FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Simulation of the Performance of MapReduce Jobs

Tiago André da Rocha Ferreira

WORKING VERSION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Ricardo Santos Morla

January 25, 2016

Abstract

The MapReduce distributed computation paradigm introduced by Google made a huge frenzy and with the arrival of an open source solution, Apache Hadoop framework, major companies rapidly joined this solution. Today companies such as LinkedIn and Facebook use Hadoop has the main data processing tool. In the State of the Art was found that there is a massive research in MapReduce, and that there is a lot o research to be done, and innumerous challenges ahead. This dissertation presents a solution to simplify the research and testing of algorithms for the MapReduce framework. The solution proposed is a tool for the simulation of MapReduce jobs in a cluster. The solution is based on the Apache Hadoop framework, and is highly customizable and allows the user to specify his how algorithms. The simulation environment is built on top of the grid simulator SimGrid. The SimGrid simulator is a large scale distributed systems simulator for peer-to-peer and cloud systems, that incorporates a network representation scheme. This network representation allows for a network analysis in the MapReduce jobs.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
DCN	Data Center Network
DFS	Distributed File System
FIFO	First-in First-out
FLOPS	Floating-Point Operations Per Second
GFS	Google File System
GigE	Gigabit Ethernet
HDD	Hard disk drive
HDFS	Hadoop Distributed File System
HFS	Hadoop Fair Scheduler
MB	Megabytes
MRSG	MapReduce over SimGrid
MSG	Meta-SimGrid
ToR	Top-of-Rack
VL2	Virtual Layer 2
XML	Extensible Markup Language
YARN	Yet Another Resource Manager

Chapter 1

Introduction

This chapter starts by introducing this thesis. It begins by describing its context, followed by the motivation and goals. Then it defines the methodology used to accomplish the goals set. This chapter ends by describing the structure of the dissertation.

1.1 Context

Is an engineer job to make the world an easier place to live in. This means create new solutions and instruments that make a task simpler. This is why people today have multiple devices all connected to a network. As the World goes further in the Era of Internet of Things and the number of devices with network capability increases, it also creates new challenges. Alongside this device expansion, high-end user's connection speed also increased in the past few years. According to Jakob Nielsen a high-end user's connection speed grows by 50% per year [7]. This was proved using data from the years of 1983 to 2014. All these networked devices and connectivity growth together result in huge amounts traffic and data generated. This data that needs to be processed and analysed. Companies such as Google, Facebook and Yahoo! use this data to improve their services. This analysis and knowledge extracted from this massive data sets is known as Big Data. Companies are very competitive and are always trying to differentiate themselves from the competitors, bringing innovations to their services according to user preferences and trends. This is where Big Data analysis plays an important role.

Handle Big Data is not an easy task and traditional systems are not efficient and not optimized for large data sets. In order to handle this complicated task Google developed a parallelized computation framework called MapReduce [8]. MapReduce allowed to simplify parallelization, fault tolerance, data distribution and load balancing. With MapReduce Google clusters are able to process more than 20 petabytes of information per day.

Apache Hadoop is an open source version of MapReduce, based on Google's implementation of MapReduce and Google File System, that is used by several major companies and organizations such as LinkedIn, Spotify and Twitter among others [9]. The Apache Hadoop framework is designed to work on commodity hardware, and can handle tens of thousand of machines working in parallel.

1.2 Motivation

According to Cisco [10] the global IP traffic will reach 1.1 zettabytes in 2016. Companies must be prepared to handle this massive increase in traffic. They must take decisions like upgrade or buy new hardware, or even acquire new facilities such as a new Data Center.

This upgrade may not be desirable, or may not be within company budget, as a result of this restrictions data will take longer to be processed. A possible solution is to improve the performance of MapReduce framework. This will help data to be processed quicker without hardware replacement.

In order to improve MapReduce performance new algorithms and solutions must be tested. Testing is one of the most important phases, as it can lead to success, failure or reveal new key elements to the final solution. To test new MapReduce improvements some researchers use small local clusters, with very limited resources and in small scale. Other researchers use simulation as prove of concept.

Is this dissertation the challenge is to explore the simulation of MapReduce to evaluate its performance. Simulation allows to test MapReduce in a very large scale and without hardware limitation. MapReduce performance depends on a variety of factors such as data input size, computation power, memory available and network bandwidth. Testing all these elements in a simulation environment is an enormous challenge. But it is of great importance to test these factors that have an impact on MapReduce performance.

1.3 Document Structure

The document is structured as follows. In Chapter 2 is explored the State of the Art of MapReduce. The current solutions and techniques used by MapReduce are studied here. Chapter 3 describes the current state of the adopted solution, and its current limitations, and some of the improvements. In chapter 4 a further development is described and some results are presented. Chapter 5 presents a use of the developed tool in the study of the impact of network in MapReduce jobs performance. Finally the conclusion is presented in chapter 6.

Chapter 2

State of the Art

In this chapter the State of the Art in MapReduce is reviewed. It is presented a definition of MapReduce and its concepts. Technologies used to simulate MapReduce environment that are fundamental to understand the remaining of this thesis are also reviewed.

2.1 Map Reduce

Presented in 2004 by Google, MapReduce introduced a new approach to parallel computing [8]. This framework allows to process large sets of data in a distributed computing environment. The main goal was to simplify the programming model, in which the programmer only needs to write the Map and the Reduce function. All the issues of parallelize the computation, distribute data, fault tolerance and load balancing are oblivious from the programmer. That is the main reason why MapReduce has become the main programming model.

The Map function works by taking some input data to generate intermediate data in the form of <*Key, Value*> pairs. The Reduce function gathers the output information from the Map function and merges all the values with the same key. The simplest example is the word count example. Imagine a document, and that the Map function receives a line of the document as the input. The map function will break the line into words and emit the word and the constant value of '1' for that word. In this case the return will be pair <*Word, 1*> as <*Key, Value*> pair. The reduce function will gather the output of every Map, in other words, all <*Key, Value*> for every line of the document and group the pairs with the same *Key*/*Word*. In the end the result is the number o occurrences of each word in the entire document. In figure 2.1 an example of this scenario is illustrated.

The MapReduce framework is set to run on large clusters with thousands of networked machines and each machine is a cluster node. There are two types of nodes, the master node and worker node. The cluster has 1 master node that is responsible for managing and assign tasks to workers using a schedule algorithm. Some schedule algorithms will be explained in 2.3. All the other nodes are the workers, that are responsible for execute the task that may be a map or reduce task. Nodes are on top of a distributed file system (DFS) that is used to store data from the jobs. The term job refers to something that the user requests the cluster to perform. A task refers to a

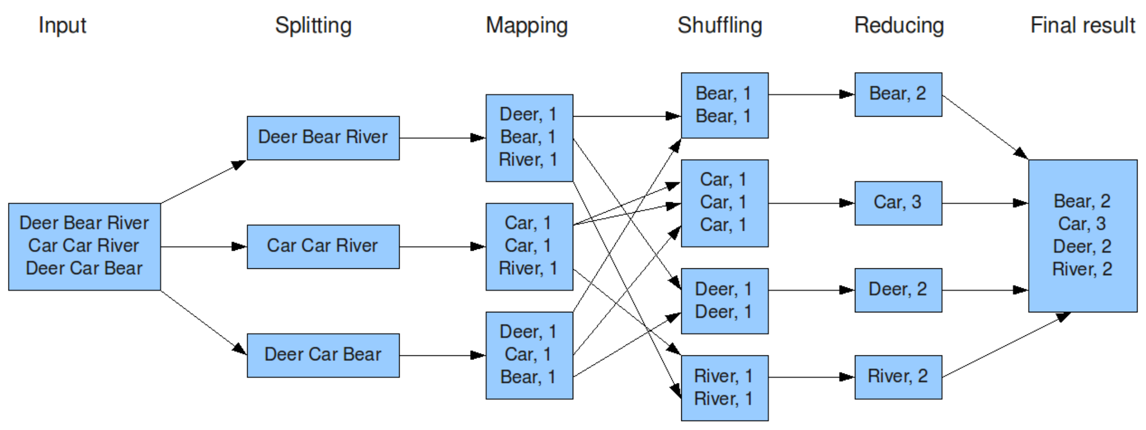


Figure 2.1: MapReduce word count example. [1]

map or reduce function performed be a worker. In MapReduce each node is also DFS node. This will be explained in more detail in section 2.1.2.

2.1.1 MapReduce Phases

MapReduce has two self-evident main phases, the Map phase and the Reduce phase [11], however to a better understanding other sub phases can be identified.

In figure 2.2 a MapReduce data flow is presented. After a job is submitted to the cluster the first phase is the Splitting phase. In this phase the input data is splitted into chunks. The chunk size may vary, but usually values of 64 megabytes or 128 megabytes are used. The chunks are then distributed in the DFS. The load balancing system in the DFS assures that files are equally distributed among the cluster.

After the Splitting phase, comes the Map phase. As said before in the Map phase, the user defined Map function takes the input data, now called chunks, and executes the task. After the Map tasks are completed and produce the <*Key, Value*> pairs, there is a Shuffle and Sort phases

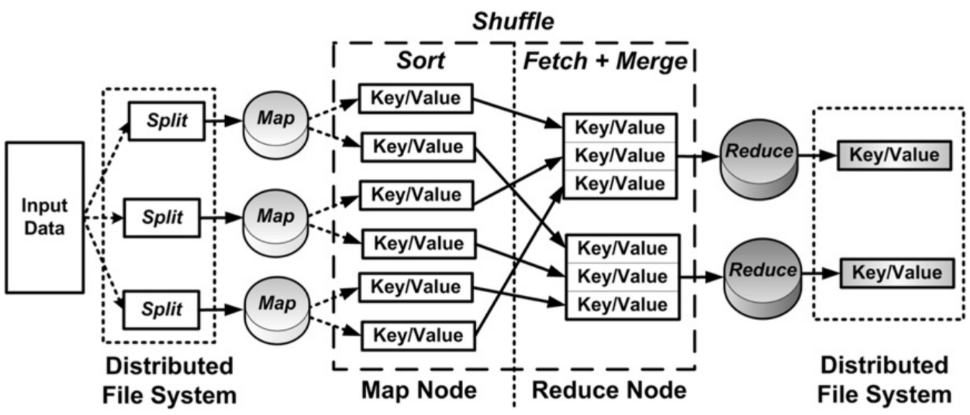


Figure 2.2: MapReduce Phases [2]

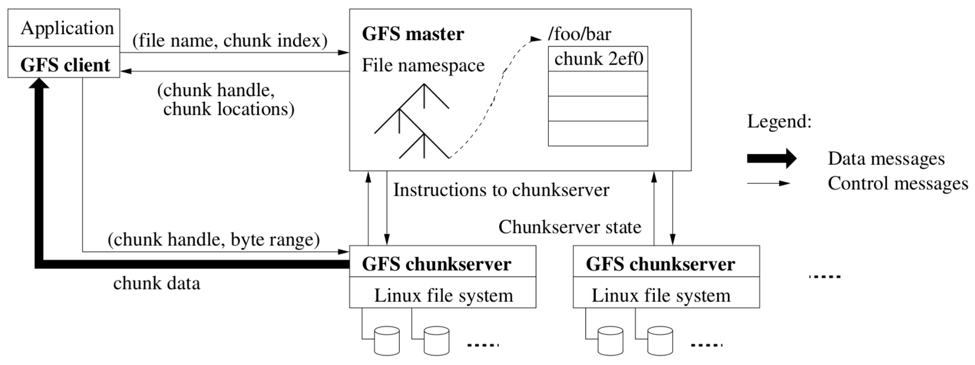


Figure 2.3: Google File System Architecture

before the final Reduce phase. In the Shuffle phase the system fetches the intermediate date files, so that all the pairs with the same key go to the same machine in the cluster, allowing that node to perform the Reduce function. In the Sort phase these files are sorted and merged. Finally in the Reduce phase the user defined Reduce function is executed. Because the Reduce phase needs the output information from the Map phase it can only start after all Map task are completed, on the other hand the Shuffle and Sort phases may start after the Map task on that node is completed.

2.1.2 Distributed File System

Because MapReduce must handle big data sets, this huge amount of data must be stored. The DFS is a key element in MapReduce framework that takes care of the responsibility of storing the input and output of MapReduce jobs. In Google MapReduce implementation they use its own proprietary DFS, the Google File System (GFS) [12]. Its key characteristics are fault tolerance, scalability, availability and performance. Figure 2.3 shows the GFS architecture. In this system master and chunkserver nodes are the same as MapReduce master and worker cluster nodes. GFS also acknowledges clients that may access to the file system.

The Master node do not store data from the jobs, it only stores file system metadata. It is the responsibility of the master node to manage and keep track of the location of the chunks throughout the whole cluster. This includes important features such as load balancing, access control and resource management. Chunkservers must store data chunks, and attend requests from clients. It must also report faults to the master node.

The DFS has important proprieties to assure the above mentioned key characteristics and make this a reliable system. The first propriety is the *locality* feature. Because the MapReduce master node is also the DFS master node, the master node will try to schedule a task to a worker that already as the chunks stored locally, avoiding file transfer. The next propriety is the *data balancing* between chunkservers. As already explained master node distributes the data chunks by the workers trying to maintain the same percentage of local storage usage. The final propriety is the *replication of chunks*. Usually a replication factor of 3 is used, meaning that each chunk is sent to 3 chunkservers. This is a really important feature that allows fault tolerance, if some node stops working, and data availability. As a result of this replication of chunks the locality propriety has a bigger probability. The downside is the extra storage space used.

2.1.3 Fault Tolerance and Backup Tasks

In an environment where there are thousands of machines, failure is inevitable. As Jeff Dean said:

> "If you have one server, it may stay up three years (1,000 days). If you have 10,000 servers, expect to lose ten a day." [13]

To face this problem, some measures such as the already mentioned replication of chunks. The master node must know all the time which machines are available. To address that the master pings every worker periodically. If the worker does not respond in the defined time window, the master marks that worker as failed. This also means that the map or reduce tasks that were assign to that worker may have to be reschedule and re-executed. It is possible that the master also fails, though the probability of this to happen is very small. But if it happens the entire job must be restarted [14].

Sometimes there may be a machine that for some reason have an inferior performance. The machines are called *stragglers*, and they are characterized by taking too much time to complete a task. This machines may become slower because of CPU competition, memory, local disk, network bandwidth or because of big map outputs. These problems can cause poor performance, especially if it occurs in the end of the job. To mitigate this problem with stragglers MapReduce introduced Backup Tasks. If some map or reduce task is close to completion the master schedules backup executions. When one of the tasks ends, whenever is the primary or the backup task, all the task instances are stopped. [8]

2.2 Apache Hadoop

Because Google's implementations of MapReduce and GFS are proprietary, open source solutions appeared. The most popular one is Apache Hadoop [15] implementation of MapReduce. These implementations are based on the papers posted by Google about the MapReduce framework [8] and GFS [12] and sponsored by Yahoo!. Because of this its implementation is very similar and uses the same guide lines. As DFS Hadoop uses the Hadoop Distributed File System (HDFS). More on that on section 2.2.1.

The Apache Hadoop framework follows the same master-slave model explained in section 2.1. Hadoop refers to the master node as Job Tracker and the worker node as Task Tracker [16]. Nevertheless their roles are exactly the same, the Job Tracker node acts as a resource manager and the Task Tracker executes map and reduce tasks.

Because Hadoop was designed to run on commodity hardware[1] fault tolerance is indispensable. Hadoop implemented the heartbeat mechanism, in which every slave node must communicate its aliveness to the master node. The Task Tracker sends small periodic messages called

[1]Commonly available hardware that can be obtained from multiple vendors [11]

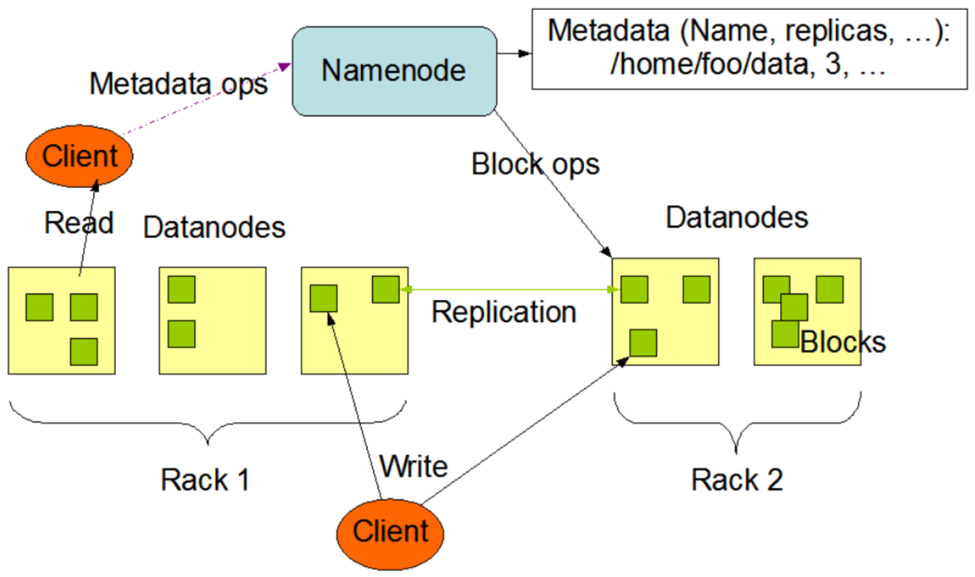


Figure 2.4: HDFS Architecture [3]

heartbeats to the Job Tracker. Hadoop also acknowledges Backup Tasks, called Speculative Tasks, that work as mention in section 2.1.3.

In most recent releases Apache Hadoop introduced Yet Another Resource Manager (YARN) that improves classical Hadoop framework in several aspects [17], however this is not addressed in this thesis.

2.2.1 Hadoop Distributed File System

As said before Apache Hadoop uses the Hadoop Distributed File System (HDFS) as its DFS that was based on GFS. The architecture consists of the Name Node, that has the same role as the GFS master node and the Data Node that works just like the GFS chunkserver.

HDFS as a very similar structure already introduced in section 2.1.2 and has the same fault tolerance mechanism such as load balance and chunk or block replication. Some differences are in the block size, HDFS uses 128MB as default instead of the 64MB default for the GFS [18].

Because "Moving Computation is cheaper than moving data" [3] HDFS has a *rack-aware* replica placement feature. As illustrated in figure 2.4 the HDFS is able to identify the Data Nodes that belong to the same rack and distribute chunks so that network bandwidth utilization, data reliability and availability are improved increasing the system performance [18].

2.3 Scheduler Algorithms

Master node use the scheduler to decide what to assign to a worker. There are two kinds of schedulers, the Task Scheduler and the Job Scheduler. The Task Scheduler helps the master node decide what task should be assigned to what worker. This is particularly helpful if the cluster has some speculative tasks of speculative workers. The user can submit multiple jobs to the cluster, and the master must manage resource allocation, and because workers have a fixed number of map and reduce slots, the Job Scheduler helps the master decide when a job can execute its tasks.

Because reduce phase needs information from the map, it can only begin after all maps from a jobs are completed. So Task Scheduling starts by assigning all maps to workers. If there are more maps tasks than workers available, tasks must wait, until a worker is available. Because the MapReduce nodes are also DFS nodes the master node always tries to fulfil locality, but if this is not possible the worker must fetch the chunk from another worker node. Finally speculative tasks are assigned. When all maps tasks are completed, the master node schedules reduce tasks. In reduce tasks there is no locality, and data is spread among the workers that executed map tasks. In order to improve performance MapReduce can use prefetching [19, 20], this means fetching already processed map tasks data during map phase. After all reduce tasks are assigned, speculative reduce tasks are assigned.

Job scheduling is a vital element in MapReduce framework, and it is no wonder that over the last few years several solution appeared trying to improve or develop new solutions to this component. The default scheduler in Hadoop is the First-in First-out (FIFO) scheduler. Later Hadoop introduced Hadoop on Demand (HOD), to correct FIFO problems. In [21] and [22] the authors proposed *Delay Scheduling* as a solution for HOD. In [23] the authors proposed Classification and Optimization based Scheduler for Heterogeneous Hadoop (COSHH), and although they did not used that name in that article, it was given in later papers namely [24] were the authors also proposed an hybrid solution between FIFO and COSHH. According to [25] Hadoop performs very poorly in heterogeneous clusters, and to address this issue [26] proposed Longest Approximate Time to End (LATE) to optimize heterogeneous clusters performance, especially straggler machines. This is with no doubt an area with an extensive research, but only the most used scheduler algorithms are will be briefly reviewed in the following sections.

2.3.1 FIFO

As mentioned before this is the default scheduler in Hadoop. Tasks are queued in a FIFO order based on their arrival times, with five priority levels, however this priority level system is not turned on by default in Apache Hadoop. Because of the basic principles this scheduling algorithm poor response times for short jobs in the presence of large jobs [21].

2.3.2 Hadoop Fair Scheduler

Because job response times with FIFO scheduler were too long, Facebook designed Hadoop Fair Scheduler. This scheduler defines a pool for each user, and each pool consists of a number of maps and reduce slots on a resource. If there are free slots in inactive pools these may be used by other pools. This scheduler objective is to give a fair share of the cluster capacity over time.

Table 2.1: Data Center Network topologies comparition [5]

Reference	Scalability	Cross-section bandwidth	Cost effectiveness	Fault tolerance	Network topology
Three-tier	Medium	Low	Low	Medium	Hierarchical
Fat-tree	High	High	High	High	Clos based hierarchical topology
DCell	Very High	Medium	High	Medium	Recursively defined topology, based on DCell0
BCube	Low	High	High	Medium	Recursively defined topology, based on BCube0
VL2	High	High	High	High	Clos based hierarchical topology
CamCube	Low	High	High	High	Torus
Jellyfish	High	High	High	High	Flat

2.3.3 Capacity Scheduler

Designed by Yahoo Capacity Scheduler [27, 28] is used when there are a large number of users, to ensure a fair allocation of computational resources. Jobs are organized in queues, with FIFO scheduling, and share the cluster, with the configured number of map and reduce slots.

2.4 Data Center Network Topology

A good network topology is crucial to a good MapReduce performance, as a bad designed topology can lead to bottlenecks and reduce MapReduce performance drastically [29]. Traffic in high performance computing is often bursty, meaning that a large volume of data is injected into the network in a small period of time [30]. This is why, as said before, features such as locality is so important. However reduce tasks don't have locality and network as an impact on its performance.

There are some factors that influence in the design of a data center topology such as the cost, scalability and of course performance. Typically network engineers look for the best cost-effective solution. In table 2.1 is a comparative of the most typical Data Center network topologies.

In figure 2.5 is shown a typical cluster architecture. In this conventional network topology there are three layers. The several racks are in the Access Layer, and each rack has several servers and a Top-of-Rack (ToR) switch. Aggregation Switches from the Aggregation Layer connect several ToR switches. Each ToR switch is connected to one or more Aggregation Switches. Core Routers are in the top layer, the Core Layer, and connect to the Aggregation Switches. Typically a used switches use GigE to connect to servers (ToR switches) and 10 GigE to interconnect switches and routers [31, 32].

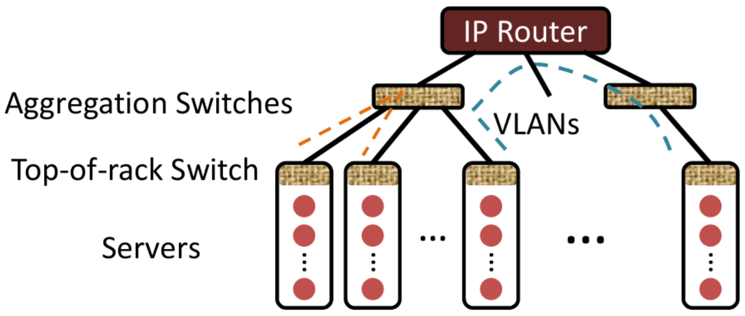


Figure 2.5: Typical Cluster Architecture [4]

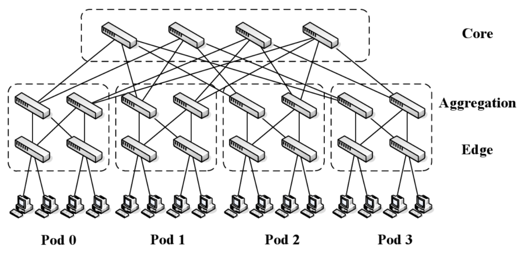
One of the most common network topology is the Fat-Tree topology 2.6(a). This topology is based on the Clos topology in which each switch of one stage are connected to every switch from the next stage [34]. This architecture is composed by k pods supporting in total $(k^3/4)$ servers. Both Aggregation e Edge layer have $k/2$ switches, and there are $(k/2)^2$ switches in the Core layer. This topology is highly scalable and very cost-effective, but requires a large number of switches [33].

Virtual Layer 2 (VL2) network topology 2.6(b) is a hierarchical Fat-Tree based topology. This topology purpose is to use commodity switches throughout the network, increasing the cost-effectiveness. This architecture is composed by $D_a/2$ Intermediate switches and D_1 Aggregation switches. $D_a/2$ and D_1 are the port number in Aggregation and Intermediate switches respectively. This configuration supports a total of $20(D_a)(D_1)/4$ servers [35].

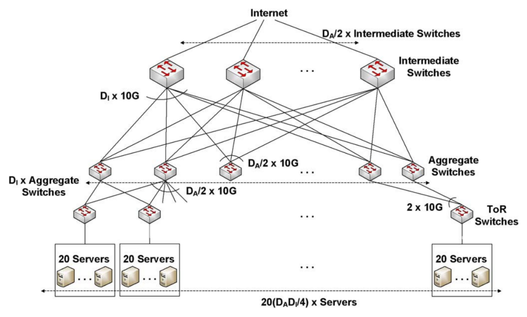
DCell network topology 2.6(c) is a hybrid topology and uses switches and servers to packet forwarding. It uses a basic building block, $DCell_0$ that is scaled up recursively. This basic building block consists in a commodity switch and an small n number of servers. The next DCell level, $DCell_1$, is constructed using $n+1$ $DCell_0s$ with each serve connected to the mini-switch and to other SCell server. In figure 2.6(c) A $DCell_1$ with $n=4$ is illustrated. The next levels of DCells are constructed in the same way. In the end the total number of server is $t_k = g_k \times t_{k-1}$ with k being the DCell level and g the number of $DCell_{k-1}$s in $DCell_k$ given by $g_k = t_{k-1}+1$. In $DCell_0$ $g_0 = 1$ and $t_k = n$ [36, 33].

Another hybrid recursive network topology is BCube 2.6(d). Although not as scalable it provides high bisection bandwidth. Like DCell, BCube also uses a basic building block, $BCube_0$, with n servers and n n-port commodity switches. Level 1 $BCube_1$ is composed by n $BCube_0$s and n n-port switches. In figure 2.6(d) a $BCube_1$ with n = 4 is illustrated. So level k has n $BCube_{k-1}$s and n^k n-port switches, empowering n^{k+1} (k + 1)-port server (k + 1) layers of switches [37, 33].

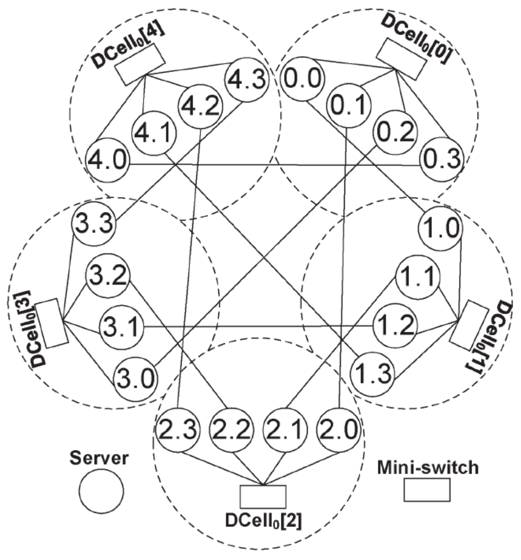
In addition to these DCNs network architectures there other topologies in literature, but will not be reviewed in this dissertation.



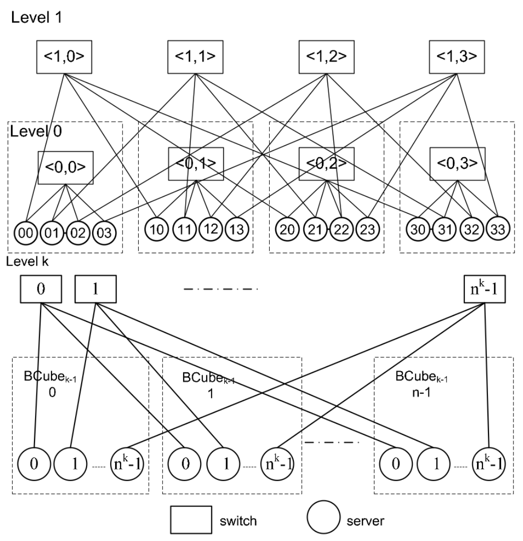
(a) Fat-Tree [31]



(b) VL2 [33]



(c) DCell$_1$ using 5 DCell$_0$ [33]



(d) BCube [33]

Figure 2.6: Network Topologies

2.5 Related Work and Simulator Comparative

There are some simulator implemented that simulate MapReduce environments. In this section a brief review of each one is presented.

- **MRPerf**

  Being a pioneer in MapReduce simulation, *MRPerf* born from the need of a simulation platform to evaluate MapReduce performance, its scalability, and modelling different cluster topologies. Because of this last reason this simulator is built on top of *ns-2* network simulator [38]. The main drawback of this simulator is that does not simulate chunk replication, speculative tasks and scheduling schemes [39].

- **MRSim**

  Later appeared *MRSim* [40]. This allowed to simulate more parameter than MRPerf such as multi core CPU, HDD and cluster configuration. This simulator was used in [23] to evaluate

Table 2.2: Comparison between network simulators [6]

	PlanetLab	ns-2	GTNetS	ChicSim	OptorSim	GridSim	SimGrid
Nodes	<850	<1 000	177 000	thousands	few 100	few 100	few 10 000

the schedule algorithm, however does not provide an interface to modify algorithms present in the simulator.

- **MapReduce over SimGrid**

  MRSG [2] is a flexible simulator built on top of SimGrid simulator. This simulator was used in [41] to test performance in a large scale in heterogeneous environments. Although it allows user to provide its own algorithms, does not simulate multiple job instances.

Other simulator such as *SimMapReduce* [39], *SimMR* [42], that allowed to use MapReduce logs to replay the results, or *Cloud^{2}Sim* [43], based on CloudSim were also implemented.

From the found solutions, all of them none of them were in a comfortable programming language such as Java. Java is a modern and a more welcoming programming language and programmers may prefer this language instead of others less friendly as is the case of C. Despite the fact that MRSim is in Java, this simulator is very focus on the hardware and resource usage and not on network analysis, and it is only able to simulate local racks.

2.5.1 SimGrid

SimGrid [44] is a simulator used mostly for large-scale distributed systems simulation. This includes a network platform. So far this simulator has been used for distributed systems simulation and peer-to-peer network simulation.

There are some network simulators to choose from, such as ns-2 or GTNetS, and although they have a high-accuracy, they lead to long simulation times [45]. Table 2.2 refers some available simulator and their scalability.

The study done in [45] and [6] compares SimGrid and GTNetS, and concludes that SimGrid as very accurate results in packet-level simulation. The main challenge arises when data size is too small, because SimGrid does not account for TCP slow-start mechanism. This is not expected to be a problem for MapReduce simulation as data size is usually sizable.

SimGrid uses XML files to describe the network topology, configuration and to specify the hosts. These files allow users to define machines power, availability, network bandwidth and latency.

2.5.2 Network impact on MapReduce performance

The research done shows that although existing simulators allow for network analysis, such as MRPerf and MRSG, because they are built on top of capable network simulators, ns-2 and SimGrid respectively, there is no study on the impact of the network architecture on MapReduce performance. As mention in section 2.5 network have in impact in the duration of MapReduce jobs, and may become an bottleneck if not design properly or in case of a link failure.

In [46] concluded that there is tremendous interest in data center network design, but the study in data center traffic is very scarce. The must similar studies are [47] and [4], but the former focuses on multiple data center services and the later although uses MapReduce traffic for its analysis it does not perform a study on the performance of MapReduce.

2.6 Conclusion

This chapter provided deep knowledge about MapReduce and the technologies used today improve MapReduce performance. Also allowed to understand that research in this field is far from over, and many new techniques are emerging every day, reason that motivated this present dissertation. This thesis focus on the open source version of MapReduce, the Apache Hadoop, but it is clear the similarity between Hadoop and Google implementation of MapReduce. Schedule algorithms for MapReduce are very important, and the research found in this topic is very extensive. With this in mind it is of great importance the the implemented simulation solution allow testing different schedule algorithms algorithms. Found network topologies show that when it comes to making a decision in building strong network for a data center it is not an easy task. All of them have strengths and weakness, and also have an impact in MapReduce performance. The attempts to build a simulation platform so far reveal that the huge amount of components associated with MapReduce cause this to be very challenging. This is why most of them only simulate some aspects of MapReduce.

Knowing the goal and the state of the art in MapReduce simulation, the next chapter introduces first steps in the simulation platform this thesis intend to accomplish.

Chapter 3

MapReduce over SimGrid

In this chapter the MRSG simulator referred in section 2.5 will be explored, as this will be the starting point of the solution proposed by this thesis. Then some limitations will be discussed and consequent improvements implemented are presented.

3.1 MRSG

MapReduce over Simgrid is a MapReduce simulator developed by [2] and has it source code available in [48]. This simulator is developed in C programming language on top of SimGrid. As it was referred in section 2.5.1, SimGrid is a simulation framework used to evaluate large-scale distributed systems, and in this thesis is intended to give some relevance to the network aspect of this simulation tool.

MRSG uses Apache Hadoop MapReduce and HDFS as a guide line and offers an API that allows the user to implement its own algorithms, such as task scheduling and data distribution. In figure 3.1 is a representation of the MRSG overall architecture with SimGrid. As it can be seen in the figure SimGrid is used to simulate the the data transportation over the network and the computation of map and reduce tasks. MRSG uses the MSG SimGrid API for this interaction. MRSG is responsible for the DFS and the MapReduce simulation. The user algorithms, SimGrid XML file and job configuration are specified through MRSG API.

The MRSG available version already has example files and to run the simulation it is required to have SimGrid installed in the system. The job configuration is a simple text file and it is possible to configure the following parameters:

- number of Map and Reduce tasks;
- chunk size;
- dfs replication factor;
- number map and reduce slots available for each machine

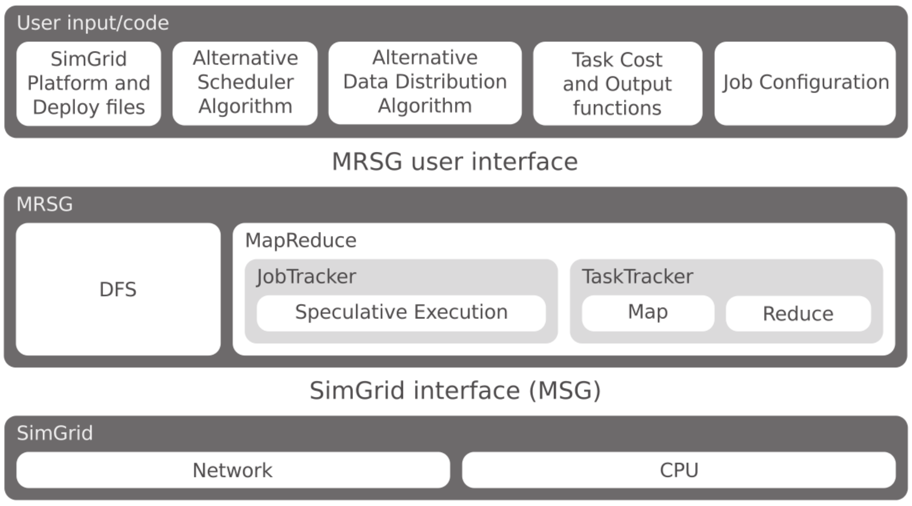


Figure 3.1: MRSG architecture [2]

The number of workers and all the network information are specified in SimGrid platform and deploy XML files.

An example of the output of MRSG job configuration is presented bellow.

```
JOB CONFIGURATION:
slots: 2 map, 2 reduce
chunk replicas: 3
chunk size: 64 MB
input chunks: 60
input size: 3840 MB
maps: 60
reduces: 24
workers: 6
grid power: 1.00038e+11 flops
average power: 1.6673e+10 flops/s
heartbeat interval: 3s
```

MRSG also has statistics with local, non-local and speculative tasks, and outputs a CSV file with the beginning and end times of each task.

3.1.1 DFS

The DFS is implemented using a matrix that maps chunks to workers. Table 3.1 illustrates the DFS matrix. Each line represent a chunk and each column represent a worker. A value of one indicates that the worker possesses that chunk, and a value of zero indicates that the worker does not possesses the chunk. The master also has access to this information just like in the HDFS

Table 3.1: Distributed file system representation

	W_0	W_1	W_2	W_{m-1}
C_0	1	0	0	1
C_1	0	1	0	0
C_2	1	0	0	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_{n-1}	0	0	1	0

implementation, and this way is easy to associate a chunk to one or more workers, allowing for replica simulation. This is also used to implement the locality property.

Although this simulator already as all implemented functions it is possible for users to specify its own algorithms. An example is the data distribution function. The pseudo-code illustrates a possible data distribution algorithm.

```
function dfs (dfs_matrix, chunks, workers, replicas)
    for each c in chunks:
        for each r in replicas:
            w = choose_worker ()
            dfs_matrix[c][w] = 1
end function
```

3.1.2 Scheduler

The scheduler in MRSG works as follows. After receive an heartbeat the master node checks the matrix and always tries to follow locality feature, thus for map tasks it chooses a chunk that is stored locally in the worker. If that is no possible it will assign a task whose chunk is stored in another worker. Speculatives tasks follow the same criteria, this is locality first, and then non-local tasks.

Reduce tasks don't have locality, so the only criteria is normal reduce tasks first and then speculative reduce tasks. In this case data is spread among workers that processes map tasks, so the worker will retrieve the data from those workers. The size of the data is defined by the user, through an API function, that calculates the map output size. MRSG also simulates data prefetching referred in section 2.3.

Just like data distribution algorithm users can specify their own scheduling algorithms allowing alternative algorithms to be tested.

3.1.3 Limitations

Although this solution already is very complete it has some drawbacks. The first one is the computation power and tasks duration that must be specified in FLOPS [1]. This is not desirable or a

[1]FLOPS is a measure for a computer processor performance, given by measuring the processor's floating point unit(FPU)

practical way to indicate the required computation power for tasks. This restriction is due to the way of how SimGrid simulates computations.

Another limitation is the job limit. It only allows to simulate one job per simulation. This prevents multiple job schedulers from being tested and as it was discovered in section 2.3 this is an important research topic.

The disc access MRSG has user implemented functions

The programming language is not a major drawback, however as mentioned in section 2.5 some people don't like C as a programming language, and prefer other more friendly programming language such as Java.

3.2 Improvements

The first improvement done was changing the programming language. The simulator was rewritten in Java programming language. SimGrid is very versatile and is supported in both languages. To be able to run the simulation in C, SimGrid packages must be installed in the machine, however in Java only the '.jar' SimGrid library must be present to run the simulation.

The reimplementation required knowledge about the SimGrid framework and required learning how this simulator works. Must of the functions implemented in SimGrid C version are present in the Java version, but not all of them. This problem created a major issue, because the needed functions were not documented and required some work arounds and were very time consuming.

A comparative betweeen both implementations is presented in section 3.3.

The seconds improvement was the task cost. The Flop measure used to implement the task cost required by SimGrid is not something people are used measure. To counter this the task cost function was modified to allow the user to import results from real measures. To test this implementation it were used results from [49].

3.3 Results

To compare both versions the bellow configurations were used. Result are in figure 3.2 and 3.3.

Job Configuration(Results in 3.2):

```
JOB CONFIGURATION:
slots: 2 map, 2 reduce
chunk replicas: 3
chunk size: 64 MB
input chunks: 60
input size: 3840 MB
maps: 60
reduces: 24
workers: 6
grid power: 1.00038e+11 flops
average power: 1.6673e+10 flops/s
heartbeat interval: 3s
```

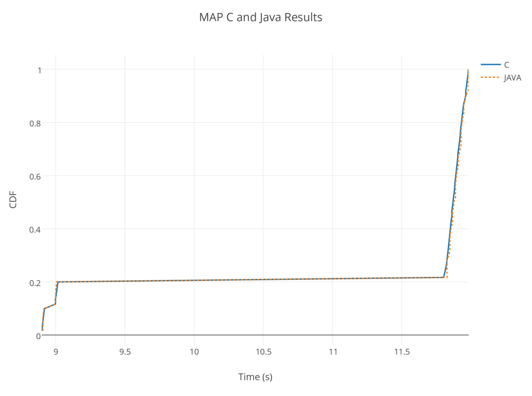
User defined functions:
MapTaskCost: 1e+11 flops;
ReduceTaskCost: 5e+11 flops;
MapOutput: 4*1024*1024;

Job Configuration(Results in 3.3):

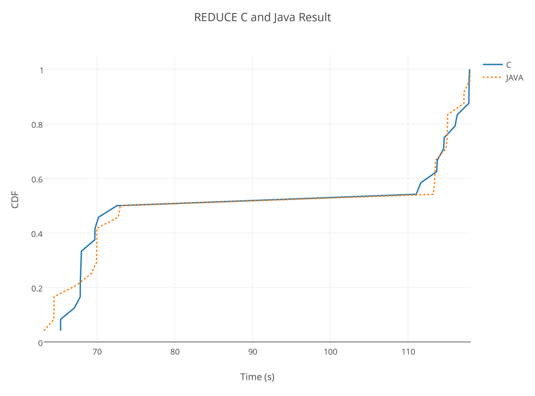
```
JOB CONFIGURATION:
slots: 2 map, 2 reduce
chunk replicas: 3
chunk size: 64 MB
input chunks: 1000
input size: 64000 MB
maps: 1000
reduces: 100
workers: 6
grid power: 1.00038e+11 flops
average power: 1.6673e+10 flops/s
heartbeat interval: 3s
```

1e+11 * $rand(1-100)^2$

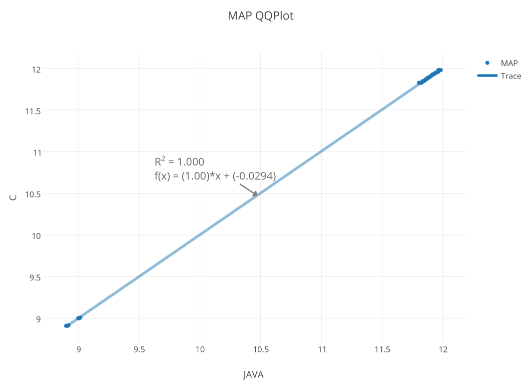
MapTaskCost: 1e+9 * $rand[1:100]^2$ flops;
ReduceTaskCost: 2e+9 * $rand[1:100]^2$ flops;
MapOutput: 4*1024*1024;



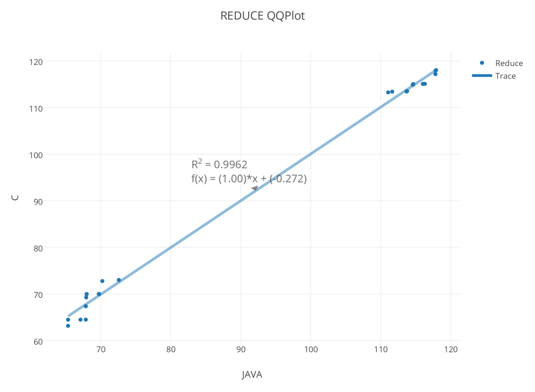
(a) MAP phase C and JAVA results



(b) REDUCE phase C and JAVA results

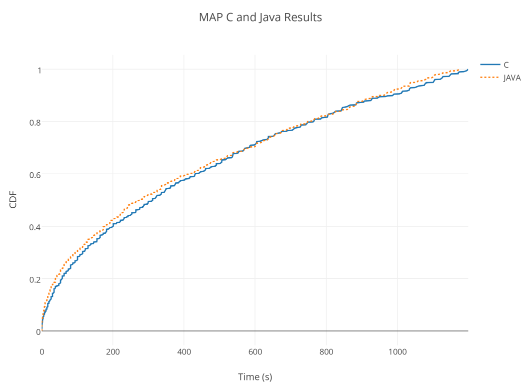


(c) MAP qq plot

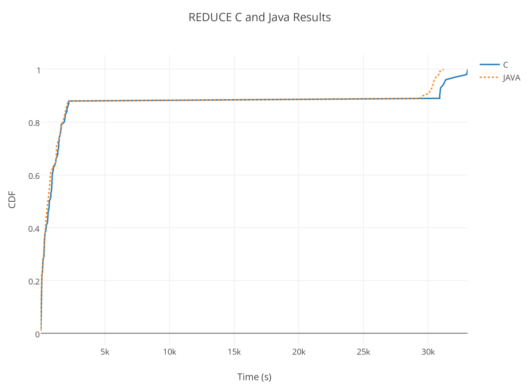


(d) REDUCE qq plot

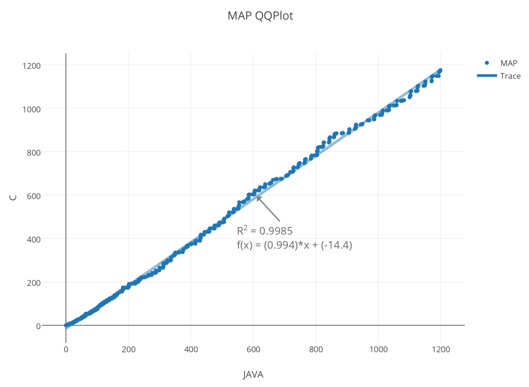
Figure 3.2: C and Java comparative



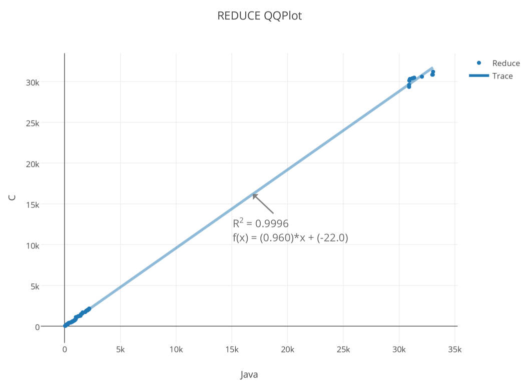
(a) MAP phase C and JAVA results



(b) REDUCE phase C and JAVA results



(c) MAP qq plot



(d) REDUCE qq plot

Figure 3.3: C and Java comparative

Chapter 4

Multiple Job Scheduler

4.1 Modifications

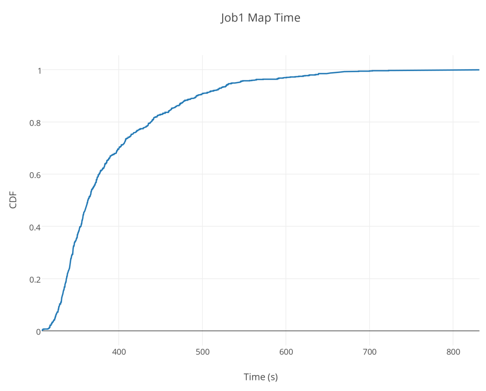
The simulator was modified to allow simultaneous multiple jobs.

In the configuration file are specified the amount of jobs to simulate and the path to the configuration files of each job. Each job configuration file contains the amount of maps and reduce tasks for that job. In the configuration file it is possible to specify if jobs are running independently, or simultaneously. If jobs run independently, the are put in a FIFO queue and the second jobs only starts after the first jobs finishes. This means that the jobs has the entire cluster for him alone, and jobs will not share resources. If jobs run simultaneously, they share the cluster resources. In this case the scheduler will use a Round Robin scheme to decide which job should use the resources.

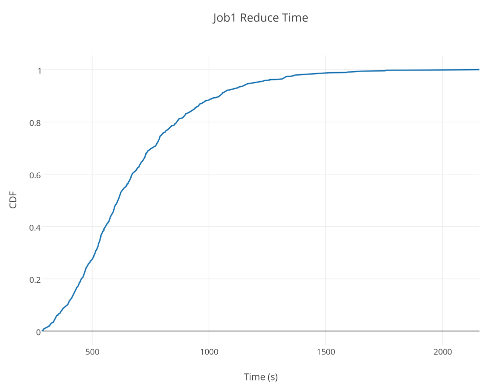
In both results the the defined user functions are: MapTaskCost: 1e+9 * Map duration values from [49];
ReduceTaskCost: 1e+9 * $Gaussian[-1;1]^2 x 100 x (rand[0:1]+1)$ flops;
MapOutput: $Gaussian[-1;1]^2 x 8 x 1024 x 1024$

Running Jobs separately: Results are in figure 4.1.

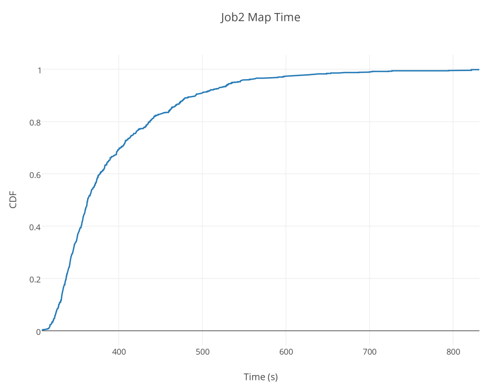
Job Configuration(Results in 3.3):



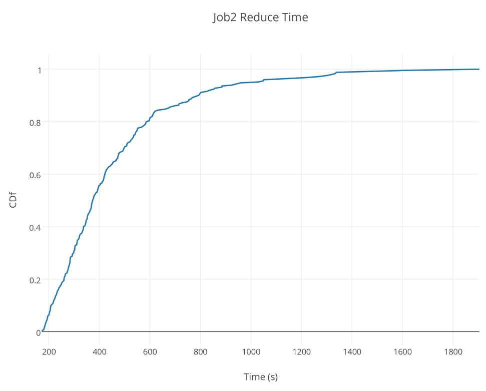
(a) Job1 Map Time



(b) Job1 REDUCE Time



(c) Job2 Map Time



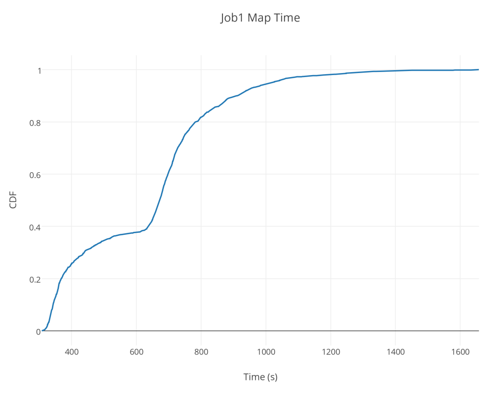
(d) Job2 REDUCE Time

Figure 4.1: Jobs running separately

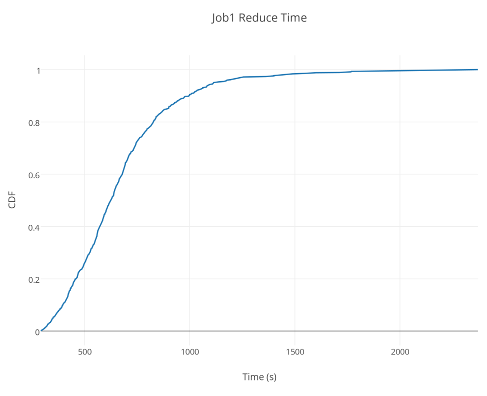
JOB 1 CONFIGURATION:	JOB 2 CONFIGURATION:
Jobs: 1	Jobs: 1
slots: 1 map, 1 reduce	slots: 1 map, 1 reduce
chunk replicas: 3	chunk replicas: 3
chunk size: 128.0 MB	chunk size: 128.0 MB
input chunks: 5000	input chunks: 3000
input size: 640000 MB	input size: 384000 MB
maps: 5000	maps: 3000
reduces: 750	reduces: 250
workers: 7	workers: 7
grid power: 7.0E9 flops	grid power: 7.0E9 flops
average power: 1.0E9 flops/s	average power: 1.0E9 flops/s
heartbeat interval: 3s	heartbeat interval: 3s

Running Jobs simultaneously: Results are in figure 4.2.

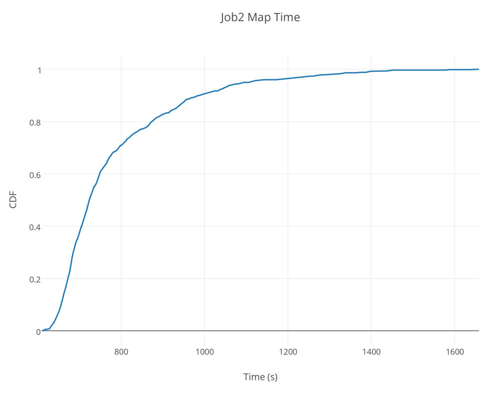
Job Configuration(Results in 3.3):



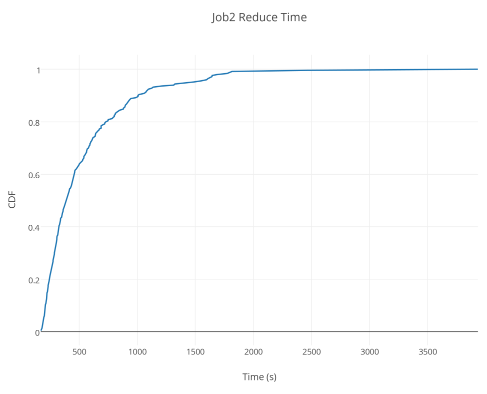
(a) Job1 Map Time



(b) Job1 REDUCE Time



(c) Job2 Map Time



(d) Job2 REDUCE Time

Figure 4.2: Jobs running simultaneously

JOB CONFIGURATION:
Jobs: 2
slots: 1 map, 1 reduce
chunk replicas: 3
chunk size: 128.0 MB
input chunks: 8000
input size: 1024000 MB
maps: 8000
reduces: 1000
workers: 7
grid power: 7.0E9 flops
average power: 1.0E9 flops/s
heartbeat interval: 3s

4.2 Results

Chapter 5

Network Impact on MapReduce Performance

In this chapter it will be presented a study from the network impact on the MapReduce performance. The topologies studied in 2.4 will be a reference for this study.

Chapter 6

Conclusion

This dissertation pretends to help the research of new algorithms to improve MapReduce framework. The proposed solution is a simulation environment in which the user can specify algorithms, such as schedule and data distribution and simulate their performance. The first step was to study MapReduce and its development state and the current solution for MapReduce Simulation. After concluding that the found solutions are incomplete, the next step was to improve one of the simulators that was already in a good development state. After that some of the results from the study are presented.

References

[1] Mapreduce word counter example. Accessed: 2016-01-13. URL: http://xiaochongzhang.me/blog/?p=338.

[2] Wagner Kolberg, Pedro De B Marcos, Julio CS Anjos, Alexandre KS Miyazaki, Claudio R Geyer, and Luciana B Arantes. Mrsg - a mapreduce simulator over simgrid. *Parallel Computing*, 39(4):233–244, 2013.

[3] Dhruba Borthakur. Hdfs architecture guide. Accessed: 2016-01-16. URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[4] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[5] Kashif Bilal, Saif Ur Rehman Malik, Osman Khalid, Abdul Hameed, Enrique Alvarez, Vidura Wijaysekara, Rizwana Irfan, Sarjan Shrestha, Debjyoti Dwivedy, Mazhar Ali, et al. A taxonomy and survey on green data center networks. *Future Generation Computer Systems*, 36:189–208, 2014.

[6] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[7] Jakob Nielsen. Nielsen's law of internet bandwidth. Accessed: 2016-01-11. URL: http://www.nngroup.com/articles/law-of-bandwidth/.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. URL: http://doi.acm.org/10.1145/1327452.1327492, doi:10.1145/1327452.1327492.

[9] JoshBaer. Poweredby - hadoop wiki. Accessed: 2016-01-11. URL: http://wiki.apache.org/hadoop/PoweredBy.

[10] Cisco. Cisco visual networking index: Forecast and methodology, 2014-2019 white paper. Accessed: 2016-01-11. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[11] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[13] Jeffrey Dean. Experiences with mapreduce, an abstraction for large-scale computation. In *PACT*, volume 6, pages 1–1, 2006.

[14] Zhifeng Xiao and Yang Xiao. Accountable mapreduce in cloud computing. In *Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 11)*, pages 1082–1087, 2011.

[15] The Apache Software Foundation. Welcome to apache^{TM} hadoop®! Accessed: 2016-01-14. URL: https://hadoop.apache.org/.

[16] Mohammad Hammoud, M Suhail Rehman, and Majd F Sakr. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 49–58. IEEE, 2012.

[17] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[19] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, Bibo Tu, and Jianping Fan. Improving data locality of mapreduce by scheduling in homogeneous computing environments. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, pages 120–126. IEEE, 2011.

[20] Sangwon Seo, Ingook Jang, Kyungchang Woo, Inkyo Kim, Jin-Soo Kim, and Seungryoul Maeng. Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–8. IEEE, 2009.

[21] Matei Zaharia, Dhruba Borthakur, J Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Job scheduling for multi-user mapreduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*, 2009.

[22] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[23] Aysan Rasooli and Douglas G Down. An adaptive scheduling algorithm for dynamic heterogeneous hadoop systems. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, pages 30–44. IBM Corp., 2011.

[24] Aysan Rasooli and Douglas G Down. A hybrid scheduling approach for scalable heterogeneous hadoop systems. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1284–1291. IEEE, 2012.

[25] Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and TN Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 61–74. ACM, 2012.

[26] Andrew Konwinski. *Improving mapreduce performance in heterogeneous environments*. EECS Department, University of California, Berkeley, 2009.

[27] Matei Zaharia. Job scheduling with the fair and capacity schedulers. *Hadoop Summit*, 9, 2009.

[28] B Thirumala Rao and LSS Reddy. Survey on improved scheduling in hadoop mapreduce in cloud environments. *arXiv preprint arXiv:1207.0780*, 2012.

[29] Xinkui Zhao, Jianwei Yin, Zuoning Chen, and Xingjian Lu. Distance-aware virtual cluster performance optimization: A hadoop case study. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[30] Ning Liu, Adnan Haider, Xian-He Sun, and Dong Jin. Fattreesim: Modeling large-scale fat-tree networks for hpc systems and data centers using parallel and discrete event simulation. In *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*, pages 199–210. ACM, 2015.

[31] M Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Z Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *Communications Surveys & Tutorials, IEEE*, 15(2):909–928, 2013.

[32] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[33] Roberto Rojas-Cessa, Yagiz Kaymak, and Ziqian Dong. Schemes for fast transmission of flows in data center networks.

[34] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[35] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[36] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Computer Communication Review*, 38(4):75–86, 2008.

[37] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[38] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. Using realistic simulation for performance analysis of mapreduce setups. In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, pages 19–26. ACM, 2009.

[39] Fei Teng, Lei Yu, and Frederic Magoulès. Simmapreduce: A simulator for modeling mapreduce framework. In *Multimedia and Ubiquitous Engineering (MUE), 2011 5th FTRA International Conference on*, pages 277–282. IEEE, 2011.

[40] Suhel Hammoud, Maozhen Li, Yang Liu, Nasullah Khalid Alham, and Zelong Liu. Mrsim: A discrete event based mapreduce simulator. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 6, pages 2993–2997. IEEE, 2010.

[41] Julio Anjos, Wagner Kolberg, Claudio R Geyer, and Luciana B Arantes. Addressing data-intensive computing problems with the use of mapreduce on heterogeneous environments as desktop grid on slow links. In *Computer Systems (WSCAD-SSC), 2012 13th Symposium on*, pages 148–155. IEEE, 2012.

[42] Abhishek Verma, Ludmila Cherkasova, and Roy H Campbell. Play it again, simmr! In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 253–261. IEEE, 2011.

[43] Pradeeban Kathiravelu and Luis Veiga. An adaptive distributed simulator for cloud and mapreduce algorithms and architectures. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 79–88. IEEE, 2014.

[44] SimGrid. Simgrid home. Accessed: 2016-01-16. URL: `http://simgrid.gforge.inria.fr/`.

[45] Kayo Fujiwara and Henri Casanova. Speed and accuracy of network simulation in the simgrid framework. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, page 12. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[46] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[47] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[48] wkolberg. Mrsg: A high-level mapreduce simulator. Accessed: 2016-01-21. URL: `https://github.com/MRSG/MRSG`.

[49] Ricardo Morla, Pedro Gonçalves, and Jorge Barbosa. A scheduler for cloud bursting of map-intensive traffic analysis jobs. 2015.