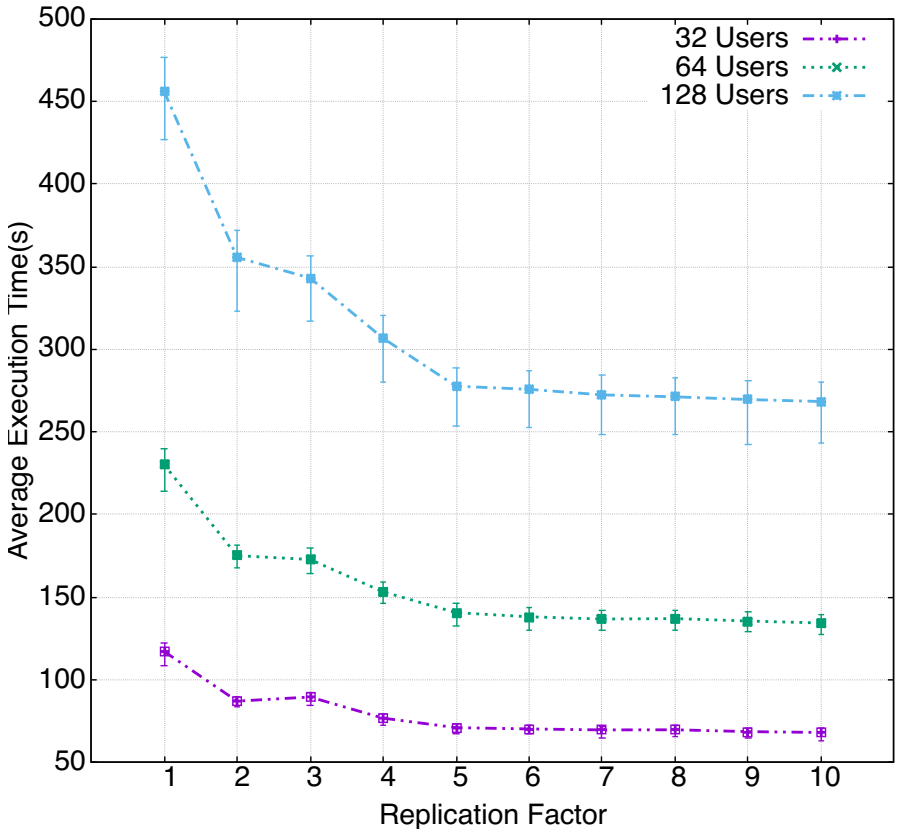


Figure 4.6: TPC-H Q6 - Replication factor test with concurrent user on 10 nodes test bed

4.1.5.4 Impact of Cluster Size

To explore the impact of cluster size, we used 2 different test beds, a 10-node cluster and a 20-node cluster. Both Figure 4.6 and Figure 4.5 show TPC-H Q6 results on different cluster sizes. Results reveal that, the performance improvements of approximately 41% are achieved for the tests on 10 nodes with different numbers of concurrent users; however, these improvements are approximately 53% on 20 nodes, when we increase replication factor from 1 to 10. So the impact of increasing the replication factor has a much more significant effect on system performance when the cluster has more nodes. Therefore, it is reasonable to say that continuing to increase the replication factor on a real production cluster (thousands of nodes), could afford an even greater performance improvement for larger numbers of replicas.

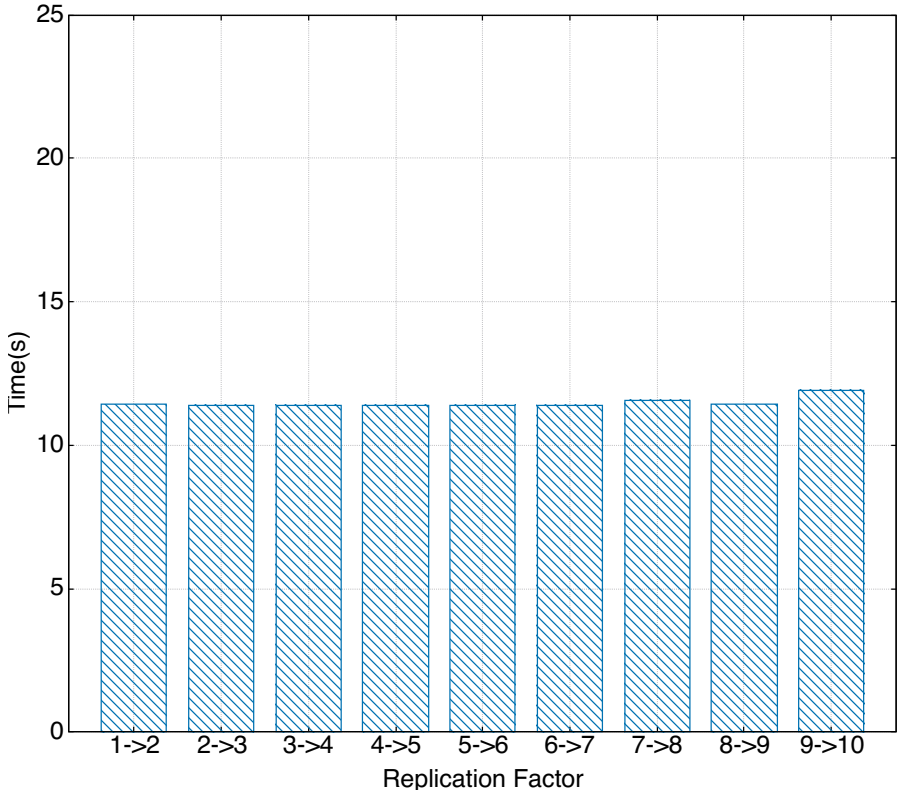


Figure 4.7: Data loading time for TPC-H Q6 on 20 nodes test bed

4.1.5.5 Data Loading Overhead

When data is replicated on a cluster, it adds an additional overhead. Since increasing replication is the core of our proposal, the data loading time overhead introduced was measured; this is shown in Figure 4.7. Creating additional copies of data blocks takes a similar amount of time for every increment of replication, which is approximately 11 seconds. On the other hand, each query that runs on hot data, can reduce the execution time up to 40 seconds. When we consider hot data is getting queried simultaneously, the performance improvement of the system will be very significant even if increasing the number of replicas introduces an additional overhead. This overhead is relatively low when we compare against the performance enhancement of the cluster.

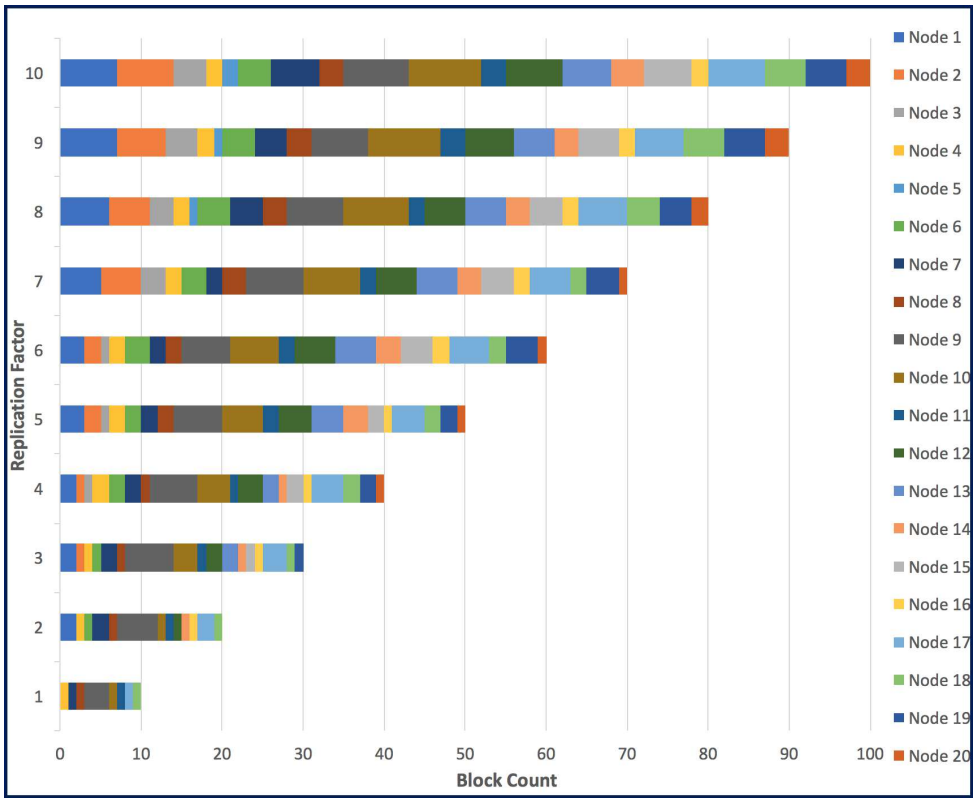


Figure 4.8: TPC-H Q6's hot data distribution on datanodes on 20 nodes test bed

4.1.5.6 Data Distribution-Replication Factor

Hadoop employs a data placement algorithm when replicating data on the cluster, which is quite effective in terms of achieving a balanced distribution of data blocks in a homogeneous cluster. This balance is maintained when the replication factor is increased incrementally from 1 to 10. Figure 4.8 shows how the data is stored after the replication factor is modified. Using replication factor 1 (just one copy), all blocks are stored in only 8 different nodes. When the replication factor was increased to 3, 17 nodes were allocated blocks of data for storing. We can see after 8, all computers store hot data and the data is balanced across nodes in the cluster.

Figure 4.9 details the block distribution after placing the data into the Hadoop cluster. The standard deviation for this distribution is approximately 5, which means that the data is not gathered in a small portion of the nodes. It is distributed equally over all the nodes, which eliminates hot spots in the cluster.

However, during the experimental phase of this work, we identified an issue in the Hadoop replica deletion algorithm. We tested a methodology of replication adaptation, where the replication factor was increased incrementally from 3 to 10 and then reset from 10 to 3. The results in figure 4.10 shows the block distribution after setting replication factor from 10 to 3, with 5 different sequential runs. The standard deviation in figure 4.10 starts at 39 and increases every time the replication factor is reduced. The data distribution becomes imbalanced, where only a few nodes keep significantly more blocks than others. Thus, these nodes turn into hot spots of the cluster and they have the potential to cause a performance degradation. The reason for this balancing issue is due to the Hadoop replica removal algorithm. Hadoop gives priority to the oldest heartbeat and if this interval is bigger than *(heartbeatInterval * tolerateHeartbeatMultiplier)*, it removes the data from that node, otherwise it removes replicas according to the free spaces on the nodes. Therefore, data can be easily unbalanced as shown in

Figure 4.10. The deletion algorithm does not consider the balancing of the block distribution. This may have a significant impact on adaptive replication factor schemes and must be addressed in their design.

Round / Node	1	2	3	4	5
Node 1	31	34	36	20	27
Node 2	26	26	38	28	30
Node 3	23	26	31	29	27
Node 4	31	25	25	24	33
Node 5	24	36	27	26	28
Node 6	35	24	26	38	29
Node 7	20	25	26	33	34
Node 8	23	31	29	27	23
Node 9	31	22	34	27	23
Node 10	25	17	29	29	32
Node 11	35	29	30	24	29
Node 12	27	25	28	26	25
Node 13	33	24	21	26	31
Node 14	29	32	21	30	18
Node 15	33	35	25	31	37
Node 16	25	35	20	21	29
Node 17	22	30	29	36	22
Node 18	21	23	19	26	26
Node 19	34	25	29	26	19
Node 20	27	31	32	28	33
Std Dev	4.8652906	5.09772911	5.13886123	4.41140865	5.02493781

Figure 4.9: Default data distribution using a weighted colour on 20 nodes test bed

Figure 4.11 shows the data distribution on the last state of block placement in figure 4.9 and figure 4.10. Having an evenly sized variety of colours shows data that is well distributed. However, larger segments of some colours represent a larger portion of data that is stored on a particular node. So, having multiple colours with average sizes in the figure means that the data is distributed well and equally among the nodes of the cluster.

In Hadoop jobs, mapping tasks typically take more time than reducing tasks. During our experiments, we observed a performance improvement on the job's mapping phase when we increased replication factor. Also, the experiments show that, once the data availability threshold is reached, no more performance gains can be achieved by continuing to increase the replication factor. It is important to note that this threshold depends upon several factors, like the cluster size and node processing capacity, not being a fixed value. It is also important to note that gain is coupled with cluster size, and our tests are conducted on a 10 - 20 node cluster to demonstrate this. In our case we can see performance improvement up to a replication factor of 9 on a 20 node cluster and 5 on a 10 node cluster. Although, we increase the replication factor for only hot data, the Hadoop job execution takes approximately 18% - 20% less time than default replication configuration.

4.1.6 Replication Summary

The focus of the research detailed in this section was to investigate the replication factor employed by HDFS. Specifically, to determine if increasing the replication of the most accessed data can

Node \ Round	1	2	3	4	5
Node 1	0	0	0	0	0
Node 2	143	163	173	182	184
Node 3	74	0	77	80	0
Node 4	69	0	48	57	0
Node 5	0	68	12	18	66
Node 6	64	0	64	37	0
Node 7	56	20	79	73	6
Node 8	0	0	0	0	0
Node 9	1	70	0	0	72
Node 10	0	0	0	0	0
Node 11	19	34	14	1	23
Node 12	0	0	0	0	0
Node 13	0	0	0	0	0
Node 14	0	72	0	1	63
Node 15	1	40	30	33	50
Node 16	72	0	57	64	0
Node 17	35	4	0	0	3
Node 18	5	77	1	9	71
Node 19	16	7	0	0	17
Node 20	0	0	0	0	0
Std Dev	39.4500017	42.9355392	44.4781141	45.6887352	46.0935833

Figure 4.10: Data distribution after reducing replication factor to default using a weighted colour on 20 nodes test bed

have a positive impact on performance; and test this proposal using a correct methodology. We constructed a physical Hadoop cluster running Hive and executed a comprehensive set of experiments on the TestDFSIO, NOAA, and TPC-H data sets, with different cluster sizes, and varying numbers of concurrent users. Despite the small size of the cluster, results clearly show that creating more copies of the hot data increases the availability of the data. This increased availability can support additional concurrent mapper jobs, thus, reducing the job execution time. Our cluster consists of only a maximum of 20 nodes, and therefore, the performance gains we achieved are limited. When the replication factor is increased from 3 to 9, we see approximately 18-20% decrease in job execution time, which remains fairly stable after replication factor of 9. The reason for this is that all data nodes in the cluster are well-balanced at 9 replicas, and therefore the number of mapper tasks that can be executed in parallel reached the maximum at this point. However, if the cluster was larger (e.g., thousands of data nodes), it is reasonable to expect the level of parallelism of jobs (specifically mapper tasks in Hadoop) to be much greater and therefore the job execution time to be reduced considerably further.

During performance tests, we also identified that reducing the replication factor can lead to a data imbalance on the cluster. Even though the replica placement algorithm is rack aware, it only considers the heartbeat signal time and balancing free space, not balancing 'hot data'. Therefore, if the popularity of data is decreased (data 'cools' down) after sometime, processing this data could be a bottleneck for the system. Also, results show that if the popularity cycle of data is very dynamic and changes quickly over time, this leads to a heavily unbalanced cluster. Especially, when we consider that Hadoop clusters are long-running systems, therefore, trends can change over time. Consequently, any bottlenecks could have a severe impact on the system performance.

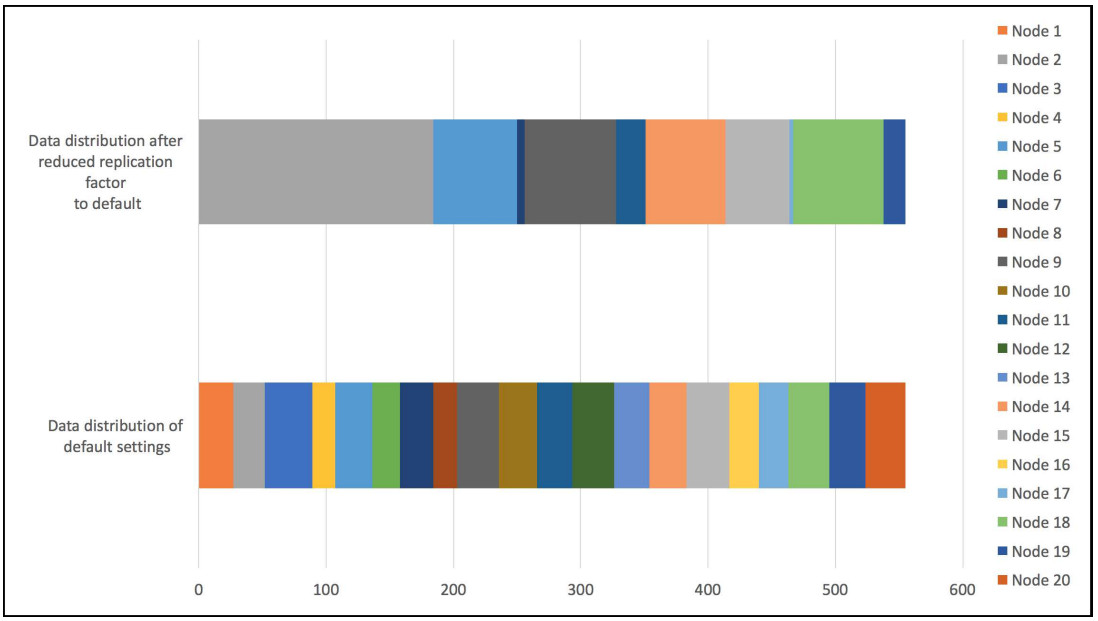


Figure 4.11: Data Distribution on datanodes with different methodology on 20 Nodes Test Bed

4.2 DATA PLACEMENT

4.2.1 Big Data Systems and HDFS

The common ground for the majority of Big Data technologies is the use of the distributed file system, HDFS, to store the source data and the results for the jobs. Despite several of these technologies potentially using other processing engines apart from Hadoop YARN (like Hive with TEZ and Spark), all of them read the data from nodes in a HDFS cluster.

This is the reason this project research for improvements in HDFS performance, particularly in the reading process, as the reading is required for any of the cited technologies, going beyond just Hadoop/YARN. By increasing the performance of HDFS, we can increase by extension all the tools that use it as a basis.

HDFS provides a way of distributed processing access the distributed data in a local fashion, i.e., the code is transferred to a node where the data resides, in order to reduce the network traffic. As the pieces of code are significant smaller than the pieces of data (a jar file have an common size of dozens of KB, and a block of data is at least 128 MB (Hadoop, 2008)), it is always more efficient to transfer the code and not the data, due to the respective sizes (Santos et al., 2018).

Observing how the HDFS works, particularly when we have several concurrent jobs using the same part of the data (so called “hot data”), we can observe that the number of tasks using the network connections increase as the number of concurrent jobs also increases. This is due to the fact that when the available processing slots (virtual cores) are depleted for a particular node, the standard Hadoop algorithm selects a different node to run the task, and the data needs to be transferred through the network.

As a Hadoop cluster can have hundreds or thousands of nodes, it’s usual to divide these nodes in racks, effectively dividing the network traffic and organizing the network in a hierarchical way. As the Hadoop system is aware of this network division (Hadoop, 2008), if the local processing slots are depleted, Hadoop tries to find a node with available slots in the same rack first, in order to keep the communication delay slower, containing the traffic inside a rack. Eventually, when the number of concurrent tasks using the same part of the data is increased

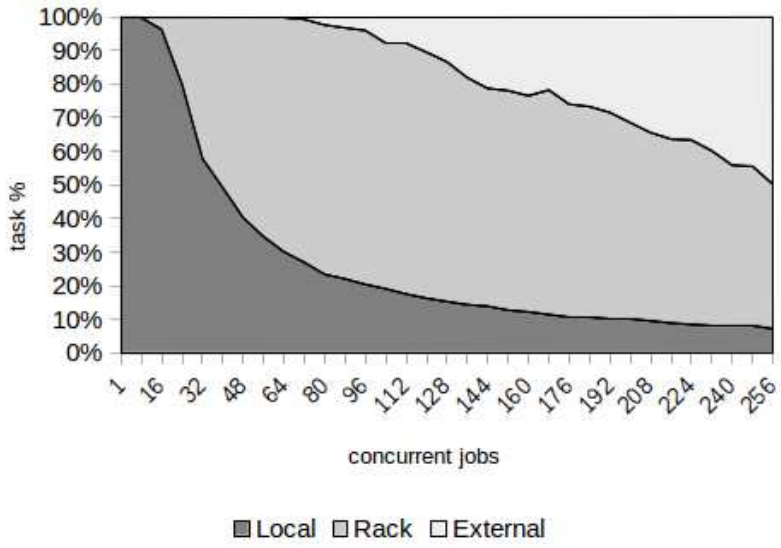


Figure 4.12: Comparison of Local, Same Rack and External tasks in Hadoop/YARN jobs

enough, the processing slots will be also depleted in the same rack. In this case, the Hadoop system allocates an available slot in another rack, which can increase the communication delay even more, as the network packets need to travel through a higher number of network switches, and possibly dispute the communication medium with other concomitant connections.

When such remote tasks become the majority of the running jobs in a given cluster, it can be expected an increased delay in the final execution time. This situation can happen more often when part of the data is more used than the rest, creating a "hot data" portion of the data space. This is even increased when multiple concurrent jobs are accessing this "hot data" at the same time, increasing the number of the remote tasks, in opposition to the local ones. This is shown in Figure 4.12, when we can see the distribution of Local, Same Rack and External tasks, showing that with few concurrent jobs (less than 16, in that case), most of the tasks are using local data, and with 256 jobs, the proportion of external tasks (reading data from a node in another rack) grows to 49.8% in a typical Hadoop test execution in our simulator.

To help decrease the external tasks, and by that, lowering the network delay and increasing the performance of Big Data jobs, this part of the research project chose the data placement approach, aiming to decrease the network usage in the reading data part of the jobs. The new data placement algorithm is effective in situations of high concurrent usage of the same parts of the data, the "hot data" portion. We obtained a significant decrease in the network delay in our simulation tests, giving room for the overall performance improvement of the jobs. It is important to remember that there are several factors contributing with the performance in a big data cluster, including the performance of the storage devices (hard drives, SSDs), the processing power (CPUs, memory) and the network, which is particularly important when the high concurrency situation occurs, due to the contention to read the information through the network connections, giving the reason for this research project.

The main research question to be answered here is: can we improve a big data cluster performance (particularly in network throughput) by improving the data placement algorithm in the cluster distributed file system?

4.2.2 Improving HDFS Performance

As all the components of a regular Hadoop stack are using HDFS to read and write information, any improvement done in its performance could have a positive effect on the rest of the stack.

In this section, we present a new data placement algorithm that benefits from the additional available processing slots in nodes distributed in the cluster, considering an heterogeneous set of nodes, with part of the nodes having more cores than the rest.

4.2.2.1 Objective

In our work, we aim at minimising the execution time of multiple Hadoop queries, in a given network topology and link layer protocol. Given the complexity of modeling such an objective, the real cost of every block placement can only be obtained from its simulation. However, we believe that reducing external and rack tasks (i.e., tasks requiring data from a machine different from the one they run on) can help on the objective, especially when this information is combined with the underlying network architecture. This is motivated by the fact that execution nodes waste CPU computation time while waiting for data transfer over the network, and in some topologies, when several concurrent jobs are using the same portion of data in the cluster, the bottlenecks are more severe.

Trying to keep most of the reading tasks reading from local storage can significantly reduce the network traffic and bottlenecks, and in this work, we study the network behavior related to the most common link layer protocols (STP and SPB), and in a spine-leaf physical topology

4.2.2.2 Problem Definition

A Big Data cluster is composed of a set $\mathcal{R}$ of racks (a.k.a., servers). Each rack contains one or several machines $m_u \in \mathcal{M}$ with $\mathcal{M}$ the set of machines in the cluster. Each machine $m_u \in \mathcal{M}$ has a finite amount $Q_{m_u,q}$ of resource $q \in \mathcal{Q}$ (e.g., CPU, RAM, storage). The assignment of machines to racks is a mapping: $MR : \mathcal{M} \mapsto \mathcal{R}$, such that $MR(m_u) \rightarrow r_v$.

A Big Data system is composed of a set of files $f_i \in \mathcal{F}$. Some of these files are highly accessed by users (i.e., hot files $\mathcal{HF} \in \mathcal{F}$), while others are accessed occasionally (i.e., cold files $\mathcal{CF} \in \mathcal{F}$). Each file requires a certain number of fixed size blocks $b_{i,k} \in \mathcal{B}_i$: $FB(f_i, \mathcal{B}_i) \rightarrow \{b_{i,1}, \ldots, b_{i,n_i}\}$ with n_i the number of blocks required by file f_i.

Every block $b_{i,j}$ that belongs to a file $f_i \in \mathcal{F}$ is replicated ρ_i times with replicas $Replicas(b_{i,j}) = \{b_{i,j,1}, \ldots, b_{i,j,\rho_i}\}$ for reliability purposes. We consider $Replicas$ as the set of all block replicas in the cluster. All replicas $b_{i,j,k}$ of a same block $b_{i,j}$ use the same storage resource (e.g., 64MB in Hadoop HDFS): $Q_{b_{i,j},storage}$, with $storage \in \mathcal{Q}$.

The assignment of replicas to machines is a mapping: $RM : Replicas \mapsto \mathcal{M}$, such that $RM(Replicas, \mathcal{M}) \rightarrow m_u$.

4.2.2.3 Constraints

The assignment of the replicas to machines is subject to various set of constraints.

4.2.2.4 Assignment Constraints

Consider a binary variable $x^u_{i,i,k}$ for every block replica $b_{i,j,k} \in Replicas$ of a block $b_{i,j} \in \mathcal{B}_i$ and for each machine $m_u \in \mathcal{M}$, which is set to 1 if $BM(b_{i,j,k}) = m_u$ and 0 otherwise. Constraints (4.1)

ensure that every replica is reassigned to one and only one machine. $\forall i \in \{1, \ldots, |\mathcal{F}|\}$, $\forall j \in \{1, \ldots, |\mathcal{B}_i|\}$, $\forall k \in \{1, \ldots, \rho_i\}$:

$$\sum_{u=1}^{|\mathcal{M}|} x_{i,j,k}^{u} = 1 \tag{4.1}$$

4.2.2.5 Capacity Constraints

The assignment of block replicas to machines follows a capacity constraint: a machine cannot host more blocks than its storage capacity (see Constraints 4.2). $\forall u \in \{1, \ldots, |\mathcal{M}|\}$:

$$\sum_{i=1}^{|\mathcal{F}|} \sum_{j=1}^{|\mathcal{B}_i|} \sum_{k=1}^{\rho_i} Q_{b_{i,j},storage} x_{i,j,k}^{u} \leq Q_{m_u,storage} \tag{4.2}$$

4.2.2.6 Conflict Constraints

We consider that replicas $b)i, j, k$ of a same block $b_{i,j} \in \mathcal{B}_i$ cannot be assigned to the same machine (see Constraints 4.2) as they would take up more storage resources without providing any benefit in terms of reliability. $\forall i \in \{1, \ldots, |\mathcal{F}|\}$, $\forall j \in \{1, \ldots, |\mathcal{B}_i|\}$, $\forall u \in \{1, \ldots, |\mathcal{M}|\}$:

$$\sum_{k=1}^{\rho_i} x_{i,j,k}^{u} \leq 1 \tag{4.3}$$

4.2.2.7 Spread Constraints

Big Data clusters are composed of various racks and for reliability and security reasons, it is good practice to spread replicas $b_{i,j,k}$ of the same block $b_{i,j} \in \mathcal{B}_i$ over at least σ_i racks (see Constraints 4.4). $\forall i \in \{1, \ldots, |\mathcal{F}|\}$, $\forall j \in \{1, \ldots, |\mathcal{B}_i|\}$:

$$\sum_{v=1}^{|\mathcal{R}|} \left[\min \left(1, \sum_{\substack{u \in \{1,\ldots,|\mathcal{M}|\} \\ \wedge MR(m_u)=r_v}} \sum_{k=1}^{\rho_i} x_{i,j,k}^{u} \right) \right] \geq \sigma_i \tag{4.4}$$

4.2.2.8 Proposed Algorithm

For the data placement problem, we propose an algorithm which aims at optimising: (i) the number of external tasks by spreading the block replicas on the different racks (increasing the chance of having a replica of the needed block in the same rack), and (ii) the number of rack task by putting as many blocks of the same file on the P (e.g., 30%) percent most powerful machines in terms of CPU cores.

The algorithm operation is based on preference ordering among the racks and the nodes in the cluster. In this case, for every replica of every block of the "hot data" files, a rack and a node are chosen based on the ordering preference, and the number of already stored blocks of that particular file as described in Algorithm 2. The first metric implemented in the simulator is the number of virtual cores (processing slots) available in the node, determined by the number of physical cores available in the processor. The algorithm selects the most appropriate node in each rack using the function described in Algorithm 3, that returns the least used node with more cores on a particular rack. With more cores available, the chance of executing a task locally

increases, if the block is stored in the same node. In that manner, the algorithm attracts the blocks of the more used files to the nodes with more resources. The objective is to decrease the network usage and improve the overall performance of the jobs.

Algorithm 2: Guided placement of hot files

input :$\mathcal{HF}$: Set<File>, $\mathcal{M}$: Set<Machine>, $\mathcal{R}$: Set<Rack>, P: Percentage
output $Pacement$: Matrix< $\mathbb{N}$ >
:
$L \leftarrow zeros(|\mathcal{M}|)$ // Load per machine
for $f_i \in \mathcal{HF}$ **do**
  $currentRack \leftarrow 0$
  $BC \leftarrow zeros(|\mathcal{M}|)$ // block count per file in a machine
  $B_i \leftarrow getBlocks(f_i)$
  for $b_{ij} \in \mathcal{B}_i$ **do**
    for $k \in \{1, 2, ..., \rho_i\}$ **do**
      // get index of 'P' most powerful machines in current rack
      $indexPMs \leftarrow getPMs(\mathcal{M}, currentRack, P)$
      // get index of best machine
      $indexBest \leftarrow bestPM(indexPMs, b_{i,j,k}, BC, L)$
      if $indexBest = -1$ **then**
        // get index of feasible machine randomly from current or next racks
        $indexBest \leftarrow random(\mathcal{M}, currentRack, L)$
      $Placement[i][j][k] \leftarrow indexBest$
      $L[indexBest] \leftarrow L[indexBest] + Q_{b_{i,j},storage}$
      $BC[indexBest] \leftarrow BC[indexBest] + 1$
      $currentRack \leftarrow (currentRack + 1)\ \%\ |\mathcal{R}|$
  return *Placement*

4.2.3 Experimental Setup

The main goal of this phase of the research is to analyze how different data placement algorithms help in decreasing the network usage in concurrent big data jobs, therefore minimising the network delay and decreasing the overall time to complete the jobs.

For the experiment, 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 (Chen et al., 2012), 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 (Chen et al., 2012), this is the common situation, and our algorithms are designed to improve the performance and decrease the delay for this type of scenario.

In order to test the different data placement strategies and the different network topologies, we need to simulate clusters with features similar to the ones found in real clusters.

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

Algorithm 3: Get Index of Best Powerful Machine

input : $indexPMs$: Set< $\mathcal{N}$ >, $b_{i,j,k}$: Block, BC: List< $\mathbb{N}$ >, L: List< $\mathbb{R}$ >
output $bestMachine$: $\mathbb{N}$
:
$bestMachine \leftarrow -1$ // index of best machine
$availableCores \leftarrow 0$ // number of free CPU cores in the best machine
for $s \in indexPMs$ **do**
  if $Q_{b_{i,j},storage} + L[s] > Q_{m_s,storage}$ **then**
    if $Q_{m_s,CPU} - BC[s] > availableCores$ **then**
      $bestMachine \leftarrow s$
      $availableCores \leftarrow Q_{m_s,CPU} - BC[s]$
return $bestMachine$

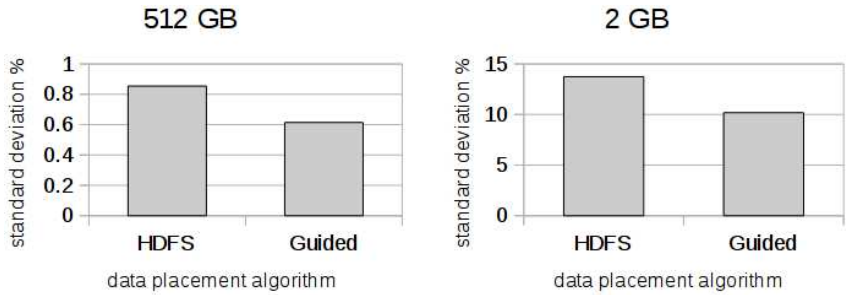


Figure 4.13: Node Storage Usage Standard Deviation.

data. In next phases of the research, these values will be part of other strategies, being modified as well.

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 (IETF, 1992).

Three sizes of clusters were used in the tests:

- 504 nodes (24 racks with 21 nodes each)
- 1024 nodes (32 racks with 32 nodes each)
- 2016 nodes (56 racks with 36 nodes 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 on Ethernet switches.

The maximum number of concurrent jobs were 128 for the tests, and scaled in 1, 32, 64, 96 and 128 for the network delay. All the tests start with 1 job and go up to the maximum, showing the progressive state of the cluster and the network. All the results are average values from 30 consecutive runs, to avoid bias due to the random nature of some of the algorithms used.

The dataset size used in the experiments is calculated to use all the available resources in the cluster, being the block size multiplied by the number of available processing slots. In that way, all the cluster resources are used in all the tests, with resulting dataset sizes of approximately 0.5 GB (504 nodes), 1 TB (1024 nodes) and 2 TB (2016 nodes).

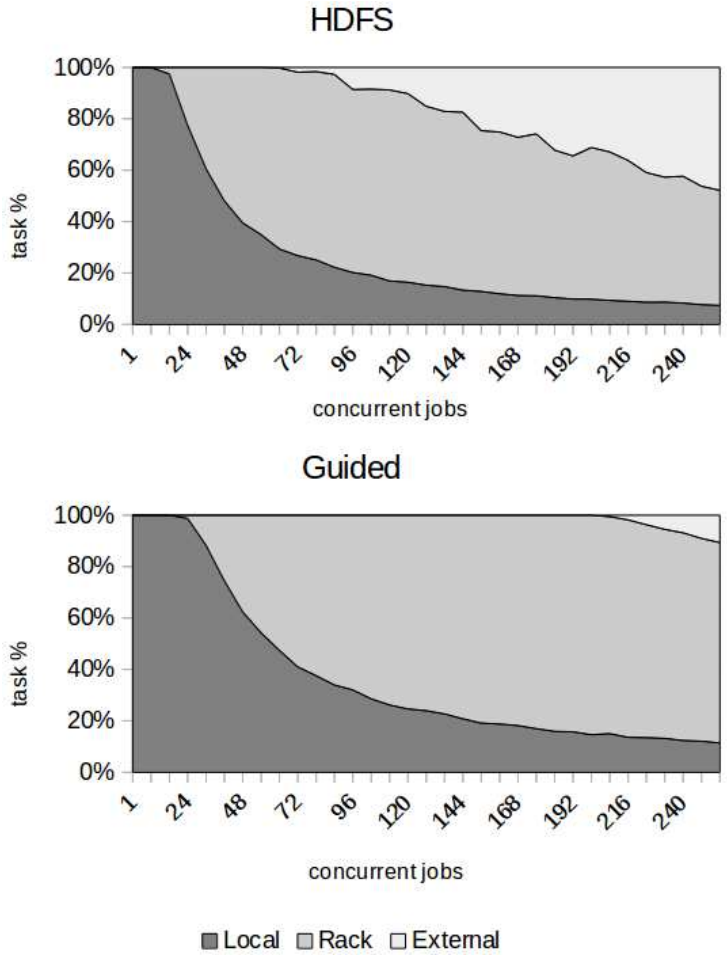


Figure 4.14: Distribution of Local, In-rack and External tasks.

For comparison, a typical data set from TPC-H (Group, 2010), used in tests in big data and database systems, in an input file of 5.2 GB, we have about 23.3 million records for the table Customers, and 41 million for the table LineItem. Those are also sizes usually found in big data jobs for data warehouse scenarios (Chen et al., 2012).

4.2.4 Experiment Results

The results are focused on the network delay metrics, and in the ratio of Local, in-Rack and External tasks, during an execution. An analysis of the block distribution among the cluster nodes was also conducted, to ensure that the implemented data placement algorithms maintain the balance between the storage usage. Figure 4.13 shows that the standard deviation of the used storage in all the nodes in the cluster is decreased using the implemented algorithm when compared to the standard HDFS. This ensures that the balance in the cluster is maintained, reducing deviations in the results.

As the objective of this research is to increase the job performance by decreasing the network utilisation, the number of tasks executed locally, in rack and remotely is an important metric, considering that when nodes need to use a network communication channel, the delay involved increases the execution time, and this is even bigger when the network communication occurs across several switches (in the external case), and when several concurrent users are disputing the communication medium.

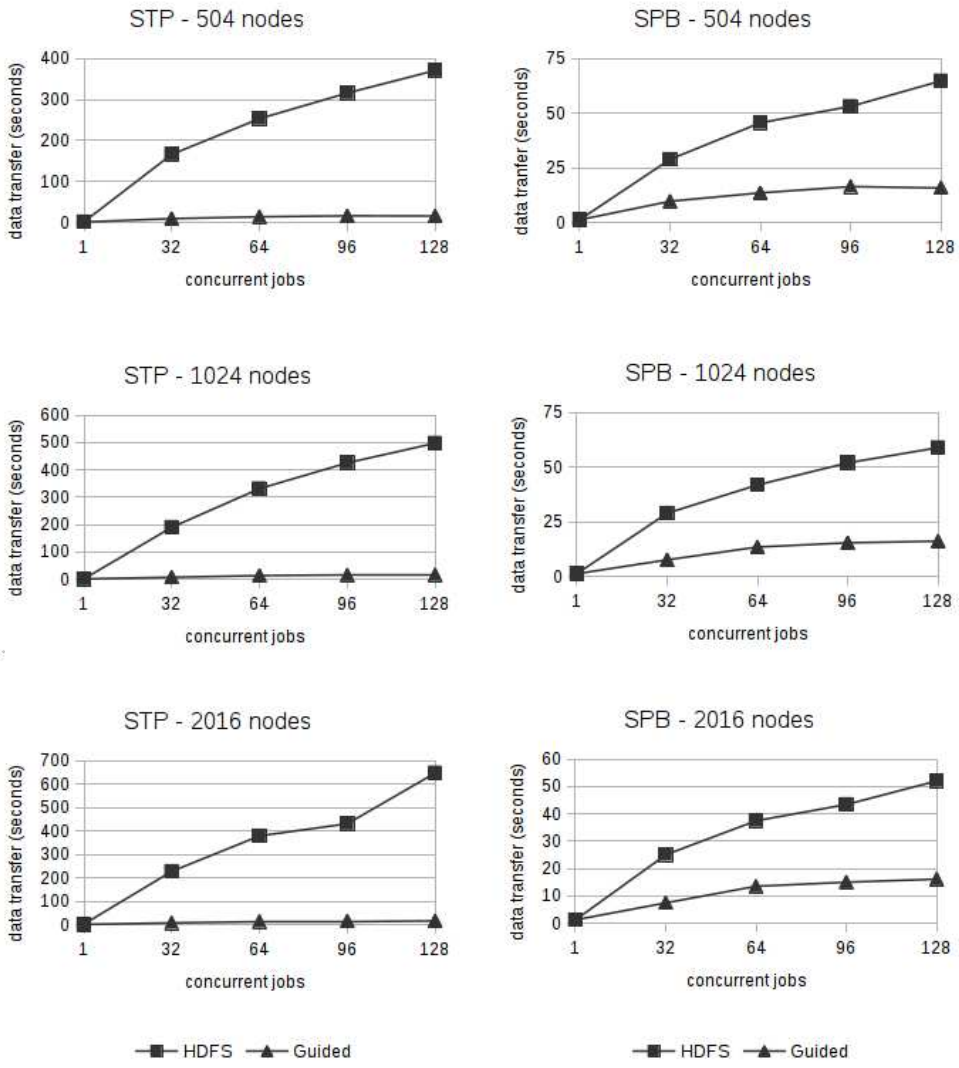


Figure 4.15: Estimated data transfer.

In Figure 4.14 is shown the proportion of local, in-rack and external tasks, related to where the tasks read the data to process. In the Guided algorithm, the number of local tasks is increased by 34.88% compared with HDFS. At the same time, the external tasks number has decreased by 798.10% compared with HDFS. This particular chart shows the results in a 1024 node cluster with 256 concurrent jobs accessing the same data. This metric is not modified by the link layer protocol used.

Based on the number of external connections, the chosen link layer protocol, and the details of the underlying network connections and equipment, the simulator can estimate the average data transfer times for the submitted jobs. This is the most important metric of this section, and serves as a basis for future development.

In this experiment, the results show that we have significant improvement in performance using both link layer protocols and using the three tested cluster sizes. Figure 4.15 shows the estimated data transfer time in all the tested configurations, with both link layer protocols. In the results, our algorithm achieved an average of 2.6 times lower data transfer time in SPB link layer protocol.

4.2.5 Data Placement Summary

In a general analysis, the proposed data placement algorithm showed gains in diminishing the network delay, by reducing the time to transfer the data in the jobs.

Regarding the link layer protocol analysis, it is clear that STP is not well suited for large arrangements of computers in an hierarchical network. Particularly when several concurrent jobs are accessing the same portion of data, when a competitive condition happens in part of the links in the network, impairing the overall network performance. In SPB, although the performance also suffers with concurrent jobs, the losses are not so severe.

The results show a significant difference between standard HDFS and the proposed data placement algorithm. This is partially due to the fact that for the default replication factor of 3, HDFS keeps 2/3 of the blocks in just one rack, impairing the even distribution of blocks in the cluster, and increasing the chance of external reading tasks, with more concurrent users. With increased replication factor, this effect tends to diminish, and this will be the subject of later studies.

The study shows that the proposed algorithm is more efficient as the number of concurrent jobs using the same data increases, with 128 concurrent jobs, the data transfer performance is improved by an average of 2.6 times, in SPB. By extension, shows that when few concurrent jobs are in place, there are no significant differences. The HDFS standard algorithm can be used in this case without penalty in performance.

It is also clear that the data placement strategy has limits regarding the cluster organisation and the number of concurrent jobs. In some conditions, the gains start to degrade after a number of concurrent jobs, causing an increased number of external tasks, due to a lower number of available processing slots in each rack. This decrease in gain is noted only with a higher number of concurrent jobs.

As the research continues, we have several additional questions to answer, regarding the data organisation and the network architecture.

This study only considered the default replication factor of 3, and although it is intuitive to think that increasing the replication factor could also increase the data locality, it could be interesting to study the levels of performance improvement, especially regarding the decrease in network delay due to more local tasks in each job. This can be even greater when the information is used by several concurrent jobs.

The network architecture also plays an important role, as demonstrated by the differences between SPB and STP link layer protocols in the spine-leaf physical topology. In the continuation of this research, other protocols and topologies will be tested, in order to study the correlation between the distributed file system features and the network features.

Although in real cases the number of racks and the number of nodes per rack are decided on a resource availability basis, some results showed that there can exist a correlation between those and the estimated network delay, due to how the link layer protocol responds to the network organisation. The cluster organisation will be probably always dictated by the resource availability (number of ports in switches, number of available connections in data centres, etc), however, the continuing research can at least suggest better cluster organisations based on this available resources.

Finally, the correlation between these several factors can be used as a surrogate function in an optimisation algorithm, analysing the appropriate strategy for a particular file (or set of files), considering the size of the cluster, the data placement algorithm, and network information.

4.3 FINAL REMARKS

Both contributions presented in this chapter are relevant for the main goal of the research, and represent steps towards a more complex and efficient optimisation in Big Data frameworks. The concept of "data locality" was demonstrated as important for Big Data systems, specially in distributed file systems as HDFS, and a novel data placement algorithm explored this to achieve more performance in the reading phase of big data jobs. In the following chapter, these contributions will be united with network information in order to provide a better understanding of how the big data applications impact the network traffic and how the network configuration can properly react to that.

5 NETWORK TOPOLOGY APPLIED TO BIG DATA SYSTEMS

This chapter shows the two main contributions of this research, a novel network topology designed specifically for Big Data system clusters in data centers, and the use of our data placement algorithm (described in Section 4.2) over this network topology. With this contribution, we aim to answer the research question, and contribute to improve the performance of Big Data systems in general.

5.1 NETWORK TOPOLOGY APPLIED TO BIG DATA SYSTEMS

5.1.1 Big Data Systems and Networks

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 a 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 base, 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.

In these huge systems, there is a constant pursuit for performance, since the execution time for jobs can take several hours even days (Villanustre, 2014). In this environment, even a minor improvement in performance can save several hours of execution time. The energy consumption and cooling costs are also directly influenced by CPU processing required for running the jobs (Hammadi and Mhamdi, 2014). Although Hadoop tries to run jobs locally (i.e., reading the data from local storage in each node), 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.

To address this issue, we have developed a novel network topology, based on a ring-mesh architecture, 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. We propose this new network topology and a forwarding algorithm for decreasing the number of network equipment used, and increasing the network performance for Big Data job execution. This will result in reduced costs in equipment acquisition and maintenance, and lower costs in energy consumption and cooling. The switches used in this topology must be SDN-enabled (i.e., use the OpenFlow protocol) and are configured by a SDN controller. The MLM network topology was published in the ICC 2019 (International Conference of Communications) (de Almeida et al., 2019).

5.1.2 The Multi-Layer-Mesh Topology

The main contribution of this section 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

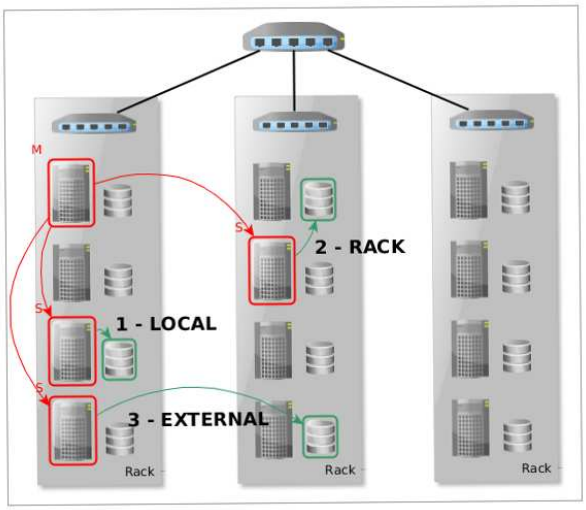


Figure 5.1: Example of local (1), same rack (2) and external (3) tasks.

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.

5.1.2.1 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 (Villanustre, 2014). 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 (Shvachko et al., 2010), 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 5.1, 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 an External task (reading from another node in another rack), with the increased delays for each step.

Then, 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 (Chen et al., 2016), 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.

5.1.2.2 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 5.2 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).

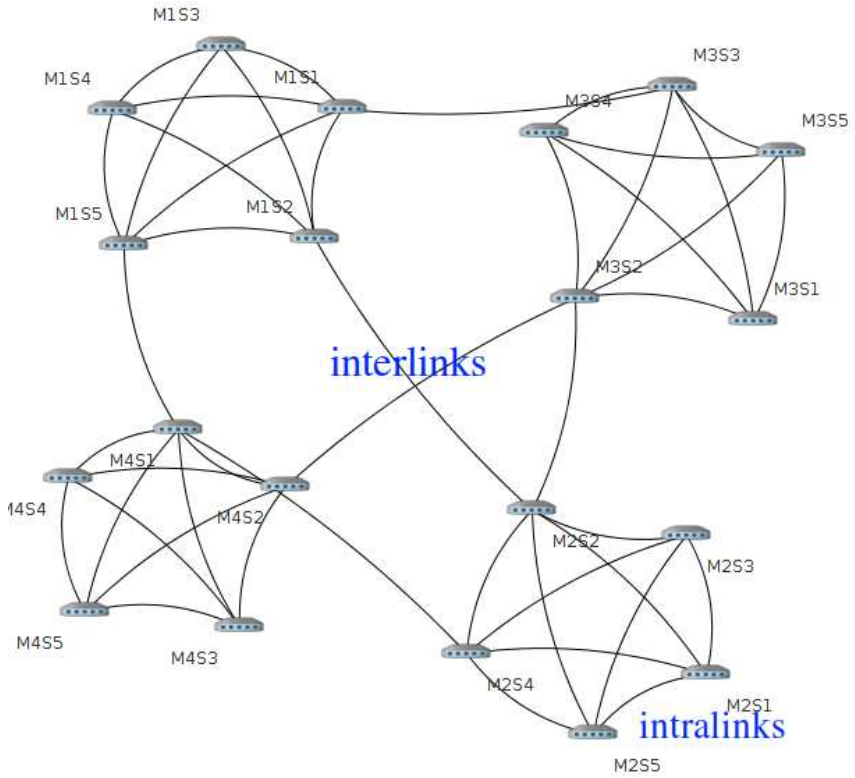


Figure 5.2: Example of a Multi-Layer-Mesh topology with full L1 meshes interconnected by interlinks.

As links are cheaper than equipment - considering copper connections - this topology reduces the amount of used resources, while keeping the traffic 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 5.3.

5.1.2.3 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 to 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.

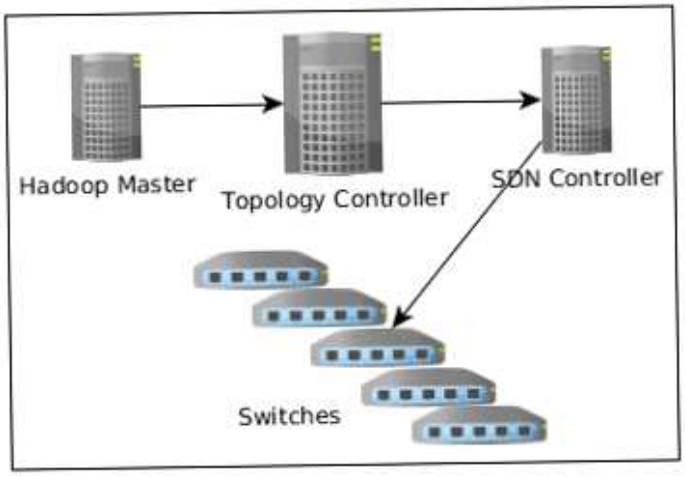


Figure 5.3: System architecture overview.

- **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 the 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.

- **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 can take some time to happen, the start of a Hadoop task is also time consuming (considering copying executable files to the nodes, allocating cpu slots and so on), 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 4. As the local and rack tasks are treated by the same switch or no switch at all, the algorithm focuses 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 the 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.

5.1.3 Experiments

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

Algorithm 4: 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
```

5.1.3.1 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 (Chen et al., 2012), 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 (Chen et al., 2012), 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. We also considered that several Ethernet switches can be interconnected, increasing the number of available ports for what the topology needs.

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 equipment 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.

5.1.3.2 Results

The results are shown in Figures 5.4, 5.5 and 5.6 for the three cluster configurations. The plots show the comparison between 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 5.7, 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. 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.

In Figure 5.8, the number of used switches and links is shown. 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 showed divided by ten, for comparison purposes. It's possible to observe that MLM uses less switches than SL, but more links in the network. As the cost of links are usually cheap (especially those of copper, as depicted in this research), the overall costs are lower in MLM.

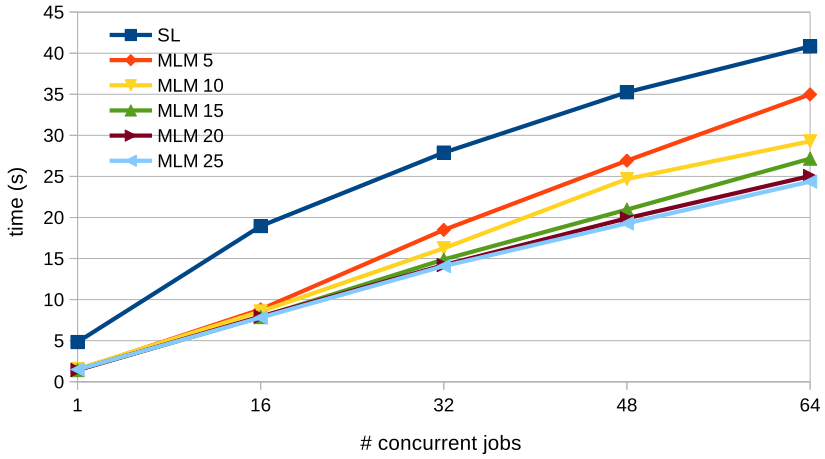


Figure 5.4: Jobs completion time (500-node cluster).

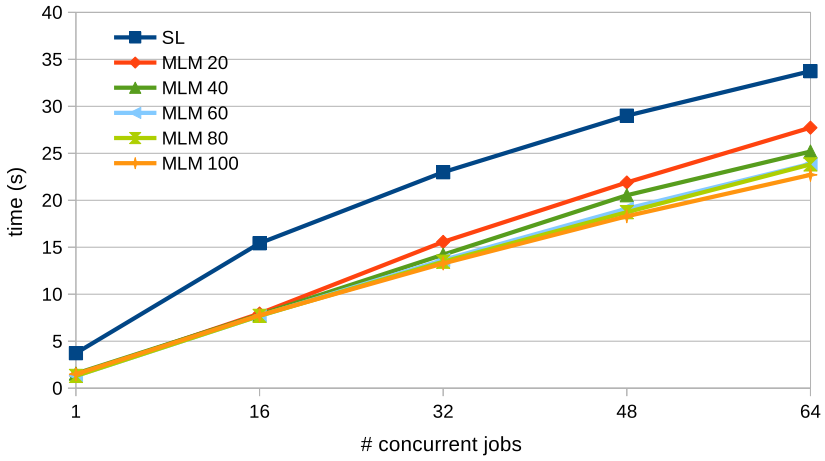


Figure 5.5: Jobs completion time (1000-node cluster).

The estimated monetary costs are shown in Figure 5.9. 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 5.7. 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 5.10, 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.

In the cost analysis, the cost of the SDN controller was not considered, due to the large variation in models and values, and because the network infrastructure will probably have a software controller of some sort, for network analysis and statistics gathering.

[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

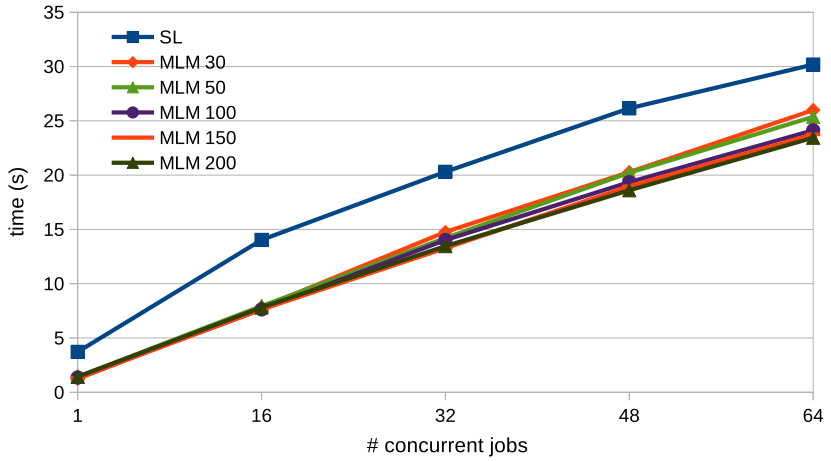


Figure 5.6: Jobs completion time (2000-node cluster).

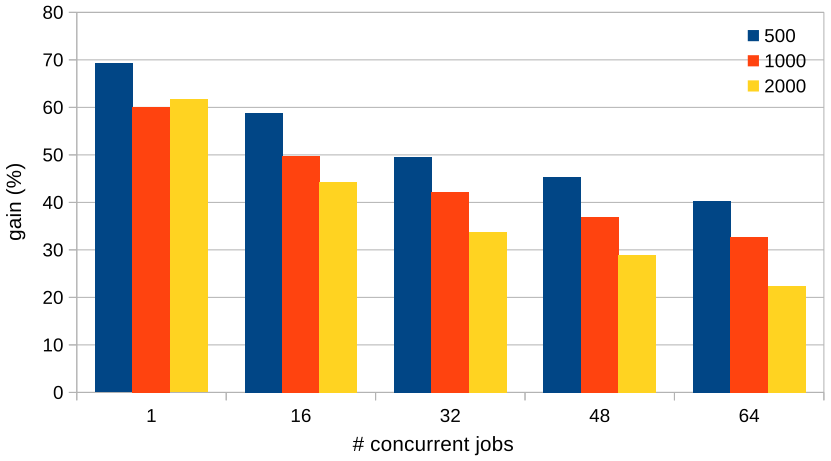


Figure 5.7: Performance gain of the MLM topology over the Spine-Leaf topology.

5.1.4 Considerations about the experiments

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 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 equipment are not needed anymore. Based on the data from a survey (Hammadi and Mhamdi, 2014), 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.

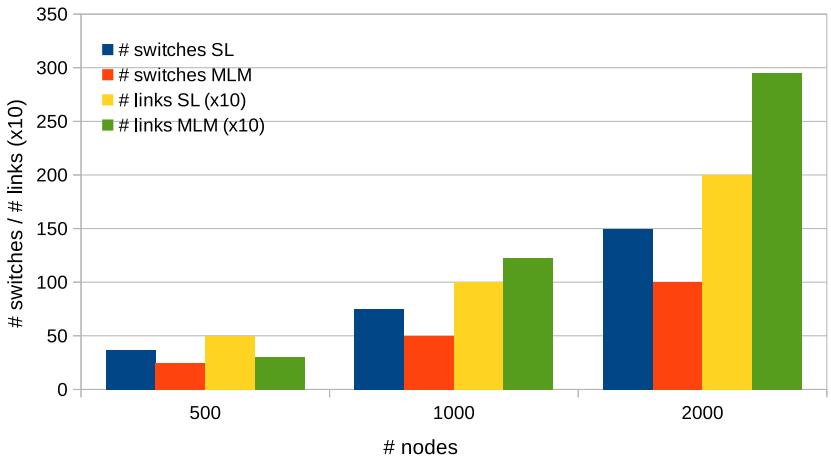


Figure 5.8: Number of switches and links *vs* cluster topology and size.

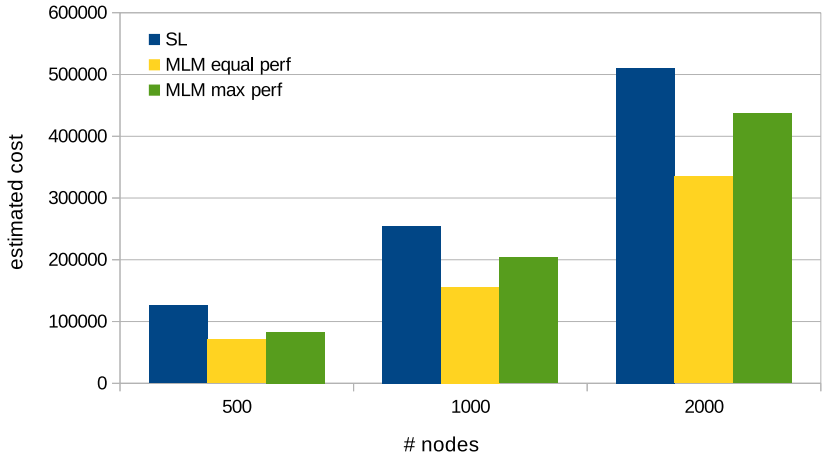


Figure 5.9: Estimated cost (€) *vs* cluster topology and size.

5.2 DATA PLACEMENT STRATEGIES USING NOVEL NETWORK TOPOLOGIES

In Section 4.2 we presented a new data placement algorithm that benefits from the additional available processing slots in nodes distributed in the cluster, considering an heterogeneous set of nodes, with part of the nodes having more cores than the rest. This algorithm will then be implemented on top of the Multi-Layer-Mesh network topology described in the previous section, to observe how the overall performance will be affected by the combined solution. This will increase the perceived performance of HDFS, and as all the components of a regular Hadoop stack are using HDFS to read and write information, any improvement done in its performance could have a positive effect in the rest of the stack. The integration between the Multi-Layer-Mesh network topology and this new data placement algorithm will be shown in the next section.

5.2.1 Objective

We aim at minimizing the execution time of multiple Hadoop queries, on a given network topology and a given link layer protocol. We believe that reducing external and rack tasks (i.e., tasks requiring data from a machine different from the one they run on) can help reach the objective, especially when this information is combined with the underlying network architecture. This is motivated by the fact that execution nodes waste CPU computation time while waiting for

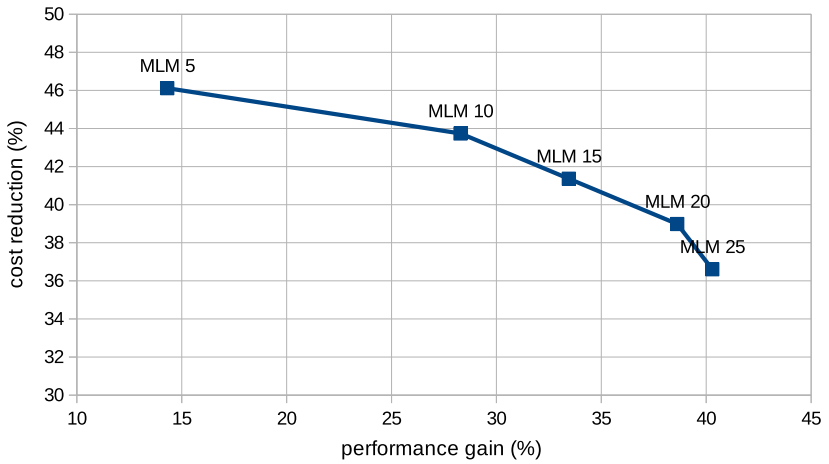


Figure 5.10: Cost *vs* performance (500-node cluster with 64 concurrent jobs).

data transfer over the network, and in some topologies, when several concurrent jobs are using the same portion of data in the cluster, the bottlenecks are more severe.

Trying to keep most of the reading tasks accessing storage can significantly reduce the network traffic and bottlenecks, and in this section, we study the network behavior related to the most common link layer protocols (STP and SPB), over a Spine-Leaf physical topology.

5.2.2 Experiments - Including Data Placement in the Routing Algorithm

The main objective of this research work is to test how the novel data placement algorithm running over a Multi-Layer-Mesh network topology affects the performance of the reading part of big data jobs. In this sense, we designed the experiments to reproduce the conditions of the previous experiment, with just the Multi-Layer-Mesh topology (de Almeida et al., 2019), adding the novel data placement algorithm, and comparing with the previous results, as well as the regular Spine-Leaf topology.

As indicated before, most of the time it is not feasible to set up a real cluster with thousands of nodes for an experiment. This is the reason why this research uses a simulator to obtain the desired values for the network topology. Using our simulator, three cluster sizes were tested, using a configuration commonly found in real clusters, and already tested for the Multi-Layer-Mesh topology.

5.2.2.1 Experimental Setup

In the experiments, we used the simulator to assemble three different cluster configurations, and executed the simulations by considering the file distribution and the 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 (Chen et al., 2012), 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 (Chen et al., 2012), 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. As the previous executed experiments, we 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.

In the same sense, the network configuration was kept in the same configuration, using a Gigabit Ethernet, with delays of 1 ms per switch, frame size of 1518 bytes and again 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.

The same 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.

Again, as the previous experiment, the number of concurrent was kept the same, ranging from 1 to 64, in intervals of 16 (1, 16, 32, 48 and 64). 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. Even in that case, our proposed topology shows performance improvement. Lower number of concurrent jobs were not tested, as in this scenario, most of the jobs are solved locally, without needing to read data through the network, being not the ideal use case for our research.

For the configuration of the network topologies, we considered the same conditions as the previous test, that is, in the Spine-Leaf topology, the number of switches in the spine is set to half of the number of available racks, and in the Multi-Layer-Mesh topology, the number of switches is equal to the number of access switches used in the Spine-Leaf topology. In the Multi-Layer-Mesh topology, the connections from the Top of Rack (ToR) switches to the internal ring starts in 2, and go up to the number of switches in the ring, and the number of internal connections in the ring starts in 0, and goes up to a total mesh.

To avoid any bias due to the usage of randomization in the simulation configuration, the results are average values computed from the values produced by 100 simulation runs, as in the previous experiments.

In the same sense, 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.

For comparison, a typical data set from TPC-H, (Group, 2010), used in tests in big data and database systems, is an input file of 5.2 GB, with about 23.3 million records for the table "Customers", and 41 million for the table "LineItem". Those are also sizes usually found in big data jobs for data warehouse scenarios (Chen et al., 2012).

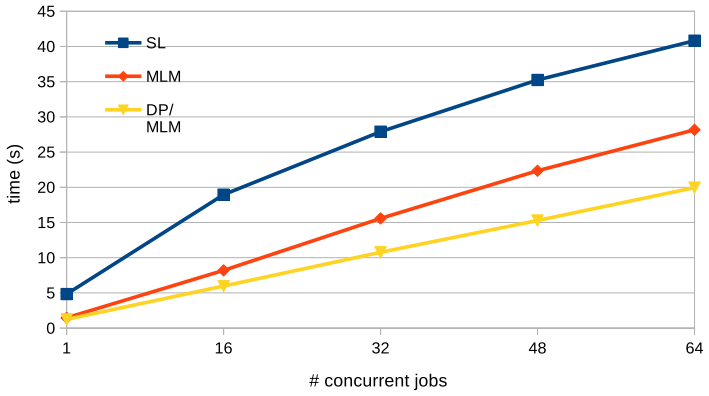


Figure 5.11: Jobs completion time DP/MLM (500-node cluster).

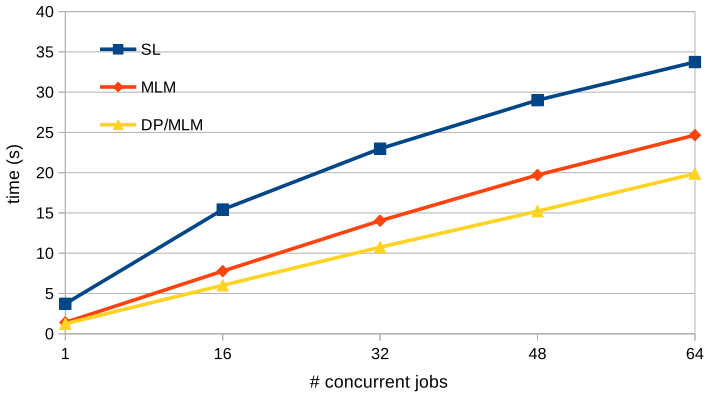


Figure 5.12: Jobs completion time DP/MLM (1000-node cluster).

5.2.2.2 Results

The test results compared the performance of the Multi-Layer-Mesh topology with the Data Placement strategy using the MLM as a transport layer, and with a regular Spine-Leef network topology.

The comparison used the average time of simulations with 1, 16, 32, 48 and 64 concurrent jobs, using the same number of interlinks as the results of Multi-Layer-Mesh topology. The results are shown in figures 5.11, 5.12 and 5.13 for the 500, 1000 and 2000 node cluster configurations, respectively. In the 500 nodes results, the chart shows the average results of MLM with 5, 10, 15, 20 and 25 interlinks, in the 1000 nodes results, the average is of 20, 40, 60, 80 and 100 interlinks, and in the 2000 cluster, the average shows the average of 30, 50, 100, 150 and 200 interlinks. The interlinks level was chosen to match the results on the Multi-Layer-Mesh topology already published, and to better show the increase in performance of the overall topology.

As in the MLM topology, the observed gain in performance is related to the number of concurrent jobs and number of interlinks. In Figure 5.14, we can observe the performance gains of the Data Placement using MLM when compared with a Spine-Leaf baseline, and in Figure 5.15 we can observe the performance gain compared with a MLM topology, using the maximum number of interlinks in all cases.

As shown in the plots, we can observe a performance gain in the 3 cluster configurations when compared to a Spine-Leaf topology. For the 500 node cluster, this gain is 51.11% in the 64 concurrent tasks case, using 25 interlinks, compared with 40.29% using just the MLM topology. The same trend occurs in the 1000 and 2000 node clusters, with a 41.08% gain in the 1000 node

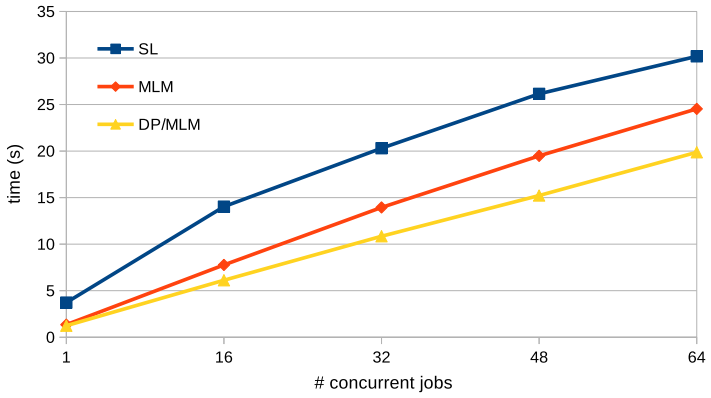


Figure 5.13: Jobs completion time DP/MLM (2000-node cluster).

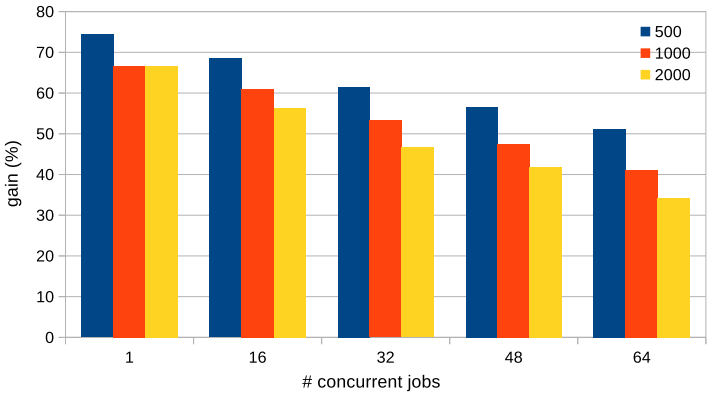


Figure 5.14: Performance gain of the DP/MLM over the Spine-Leaf topology.

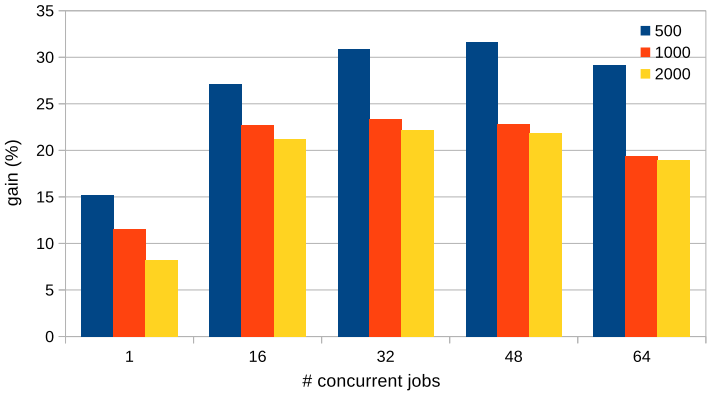


Figure 5.15: Performance gain of the DP/MLM over the MLM topology.

case, and 34.16% in the 2000 node case. Overall, the average gain is 42% for the 64 concurrent tasks case.

Compared with the regular HDFS data placement algorithm using MLM topology, the Data Placement over MLM shows an average gain of 22.5% in the 64 concurrent tasks case, as shown in Figure 5.15.

As in the results from MLM, it's worth noting that there is a significant gain for just one running task, following the case that our proposed topology can be useful for this particular use case.

Regarding the resource reduction, there is no contribution of the Data Placement, as the network infrastructure is the same, showing the same results as Figure 5.8 and Figure 5.9 in the previous section, for the number of switches and estimated costs.

5.2.3 Considerations about the results

As shown previously in Section 5.1, our novel Multi-Layer-Mesh topology and SDN-based path forwarding protocol can increase the performance of Big Data systems and reduce the equipment usage in those systems as well.

When combined with a novel Data Placement algorithm, the MLM topology shows an increased level of performance improvement, with an average gain of 42% over the regular HDFS data placement algorithm using a Spine-Leaf network topology and 22.5% over the MLM topology, for the selected cases.

The MLM topology then can help in finding novel data placement strategies with an increased performance, not interfering negatively in its performance, on the contrary, improving it.

This shows how the MLM topology can help in increasing the performance of different data placement strategies running over an HDFS infrastructure. It is interesting to note that the presented data placement has limitations regarding the eventual reorganization of data blocks in HDFS due to the possible failure of nodes, when the data blocks need to be replicated again in different nodes. This can be addressed by a change in the replacement strategy in HDFS, to be conducted in future research works.

5.3 FINAL REMARKS

This chapter presented the main contributions of this research work, a novel network topology designed for Big Data systems in data centers, and the use of a novel data placement algorithm in collaboration with this network topology in order to increase the performance of those systems. The Multi-Layer-Mesh topology increases the performance by an average of 31.77% and when combined with the novel data placement algorithm, the increase in performance achieved 42%.

The novel network topology is also useful to reduce equipment usage in the network infrastructure and by extension, reduce the energy consumption as well.

The results show that the network topology can be a good base infrastructure for further improvements in Big Data systems performance, that can benefit from network strategies linked with data movement. The implemented data placement algorithm shows how this combination can be beneficial.

6 CONCLUSION AND FUTURE CONTRIBUTIONS

Based on the previous contributions (Big Data Network Simulator, Data Placement and Network Topologies for Big Data), this research has brought together these concepts under a new framework, in order to provide better performance and less resource utilisation in Big Data clusters. As stated in the Introduction, our main research question is if we can improve the performance of Big Data systems by modifying how the data will be moved and placed in the cluster according to the usage of data by the system. Considering that using our Multi-Layer-Mesh network topology, and executing our data placement algorithm over the topology, we achieved an increase in performance, we can then answer positively this question, as described below.

6.1 IMPROVING PERFORMANCE IN BIG DATA SYSTEMS

We started exploring the concept of "data locality", as shown in Chapter 4, executing experiments about the impact of block replication in HDFS using a physical cluster with 20 nodes, with published results at the ACM/SPEC ICPE International Conference on Performance Engineering (Ciritoglu et al., 2018). These experiments showed that when the data blocks are made more available in the cluster, the tasks have a higher probability of reading it locally, thus avoiding network transport and a possible bottleneck. However, there are obvious limitations to this strategy as it is not practical to replicate a high number of blocks around the cluster due to the storage usage and even the resulting network traffic. Even then, the importance of data locality was established.

With these results,we intended to test this in a bigger scale, on clusters with hundreds or thousands of nodes. Due to the practical difficulties on testing on a infra-structure this big, and considering the lack of network and data movement from available simulators, we then implemented our own Big Data system simulator, BigDataNetSim (De Almeida et al., 2018), focusing on the information we needed, as described in Chapter 3. The BigDataNetSim was published at the IEEE/ACM DS-RT, International Symposium on Distributed Simulation and Real Time Applications.

Using the simulator, we developed and tested a data placement algorithm, considering how the data locality can impact the performance of concurrent jobs executing in a Big Data system, as described in Chapter 4.

Then, to continue to explore our hypothesis, we designed a novel network topology, as shown in Chapter 5. Our Multi-Layer-Mesh (de Almeida et al., 2019) topology (and SDN-based path forwarding protocol) increased the performance by an average of 31.77% (from 22.32% up to 40.29%) when using the maximum number of interlinks. This network topology can also reduce the network infrastructure costs by an average of 36.03%, if the same level of performance is maintained. The results of Multi-Layer-Mesh were published at the IEEE ICC, International Conference on Communications. The reduction in usage of network equipment also means reduction in energy consumption from the switches and related cooling systems. As network devices are responsible for 20%–30% of the energy consumption of a data center (Hammadi and Mhamdi, 2014) and for each watt consumed by a network device or a server, one watt is consumed for cooling the same equipment, there is a perceptible gain by using this network topology.

Finally, we tested our data placement algorithm over the Multi-Layer-Mesh topology, to determine how the topology drives performance improvements in Big Data systems performance,

when combined with other strategies. The data placement strategy was selected, due to the previous studies and the results based on the data locality concept.

When used over our Multi-Layer-Mesh network topology, our novel data placement algorithm outperforms the regular HDFS data placement algorithm by an average gain of 42%, regarding its limitations, as shown in Chapter 5. This illustrates how the Multi-Layer-Mesh topology can therefore help in increasing the performance of various data placement strategies running over an HDFS infrastructure, opening the possibilities for new research works in this topic.

As this research is based on pragmatic issues, it is important to highlight the public potential that can benefit from the contributions shown here. The various companies and corporations that use HDFS-based Big Data systems can directly benefit from the contributions presented here. In addition, the Multi-Layer-Mesh network topology can be used in other systems that are supported by clusters with high network traffic, thus covering data center companies and cloud service providers, which keep a large number of nodes connected to each other.

In conclusion, the aggregated results of the research allow the affirmative answer to the research question of the thesis, as mentioned at the beginning of this chapter, as well as having a potential public use in various technology sectors.

6.2 FUTURE CONTRIBUTIONS AND DEVELOPMENTS

As demonstrated in the Literature Review chapter, there are several opportunities to increase the performance of Big Data systems using network and data placement techniques. There are several techniques to learn the best paths or the best precedence queues, and then apply this knowledge to the SDN switches. Regarding future work, we plan to improve the research in some areas, as follows.

As this research was conducted assuming that regular switches were used, we can adapt the topology and the simulator to evaluate optical switches and connections as well. We can also improve the way the path/traffic analysis is executed, allowing the controller to react to changes in the network during the jobs' execution. The network-accelerated query processing is also an area of possible future work, as the improvements in data and task placement strategies.

REFERENCES

Hadoop distributed file system.

Hdfs architecture.

Hdfs erasure coding.

Apache (2013). Yarn scheduler load simulator.

Ardagna, D., Barbierato, E., Evangelinou, A., Gianniti, E., Gribaudo, M., Pinto, T. B. M., Guimarães, A., Couto da Silva, A. P., and Almeida, J. M. (2018). Performance prediction of cloud-based big data applications. In *ICPE*, pages 192–199.

Arres, B., Kabachi, N., Boussaid, O., and Bentayeb, F. (2015). Intentional data placement optimization for distributed data warehouses. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 80–86.

Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks.

Casado, M., Foster, N., and Guha, A. (2014). Abstractions for software-defined networks. *Commun. ACM*, 57(10):86–95.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 1–12, New York, NY, USA. ACM.

Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: A retrospective on evolving sdn. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 85–90, New York, NY, USA. ACM.

Cela, A., Lee, Y. C., and Hong, S.-H. (2018). Resource provisioning for memory intensive graph processing. In *Proceedings of the Australasian Computer Science Week Multiconference 2018*, ACSW '18, pages 15:1–15:7, New York, NY, USA. ACM.

Chen, T., Gao, X., and Chen, G. (2016). The features, hardware, and architectures of data center networks: A survey. *Journal of Parallel and Distributed Computing*, 96:45 – 74.

Chen, Y., Alspaugh, S., and Katz, R. (2012). Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proc. VLDB Endow.*, 5(12):1802–1813.

Ciritoglu, H. E., Batista de Almeida, L., Cunha de Almeida, E., Buda, T. S., Murphy, J., and Thorpe, C. (2018). Investigation of replication factor for performance enhancement in the hadoop distributed file system. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 135–140, New York, NY, USA. ACM.

Clusters, H. M. (2016). Hadoop mini clusters.

Cui, L., Yu, R., and Yan, Q. (2016). When big data meets software-defined networking: Sdn for big data and big data for sdn. *IEEE Network*, 30:58–65.

De Almeida, L. B., De Almeida, E. C., Murphy, J., De Grande, R. E., and Ventresque, A. (2018). Bigdatanetsim: A simulator for data and process placement in large big data platforms. In *DS-RT 2018*, pages 1150–1160.

de Almeida, L. B., Magoni, D., Perry, P., de Almeida, E. C., Murphy, J., and Ventresque, A. (2019). Multi-layer-mesh: A novel topology and sdn-based path switching for big data cluster networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Desai, A., Nagegowda, K. S., and Ninikrishna, T. (2016). An approach to efficient network design and characterization using sdn and hadoop. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pages 1–6.

Desai, A. and S, N. K. (2015). Advanced control distributed processing architecture (acdpa) using sdn and hadoop for identifying the flow characteristics and setting the quality of service(qos) in the network. In *2015 IEEE International Advance Computing Conference (IACC)*, pages 784–788.

Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A., and McPherson, J. (2011). Cohadoop: flexible data placement and its exploitation in hadoop. *Proceedings of the VLDB Endowment*, 4(9):575–585.

Filho, E. R. L., de Almeida, E. C., and Scherzinger, S. (2019). Don't tune twice: Reusing tuning setups for sql-on-hadoop queries. In *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*, pages 93–107.

Ghalwash, H. and Huang, C. (2017a). Software-defined extreme scale networks for bigdata applications. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7.

Ghalwash, H. and Huang, C. H. (2017b). Software-defined extreme scale networks for bigdata applications. In *IEEE High Performance Extreme Computing Conference*, pages 1–7.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43.

Giraph, A. (2012). Apache giraph.

Group, T. (2010). Tpc-h decision support benchmark.

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

Hadoop, A. (2008). Hadoop documentation.

Hammadi, A. and Mhamdi, L. (2014). A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1 – 21.

Hammoud, S., Li, M., Liu, Y., Alham, N. K., and Liu, Z. (2010). Mrsim: A discrete event based mapreduce simulator. In *7th Int'l Conf. on Fuzzy Systems and Knowledge Discovery*, pages 2993–2997.

He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., and Xu, Z. (2011). Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *IEEE 27th International Conference on Data Engineering*, pages 1199–1208.

Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. (2011). Starfish: a self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272.

IETF (1992). Rfc 1323.

Jin, H., Yang, X., Sun, X.-H., and Raicu, I. (2012). Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 516–525. IEEE.

Jung, J. and Kim, H. (2012). Mr-cloudsim: Designing and implementing mapreduce computing model on cloudsim. In *2012 International Conference on ICT Convergence (ICTC)*, pages 504–509.

Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. (2009). The nature of data center traffic: Measurements & analysis. In *9th ACM Internet Measurement Conference*, pages 202–208.

Kolberg, W., Marcos, P. D. B., Anjos, J. C. S., Miyazaki, A. K. S., Geyer, C. R., and Arantes, L. B. (2013). Mrsg - a mapreduce simulator over simgrid. *Parallel Comput.*, 39(4-5):233–244.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Latah, M. and Toker, L. (2018). Artificial intelligence enabled software defined networking: A comprehensive overview. *CoRR*, abs/1803.06818.

Lin, C. and Liao, J. (2015). An sdn app for hadoop clusters. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 458–461.

Lin, C. Y. and Lin, Y. C. (2015). A load-balancing algorithm for hadoop distributed file system. In *18th International Conference on Network-Based Information Systems*, pages 173–179.

Lin, J.-C., Yu, I. C., Johnsen, E. B., and Lee, M.-C. (2016). Abs-yarn: A formal framework for modeling hadoop yarn clusters. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633*, pages 49–65.

Liu, Y., Li, M., Alham, N. K., and Hammoud, S. (2013). Hsim. *Future Genereration Computer Systems*, 29(1):300–308.

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: A system for large-scale graph processing. In *SIGMOD*, pages 135–146.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74.

Milo, T. (2020). Getting rid of data. *ACM Journal of Data and Information Quality*, 12(1):1:1–1:7.

Mumak, A. (2008). Apache mumak.

Narayan, S., Bailey, S., and Daga, A. (2012a). Hadoop acceleration in an openflow-based cluster. In *SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 535–538.

Narayan, S., Bailey, S., and Daga, A. (2012b). Hadoop acceleration in an openflow-based cluster. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 535–538.

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

Panda, R., Zheng, X., Gerstlauer, A., and John, L. K. (2018). Camp: Accurate modeling of core and memory locality for proxy generation of big-data applications. In *DATE*, pages 337–342. IEEE.

Plank, J., Luo, J., Schuman, C., Xu, L., and Wilcox-O'Hearn, Z. (2009). A performance evaluation and examination of open-source erasure coding libraries for storage. pages 253–265.

Porter, G. (2010). Decoupling storage and computation in hadoop with superdatanodes. *ACM SIGOPS Operating Systems Review*, 44(2):41–46.

Qazi, Z. A., Lee, J., Jin, T., Bellala, G., Arndt, M., and Noubir, G. (2013). Application-awareness in sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 487–488, New York, NY, USA. ACM.

Qiao, Y., Wang, X., Fang, G., and Lee, B. (2016). Doopnet: An emulator for network performance analysis of hadoop clusters using docker and mininet. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 784–790.

Qin, P., Dai, B., Huang, B., and Xu, G. (2017a). Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data. *IEEE Systems Journal*, 11:2337 – 2344.

Qin, P., Dai, B., Huang, B., and Xu, G. (2017b). Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data. *IEEE Systems Journal*, 11(4):2337–2344.

Randstad (2017). Five best it jobs for 2017.

Ren, Z., Liu, Z., Xu, X., Wan, J., Shi, W., and Zhou, M. (2012). Waxelephant: A realistic hadoop simulator for parameters tuning and scalability analysis. In *2012 Seventh ChinaGrid Annual Conference*, pages 9–16.

Santos, P. C., Oliveira, G. F., Tome, D. G., de Almeida, E. C., Zanata, M., and Carro., L. (2018). Hipe: Hmc instruction predication extension applied on database processing. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 261–264.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies*, pages 1–10.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. (2007). The end of an architectural era: (it's time for a complete rewrite). In *VLDB*, pages 1150–1160.

Szyperski, C., Petitclerc, M., and Barga, R. (2016). Three experts on big data engineering. *IEEE Software*, 33(2):68–72.

Teng, F., Yu, L., and Magoulès, F. (2011). Simmapreduce: A simulator for modeling mapreduce framework. In *5th FTRA International Conference on Multimedia and Ubiquitous Engineering*, pages 277–282.

Tsao, Y. C. and Hou, T. C. (2015). Bridge priority provisioning for maximizing equal cost shortest path availability. In *IEEE 16th International Conference on High Performance Switching and Routing*, pages 1–6.

Verma, A., Cherkasova, L., and Campbell, R. H. (2011). Play it again, SimMR! In *IEEE International Conference on Cluster Computing*, pages 253–261.

Villanustre, F. G. (2014). Big data trends and evolution: A human perspective. In *Proceedings of the 3rd Annual Conference on Research in Information Technology*, pages 1–2.

Wang, X., Veeraraghavan, M., Lin, Z., and Oki, E. (2017). Optical switch in the middle (osm) architecture for dcns with hadoop adaptations. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7.

White, T. (2015). *Hadoop: The Definitive Guide*. O'Reilly Books.

Xia, Y., Sun, X. S., Dzinamarira, S., Wu, D., Huang, X. S., and Ng, T. S. E. (2017). 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*, pages 295–308.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., and Qin, X. (2010). Improving mapreduce performance through data in heterogeneous hadoop clusters. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9. IEEE.

Ye, X., Huang, M., Zhu, D., and Xu, P. (2012). A novel blocks placement strategy for hadoop. In *IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 3–7.

Zhang, H., Xu, B., Yan, J., Liu, L., and Ma, H. (2016a). Proactive data placement for surveillance video processing in heterogeneous cluster. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 206–213.

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