Universidad Carlos III de Madrid

DEPARTMENT OF COMPUTER SCIENCE

BACHELOR THESIS

ADAPTATION, DEPLOYMENT AND EVALUATION OF A RAILWAY SIMULATOR IN CLOUD ENVIRONMENTS

Author: Silvina Caíno Lores

Supervisor: Alberto García Fernández

BACHELOR DEGREE IN
COMPUTER SCIENCE

LEGANÉS, MADRID
JUNE 2014

CONTENTS

INDEX OF TABLES

INDEX OF FIGURES

ACKNOWLEDGEMENTS

In a personal level, I would like to express my deep appreciation for the huge moral support my mother, my family and Silvia had provided me with, for it helped me to endure during the toughest periods of my studies. I also find necessary to acknowledge the valuable opinion of some friends, with a special mention to Ángel and his tireless willingness to discuss my project-related concerns with me.

I would like to offer my special thanks to all the ARCOS staff, whose deep knowledge and passion continuously inspire me to pursue a career in research. Additionally, wish to thank the help provided by the Tucán cluster administrators for their technical support. I want to dedicate a warm thankful line to the lab fellows, as they have always tried to cheer me up when things seemed to go plain wrong.

Finally, I am deeply grateful to my supervisor, Alberto García, for his priceless advice and dedication to this project, and to the ARCOS director, Prof. Jesús Carretero, for his knowledgeable key suggestions and for giving me the opportunity to collaborate with his team.

ABSTRACT

Many scientific areas make extensive use of computer simulations to study real-world processes. As they become more complex and resource-intensive, traditional programming paradigms running on supercomputers have shown to be limited by their hardware resources.

The Cloud and its elastic nature has been increasingly seen as a valid alternative for simulation execution, as it aims to provide virtually infinite resources, thus unlimited scalability. In order to benefit from this, simulators must be adapted to this paradigm since cloud migration tends to add virtualization and communication overhead.

This work has the main objective of migrating a power consumption railway simulator to the Cloud, with minimal impact in the original code and preserving performance. We propose a data-centric adaptation based in MapReduce to distribute the simulation load across several nodes while minimising data transmission.

We deployed our solution on an Amazon EC2 virtual cluster and measured its performance. We did the same in in our local cluster to compare the solution's performance against the original application when the Cloud's overhead is not present. Our tests show that the resulting application is highly scalable and shows a better overall performance regarding the original simulator in both environments.

This document summarises the author's work during the whole adaptation development process .

CHAPTER 1

INTRODUCTION

1.1 Motivation

Scientific simulations constitute a major set of applications that attempt to reproduce real-world phenomena in a wide range of areas such as engineering, physics, mathematics and biology. Their complexity usually yields a significant resource usage regarding CPU, memory, I/O or a combination of them.

As simulations become more and more complex, the amount of input, intermediate, and output data has increased notably [1]. Furthermore, if it is desired to study several variables involved in the experiment, or the simulation behaviours across different domains (time domain and spatial domain, for instance), a single simulation is not sufficient, hence time and resources needed to run a simulation must be factored by the number of simulations that have to be executed. Therefore, the use of simulators is limited by the availability, computing power and overall resources of the underlying computing infrastructure, which can be constituted by datacenters, supercomputers or clusters, for example. In this context, the usage of HPC technologies –such as MPI, OpenMP, GPGPUs– over clusters and supercomputers is the current major trend in simulations [2].

Another recent option is Cloud Computing, which has been increasingly studied as an alternative to traditional grid and high-performance distributed environments for resource-demanding and data-intensive scientific simulations [3]. Cloud Computing emerged with the idea of virtual unlimited resources obtainable on-demand with minimal management effort [4]. It would enable the execution of large simulations with virtual hardware properly tailored to fit specific use cases like memory-bound simulations, CPU-dependant computations or data-intensive analysis. It holds further advantages, such as elasticity, automatic scalability and instance resource selectivity which, along with its so-called pay-as-you-go model, allow to adjust the required instances to the particular test case size while cutting-down the resulting costs.

Cloud's capabilities to achieve virtually unlimited scalability are very attractive, especially if we also consider that recent advances in cloud interoperability and cloud federations can contribute to separate application scalability from datacenter size [5, 6]. From this point of view, application migration to the cloud would be beneficial for simulator's users as more complex cases could be tackled.

This project approaches a railway installation power consumption simulator as a particular case and good example of an application that holds potential scalability issues on large cases. The initial version of the simulator, based on multi-threading, is memory bounded, strongly limited by the number of instants to be simulated simultaneously and, therefore, by the number of threads. This limitation can be overcome by breaking the simulation into a set of smaller simulations that could run independently across several nodes, specially if we are able to scale the cluster size to the necessary resources. Hence, it seems natural to adapt and migrate the simulator to a scalable virtual cluster on the Cloud, given its elastic features.

The MapReduce framework was selected in order to successfully migrate the algorithm because it is inherently data-centric [7]. Our selected MapReduce implementation is the popular Apache Hadoop [8], whose distributed file system allows automatic load balance and data replication to help with robustness and data locality. Hadoop has been increasingly adopted into cloud environments along with other MapReduce frameworks, resulting in reduced costs given its parallelism exploitation capabilities [9].

1.2 Objectives

The main objective of this project is to develop a functional adaptation of the railway power consumption simulator to the Cloud, in order to allow end-users the execution of large test cases that would overwhelm standalone and cluster hardware resources.

We aim to achieve the same results that the original application outputs but running several smaller simulations independently, following a data-centric approach to minimize intercommunication between the nodes in the Cloud. The adaptation must not penalize performance regarding the original multi-thread application, hence virtual cluster configuration and optimization tasks must be executed as well to optimise the result.

Finally, since the simulator is already production-ready, we must minimise the influence of the adaptation in the original application so that we can reuse as much code as possible. Moreover, we must provide end users with a ready-to-use solution that contains the application and the necessary environment.

The following list gives a quick overview of the goals of this project:

1. Provide a data-centric adaptation of the simulator.
2. Minimise impact in the original code.
3. Deploy on a virtual cluster on the Cloud.
4. Deliver a bundled deployment solution.
5. Analyse the effects of the migration in the simulator's overall performance.

1.3 Document Structure

This document describes the work performed during all the development process of the final application, including problem analysis, solution design and implementation and evaluation. It also contains a summary of the current state of the affairs in related research and development areas, and a sale offer proposal in case it is desired to monetise the product.

The report contains the following chapters:

- Chapter 1, *Introduction*, contains an overview of the project's context, goals and proposed approach. It additionally provides a list of acronyms and abbreviations used in the document for quick reference.
- Chapter 2, *State of the Art*, describes the cutting-edge technologies related to the project, as well as other solutions that had been proposed in previous research.
- Chapter 3, *Analysis*, characterises the given application to establish a formal set of requirements that guide the rest of the development.
- Chapter 4, *Design*, specifies the top-level solution and the models that will be implemented to comply with the requirements.
- Chapter 5, *Implementation and Deployment*, gives highlights on how the design was implemented to create the final product and how to deploy it.
- Chapter 6, *Verification, Validation and Evaluation*, specifies the test plan, the performed tests and their execution environments, and evaluates the final application against the original one in terms of performance.
- Chapter 7, *Budget*, breaks down the project's costs to build a suitable offer that includes risks and benefits for a potential sale. It also describes the product life-cycle to justify the utilised resources and planning.
- Chapter 8, *Conclusions and Future Work*, gathers key aspects that can be extracted from the project, as well as future improvements and research lines.

The document is completed with a *Bibliography* that contains the referenced resources through the whole report.

We recommend the reader to notice that all the internal references in the document are linked for easier navigation.

1.4 Definitions, Acronyms and Abbreviations

AMI *Amazon Machine Image*, Amazon's instance launch information bundle that includes OS, block device mapping, applications, platforms and permission specification.

API *Application Programming Interface*, specification of a device or software component functionality that can be accessed by a set of methods.

ARCOS *ARquitectura de COmputadoreS*, Universidad Carlos III de Madrid computer architecture group.

CLI *Command-Line Interface*, interaction method in which the user communicates with the computer by means of a sequence of text lines.

Cloud Distributed computing model based in virtualization.

EC2 *Elastic Compute Cloud*, Amazon's cloud computing pay-per-use infrastructure,

EMR *Elastic Map Reduce*, Amazon's Platform-as-a-Service offer that includes a scalable and dynamically resizeable Hadoop MapReduce distribution for virtualized clusters running on EC2.

FLOPS *FLoating-point Operations Per Second*, a standard performance measure unit.

GFS *Google File System*, Google's distributed file system for its proprietary MapReduce implementation.

GPGPU *General-Purpose Computing on Graphics Processing Units*, programming and computing paradigm that uses graphics processing devices' instead of classic central processing units.

HDFS *Hadoop Distributed File System*, Apache Hadoop's file system to support Mapreduce and other components.

HPC *High-Performance Computing*, refers to the usage of high-end computing resources to solve complex problems or analyse large amounts of data.

IaaS *Infrastructure-as-a-Service*, Cloud service model in which users pay for instance and hardware usage.

MPI *Message Passing Interface*, standardised implementation of the message-passing paradigm for process intercommunication.

MRv1 *MapReduce First Generation*, 1.x branch of the Hadoop MapReduce project.

MRv2 *MapReduce Next Generation*, 2.x branch of the Hadoop project, running on top of YARN.

NIST *National Institute of Standards and Technology*, a measurement standards laboratory from the USA.

OpenMP *Open Multi-Processing*, API that supports multi-platform shared memory multiprocessing programming.

OS *Operating System*, software system component that manages computer hardware resources and provides common services for other programs.

PaaS *Platform-as-a-Service*, Cloud service model in which users pay for ready-to-use platform and framework usage.

RAM *Random-Access Memory*, volatile information storage type.

S3 *Simple Storage Service*, web-services interface that can be used to store and retrieve data from Amazon's highly scalable storage infrastructure.

SaaS *Software-as-a-Service*, Cloud service model in which users pay for production-ready applications that can be automatically scaled to the number of requests.

SARTECO *Sociedad de ARquitectura y TEcnología de COmputadores*, Spanish non-profit association dedicated to promote research in the field of computer architecture.

SSH *Secure SHell*, cryptographic network protocol for secure data communication, remote command-line execution, and other security services.

TEPS *Traversed Edges Per Second*, performance measure unit that considers computational power and network capabilities.

UML *Universal Markup Language*, modelling language designed to provide a standard way to visualize the design of a system.

XaaS *Anything-as-a-Service*, relative to the current trend of migrating most applications to the Cloud, including databases, security frameworks, simulations, network and storage, among others.

YARN *Yet Another Resource Negotiator*, Hadoop related project that performs cluster resource management for versions that belong to the 2.x branch.

CHAPTER 2

STATE OF THE ART

2.1 Railway Simulators

Railway infrastructures designers can benefit from simulations as they can evaluate the viability and fitness of their prototypes and experimental designs without the need for their implementation. Simulators in this field operate by translating train and infrastructure features into a simplified mathematical model, which shall be able to reflect the effect of user-defined configurations in the expected real world behaviour [10].

The lines below describe the main railway simulator's families and describe some examples of production applications that belong to them [11]:

Railway dynamics This field studies the longitudinal train dynamics, which are defined as the motions of vehicles in the direction of the track, including the motion of the train as a whole and any relative motions between other vehicles [12]. An example of this sort of simulators can be found in [13]. This simulator is able to generate optimal solutions by means of artificial intelligence, while considering passenger safety and comfort.

Overhead contact line design The contact lines are the interfaces between fixed installations in the railway infrastructure and the moving vehicles [14]. Both

[15] and [14] describe simulators that help with designing these infrastructures, yet they follow different approaches. While the former is able to iterate through a predefined inventory in order to generate solutions, the latter focuses only in single design; moreover, the first option is able to find optimal solutions for limited parameters.

Power provisioning Simulators like [16] are meant to determine the location of power supply stations along the railways and their capacity. The algorithm used in this project belongs to this category.

Pantograph-catenary interaction Pantographs are the structures placed on top of trains that remain in contact with the power wire, known as catenary, in order to supply the vehicles with the necessary energy to operate [17]. Simulation tools such as Sicat Dynamic [14] and Calpe [18] evaluate the behaviour of the catenary while the pantograph is interacting with it. An interesting features of Calpe is that it is designed for high-performance environments instead of being a desktop utility like Sicat.

As shown by some of the exhibits above, nowadays simulators are not only required to provide a proper evaluation of a given solution described by the user, but they are also expected to find those suitable solutions by themselves in reasonable time. Therefore, desktop applications are starting evolve and migrate to other computing paradigms that provide higher performance and computing power.

2.2 High-Performance Computing

The term High-Performance Computing refers to the application of aggregated computing resources and parallel processing algorithms and techniques to solve complex computational problems or analyse large amounts of data. Its applications are specially focused in scientific modelling, simulations and analysis, which tend to involve large amounts of data and sophisticated algorithms. Its main goal is to solve such problems in the minimum possible time, hence supercomputers tend to be composed of multiple interconnected nodes to increase concurrency.

2.2.1 Current Supercomputers and Petascale Systems

Complex resource-intensive applications have traditionally found in high performance infrastructures the necessary hardware to to fit their high-end needs. Supercomputers constitute a canonical sample of systems that are designed to achieve the highest number of floating-point operations per second (*FLOPS*)[19]. HPC clusters and grids result from the association of a set of supercomputers under the same local network or across several administratively distributed systems, respectively; they can also be heterogeneous and gather both CPU and GPU nodes [20, 21].

Current leading systems in the Top500 rank [22] are GPU-based and capable of reporting over one quadrillion flops (a *petaflop*) under the standardised Linpack benchmark [23]. Some examples of the so-called petascale infrastructure are shown in Table 2.1, which includes the top five positions in the Top500 ranking of November 2013 [24].

System	**Performance (Pflop/s)**	**Power (MW)**	**Location**
Tianhe-2	33.86	17.81	China
Titan	17.59	8.21	USA
Sequoia	17.17	7.89	USA
K-Computer	10.51	12.66	Japan
Mira	8.59	3.95	USA

Table 2.1: *Top five positions in the Top500 ranking of November of 2013.*

Despite performance is a proper quantitative measure of an HPC system's quality, researchers, developers and end-users are increasingly aware of other critical characteristics that must be considered in order to show the actual capabilities of the tested system for the efficient execution of 3D simulations and analytics workflows, while minimizing computing cost.

The Graph500 rank [25] includes shared-memory, distributed memory and cloud benchmarks for large scale graph-oriented algorithms. Its goal is to evaluate HPC system's behaviour when approaching complex data-intensive applications, measured in traversed edges per second (*TEPS*). Current leading positions in the November of 2013 Graph500 ranking are shown in Table 2.2 [26].

System	**Performance (TTEPS)**	**Location**
Sequoia	15.36	USA
Mira	14.33	USA
JUQUEEN	5.85	Germany
K-Computer	5.52	Japan
Fermi	2.57	Italy

Table 2.2: *Top five positions in the Graph500 ranking of November of 2013.*

2.2.2 Future Goals: Green HPC and Exascale Infrastructures

Nowadays, sustainability and energy efficiency is key in the development and evaluation of HPC infrastructures. Following the Top500 philosophy, the Green500 list [27] is dedicated to rank supercomputers, but in terms of their efficiency, which is measured in performance-per-Watt.

Table 2.3 shows that current leading positions in the rank do not match any of the Top500 systems [28], and their total power consumption is significantly less that the shown by the latter. This indicates that there is still a lot of research to be done in order to reduce the gap between performance and efficiency, especially considering that supercomputers will keep increasing their target performance to reach the exascale goal [29].

System	**Performance (Tflops/W)**	**Power (kW)**	**Location**
TSUBAME-KFC	4.50	27.78	Japan
Wilkes	3.63	52.82	UK
HA-PACS TCA	3.52	78.77	Japan
Piz Daint	3.19	1,753.66	Switzerland
Romeo	3.13	81.41	France

Table 2.3: *Top five positions in the Green500 ranking of November of 2013.*

Exascale systems will become the next generation of supercomputers, capable of performing with at least one exaflop. Scientific simulations will likely benefit from the upcoming exascale infrastructures [30], however many challenges must be overcome including, processing speed and data locality and power consumption, for instance [31]; among them, energy efficiency seems to be the most limiting factor [32].

Nowadays, cheaper and lower power alternatives are on research to overcome such difficulties. For instance, low-end processors are being considered to build large scale supercomputers, and Cloud Computing appeared as a cheaper, elastic possibility to achieve the ideal situation of unlimited sustainable scalability.

2.3 Cloud Computing

Cloud Computing is a popular paradigm that relies of resource sharing and virtualization to provide the end user with a transparent scalable system that can be expanded or reduced on-the-fly.

Cloud providers operate at several levels of virtualization, which are known as service models. The NIST definition of Cloud Computing provides a description of the three basic service models [33]:

Infrastructure-as-a-Service In this model, providers offer physical or virtual resources like instances of raw virtual machines, block storage, virtual networks and disk imaging.

Platform-as-a-Service It provides a full computing platform that may include an operating system, a specific programming language execution environment, a database system or a particular web server. This allows developers to have access to a wide range of licensed software ready to create or deploy their applications, without managing the underlying hardware.

Software-as-a-Service It constitutes a higher level of abstraction in which users are provided with direct access to applications and databases without getting involved in the platform of infrastructure in which the software runs.

2.3.1 The Upcoming Anything-as-a-Service Model

The former models have been proved successful in current economies of scale and have been extended to higher levels of abstraction like Database-as-a-Service [34, 35], Network-as-a-Service [36] and Security-as-a-Service [37]. This is leading to a generic Anything-as-a-Service (XaaS) vision. Simulations are not unaware of this trend, and there are current efforts to deploy simulation services on the cloud, like [38] and [39]. With the explosion of cloud services and the Anything-as-a-Service (XaaS) model, cloud service providers have also started to offer several HPC paradigms [40]. Examples of this are MapReduce [41], MPI implementations [42], and GPGPU processing [43, 44].

The so-called Simulation-as-a-Service model and the native frameworks offered by Cloud providers seem promising for scientific simulations that are required to be scalable, but there is also a number of challenges that must be faced. The Magellan Final Report [45] exposes that scientific applications executed on the cloud suffer the overhead associated to the virtualization layer and the absence of high-bandwidth, low-latency interconnections in current virtual machines. Considering that nodes can be located in different datacenters, communication-sensitive paradigms such as MPI would show decreased performance in such environments against traditional infrastructures.

2.3.2 Trends in Cloud Migration and Adaptation Techniques

As already mentioned, scientific applications and their adaptability to new computing paradigms like the Cloud have been dragging increasing attention from the scientific community in the last few years.

The possibility to run simulations in the Cloud in terms of cost and performance was studied in [46], concluding that performance in the Abe HPC cluster and Amazon EC2 is similar –besides the virtualization overhead and high-speed connectivity loss in the cloud– and that clouds are a viable alternative for scientific applications. Hill [47] investigated the trade-off between the resulting performance and achieved scalability on the cloud versus commodity clusters; despite at the time of this work the Cloud could not properly compete against HPC clusters, its low maintenance and cost made it a viable option for small scale clusters with a minimum performance loss.

In this context, trends are naturally evolving to migrate applications to the Cloud by means of several techniques, and this includes scientific simulations as well. D'Angelo [48] describes a Simulation-as-a-Service schema in which parallel and distributed simulations could be executed transparently, which requires dealing with model partitioning, data distribution and synchronization. He concludes that the potential challenges concerning hardware, performance, usability and cost that could arise could be overcome and optimized with the proper simulation model partitioning.

In [49], Srirama et al. study how some scientific algorithms could be adapted to the Cloud by means of the Hadoop MapReduce framework. They establish a classification of algorithms according to the structure of the MapReduce schema these would be transformed to and suggest that not all of them would be optimally adapted by their selected MapReduce implementation, yet they would suit other similar platforms such as Twister or Spark. They focus on the transformation of particular algorithms to MapReduce by redesigning the algorithms themselves.

Application adaptation middlewares have also been developed to allow legacy code migration to the Cloud. For instance, in [50] a virtualization architecture is implemented by means of a Web interface and a Software-as-a-Service market and development platform. This generalist approach is suitable to provide multi-tenancy in desktop applications, but might not suffice for the resource-intensive computations required by large-scale simulations.

Finally, in [51] we find interesting efforts to move desktop simulation applications to the Cloud via virtualized bundled images that run in a transparent multi-tenant fashion from the end user's point of view, while minimizing costs. However, the virtualization middleware might affect performance since it does not take into account any structural characteristics of the model, which could be exploited to minimize migration effects or drastically affect execution times or resource consumption.

Despite Cloud Computing has proven itself useful for a wide range of scientific applications, its utility for tightly-coupled HPC applications is still under research and development, mostly because of the added communication overhead and the heterogeneous underlying hardware [52].

2.4 MapReduce

As seen in the previous section, one of the promising models that has been increasingly considered to adapt simulations to the Cloud is the MapReduce parallel computing framework, specially in cases in which data locality is key to improve performance by reducing data transmission overhead.

The applicability of the MapReduce scheme for scientific analysis has been notably studied, specially for data-intensive applications, resulting in an overall increased scalability for large data sets, even for tightly coupled applications [53].

The MapReduce paradigm [7] consists of two user-defined operations –*map* and *reduce*– and three additional phases that handle the original data, the intermediate results and the final output. Figure 2.1 shows the MapReduce dataflow and their stages, which behaves as follows:

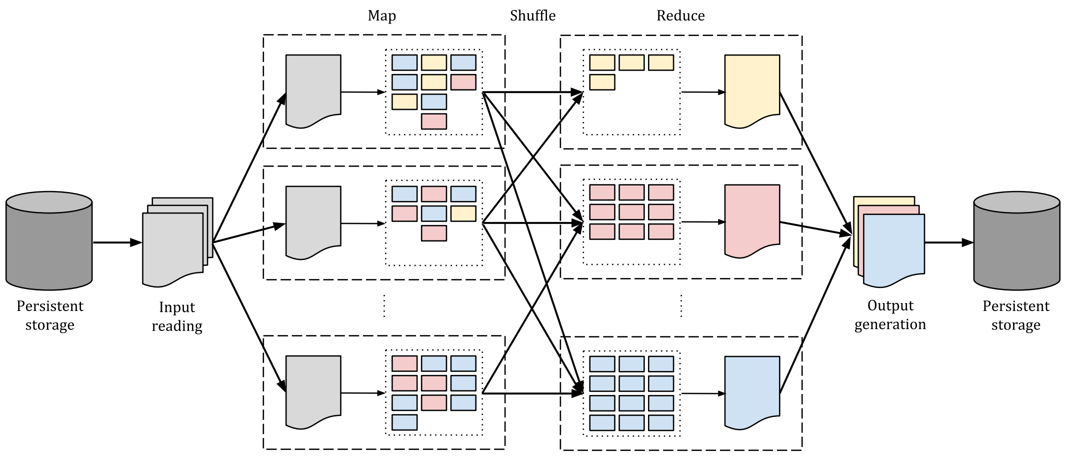


Figure 2.1: *MapReduce dataflow.*

Input reading The framework reads the input data from persistent storage and generates data chunks and assigns each of them a key, k_1, and a specific processor in the system.

Map Each input chunk is processed by a single independent map, thus spawning as many maps as k_1 values the previous phase had generated. The map's output is constituted by a set of intermediate pairs organized by a new key, k_2.

Shuffle The intermediate output is organized in lists of values for each k_2, which are assigned to a specific reducer.

Reduce As in the map function, each k_2 value list is manipulated by an autonomous reducer. The output is indexed by a new key, k_3, to allow the framework to produce the final output.

Output generation The reducer's output is collected and sorted by the framework, according to k_3, in order to write the final results to disk.

As a data-centric paradigm in which large amounts of information can be potentially processed, the *map* and *reduce* operations run independently and only rely upon the input data they are fed with. Thus, several instances can run simultaneously with no further interdependence. Moreover, data can be spread across as many nodes as needed to deal with scalability issues.

Given its popularity, there are several MapReduce implementations for distributed environments, with different capabilities and specifications. Some of the most popular are listed below:

Twister We mentioned that MapReduce performs poorly with iterative algorithms given the need of spilling intermediate data to disk between iterations. This system supports an efficient implementation of MapReduce for such applications by means of static data reusage among the tasks involved [54].

Peregrine This system eliminates the need of intermediate output writes. This feature, along with its implementation of MapReduceMerge [55], result in an overall improvement of MapReduce's capabilities. Moreover, it allows broadcasts of global-like variables and supports automatic task execution plan optimizations, so that the developer does not have to be tied to the classic MapReduce structure.

Hadoop MapReduce The most popular MapReduce implementation derived from the original Google's MapReduce and Google File System (GFS), providing an open-source alternative for both of them, is designed to be run on commodity hardware. Nowadays, Hadoop MapReduce is executed on top of the Hadoop Distributed File System (HDFS), the Hadoop Common platform and the Yet Another Resource Negotiator (YARN) resource manager, while sharing this environment with other related projects such as HBase (database system), Hive (data warehouse infrastructure) and Mahout (machine learning algorithms). It was designed to deal automatically with failures in one or several nodes of the cluster, thus resulting in a high-availability solution for data-processing infrastructures.

Spark This Hadoop-related project is focused on improving MapReduce's deficient performance regarding iterative jobs and interactive analytics [56]. Examples of these uses cases include parameter optimization on a static dataset, in which each iteration constitutes a job, and queries on large partitioned datasets, requiring a job per query. Spark's approach is based on a read-only Resilient Distributed Dataset that can be loaded into memory across many machines allowing multiple parallel operations on the same input data with no need for intermediate writes. Furthermore, Spark is not tied to the MapReduce framework and supports other programming models.

Elastic MapReduce Amazon's Elastic MapReduce (EMR) is a web service dedicated to process data on Hadoop MapReduce. It provides further advantages regarding multiple cluster manipulation, virtual cluster on-the-fly resizing, Simple Storage Service (S3) integration and HDFS support on local ephemeral storage.

Cloud MapReduce Similarly to EMR, this is another MapReduce implementation for the Cloud built on top of Cloud OS, a resource manager for the set of machines integrated to build the underlying cloud. Besides allowing incremental scalability and resizing, its most interesting feature is its decentralized and symmetric architecture in which all the nodes have the same responsibilities, even on heterogeneous environments [57].

Hadoop MapReduce

We selected Apache Hadoop [8] as our cloud migration platform given its increasing popularity and community support. Its distributed file system is a great addition to the framework, since it allows automatic load balance and includes a distributed cache that supports auxiliary read-only file storage for tasks among all nodes. Besides the former technical features, Hadoop has been increasingly adopted into cloud environments along with other MapReduce frameworks, resulting in reduced costs given its parallelism exploitation capabilities [9].

In addition, studies regarding the relationship between the Cloud and Hadoop MapReduce for scientific applications have established that performance and scalability tests results are similar between traditional clusters and virtualized infrastructures running this platform [58].

Since the original code was written in C++ and we wanted to maximize code reusage, we took advantage of the Hadoop Pipes API for C++. Despite Pipes does not allow to take full advantage of Hadoop's potential given its limited functionality, it provided all the necessary tools to execute our framework, including *map* and *reduce* interfaces, basic data type support and Distributed Cache access on job submission.

As a constantly changing technology, Hadoop evolves fast into more sophisticated and flexible versions, moving from the MapReduce-only infrastructure (versions 1.x) to the all-purpose YARN manager (versions 2.x). The following paragraphs give an overview of both platforms architectures and features, and describe their common underlying distributed file system, HDFS.

HDFS

HDFS was designed to suit Hadoop's requirements and needs. It supports the large data sets (from GB to TB of data) that Hadoop is meant to operate with, and is was built for batch read and write operations according to the MapReduce model.

In order to comply with Hadoop's fault tolerance goals, HDFS was designed for reliability and automatic recovery in case of error in one of the components of the cluster, thus providing block-level data replication and coherency mechanisms across the involved nodes.

Figure 2.2 gives an architectural overview of this file system. In it there is a *NameNode* that contains all the metadata of the system such as file-block mapping, block location among the nodes, paths and replication information. This information is requested by the client in order to perform read and write operations directly on the nodes themselves, once it has been provided with block locations. Since the *NameNode* constitutes a single point of failure in this system, it is replicated into a *Secondary NameNode* as backup; the latter would serve clients' metadata requests in case of failure.

A very interesting feature of the *NameNode* is its rack-awareness, for we can configure the network's hierarchy to let it know which nodes are topologically closer to each other. This is crucial as data exchange between closer nodes is faster and replication is more effective if performed between two different racks, since the failure of a section of the network would not affect the system's availability.

In order to remain updated, the *NameNode* is in contact with the *DataNodes*, which constitute the actual storage nodes since blocks are stored in their local disks.

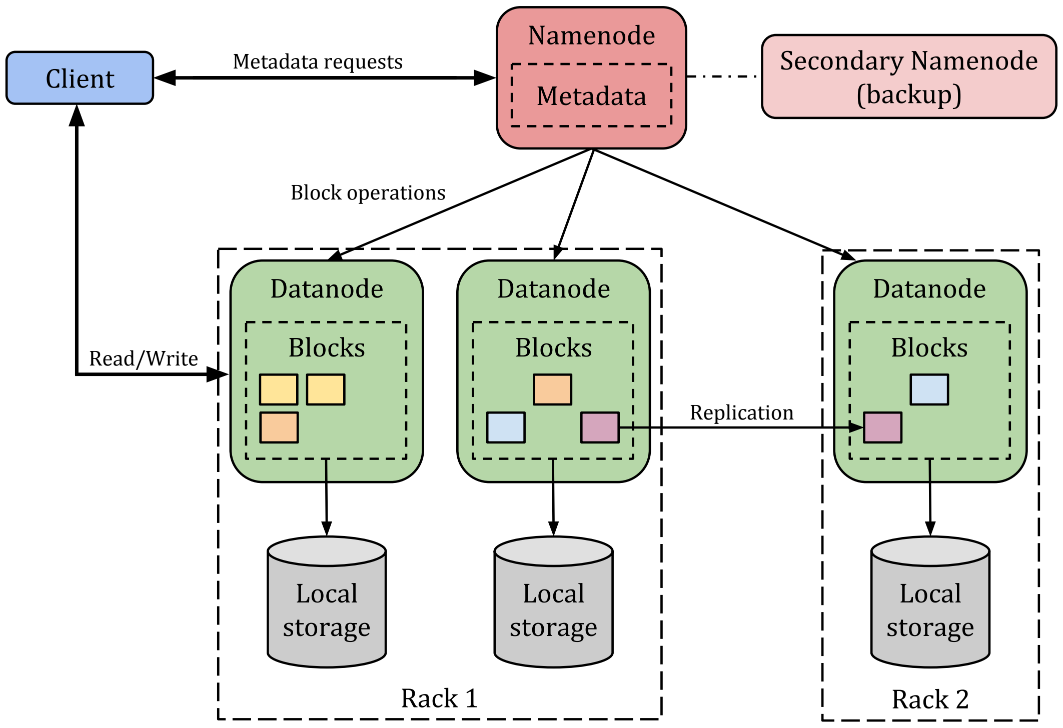


Figure 2.2: *HDFS architecture.*

This communication is performed by means of block operations that provide metadata to the *NameNode* and request operations such as replication to the *Datanode*. In case of replication, data copy is performed between *DataNodes* once it was authorized by the *NameNode*.

First Generation MapReduce Runtime (MRv1)

Hadoop 1.1.2 MapReduce's architecture –shown in shown in Fig. 2.3– is based on a fixed number of slots that can be assigned to mappers and reducers equally. These are fully managed across the nodes by a unique *JobTracker*, which also coordinates mappers and reducers, provides progress information to the client which also assigns a *TaskTracker* per node to handle local operations and slot usage.

The *JobTracker* constitutes a single point of failure and may become a bottleneck in very large clusters [59]. It also lacks resource allocation flexibility, so it is starting to become obsolete and it is being increasingly replaced by Next Generation MapReduce.

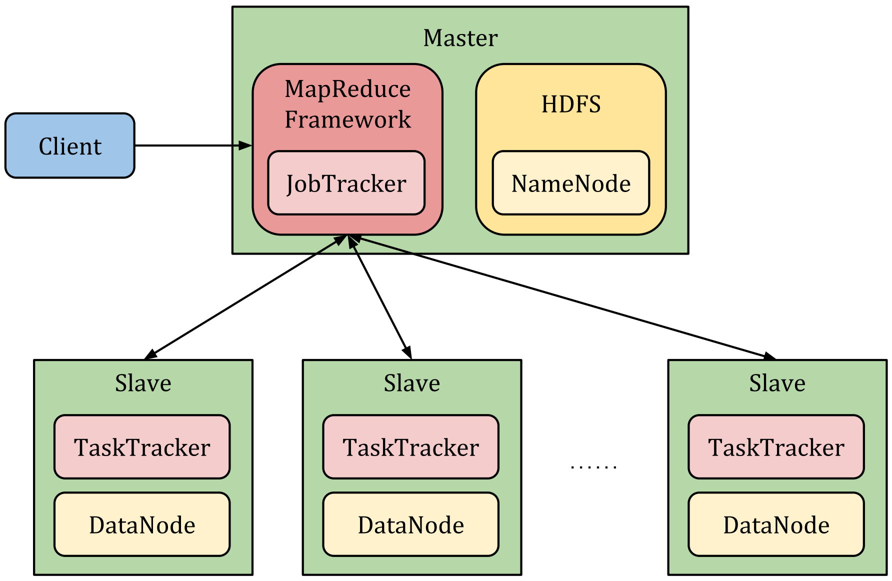


Figure 2.3: *MapReduce 1.x architecture.*

Next Generation MapReduce (MRv2) and YARN

Next Generation MapReduce (MRv2) encapsulates cluster resource management capabilities into Yet Another Resource Negotiator (YARN), leaving MapReduce-specific functionalities and configuration in an independent module. This allows to avoid some scalability issues originated in the *JobTracker* by dividing its functionality, and results in a general-purpose platform for other programming paradigms and applications [60].

Figure 2.4 shows MRv2's architecture and Hadoop's component communication with solid lines for resource managing, and dashed lines for MapReduce application control. The MapReduce functionalities handled by the previous *JobTracker* were moved to the new *ApplicationMaster*, while a *ResourceManager* is in charge of the cluster's resource management and a *HistoryServer* provides clients with information of completed jobs. *TaskTrackers* were replaced with *NodeManagers* that are responsible for the resources and container management on each node. Each container can hold a map or reduce task and can be configured regarding the available computational power, memory and input/output capabilities of the node, which yields an increased flexibility for task scheduling.

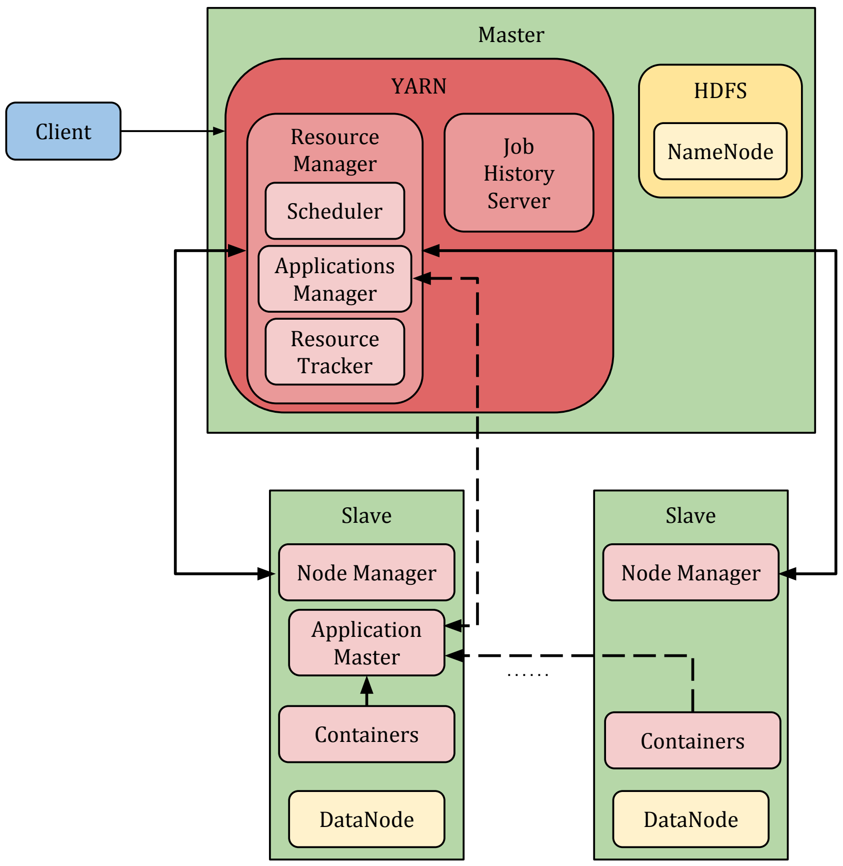


Figure 2.4: *MapReduce 2.x architecture.*

CHAPTER 3

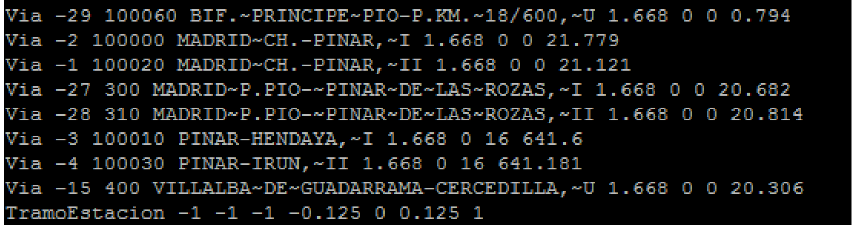
ANALYSIS

3.1 Application Description

The railway power consumption simulator transformed during this project is built to calculate the instantaneous power demand taking into account all railway elements. Its goal is to indicate whether the power provisioned by power stations is enough or not, given train position and power consumption and the infrastructure elements involved, such as tracks, overhead lines and power stations.

The simulator handles two classes of input files:

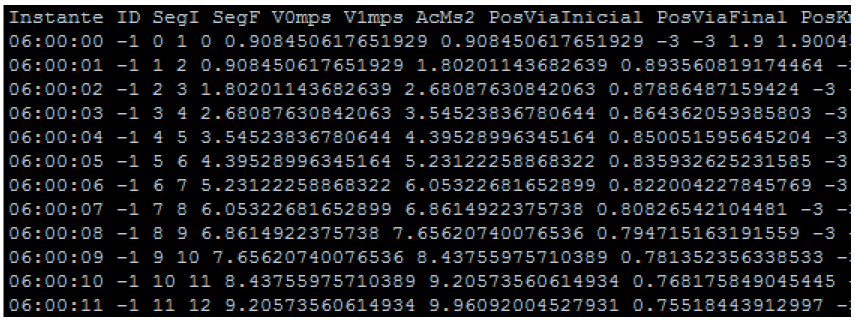
- An infrastructure specification file (Fig. 3.1) containing the initial and final time of the simulation, besides a wide range of domain-specific simulation parameters such as station and railway specifications and power supply definition.



```
Parametros Escenario1 00:00:00 32.00:00:00 1 0.1 40 15 0.5 0.5 90 1 8
Via -29 100060 BIF.~PRINCIPE~PIO-P.KM.~18/600,~U 1.668 0 0 0.794
Via -2 100000 MADRID~CH.-PINAR,~I 1.668 0 0 21.779
Via -1 100020 MADRID~CH.-PINAR,~II 1.668 0 0 21.121
Via -27 300 MADRID~P.PIO-~PINAR~DE~LAS~ROZAS,~I 1.668 0 0 20.682
Via -28 310 MADRID~P.PIO-~PINAR~DE~LAS~ROZAS,~II 1.668 0 0 20.814
Via -3 100010 PINAR-HENDAYA,~I 1.668 0 16 641.6
Via -4 100030 PINAR-IRUN,~II 1.668 0 16 641.181
Via -15 400 VILLALBA~DE~GUADARRAMA-CERCEDILLA,~U 1.668 0 0 20.306
TramoEstacion -1 -1 -1 -0.125 0 0.125 1
```

Figure 3.1: *Infrastructure input file sample.*

- A set of train movement parameters files (Fig. 3.2), structured in a time-based manner, in which each line contains the calculation of speed and distance profiles for a particular train at a specific instant regarding the infrastructure constraints, with a one second interval.



```
Instante ID SegI SegF V0mps V1mps AcMs2 PosViaInicial PosViaFinal PosK
06:00:00 -1 0 1 0 0.908450617651929 0.908450617651929 -3 -3 1.9 1.9004
06:00:01 -1 1 2 0.908450617651929 1.80201143682639 0.893560819174464 -
06:00:02 -1 2 3 1.80201143682639 2.68087630842063 0.87886487159424 -3
06:00:03 -1 3 4 2.68087630842063 3.54523836780644 0.864362059385803 -3
06:00:04 -1 4 5 3.54523836780644 4.39528996345164 0.850051595645204 -3
06:00:05 -1 5 6 4.39528996345164 5.23122258868322 0.835932625231585 -3
06:00:06 -1 6 7 5.23122258868322 6.05322681652899 0.822004227845769 -3
06:00:07 -1 7 8 6.05322681652899 6.8614922375738 0.80826542104481 -3 -
06:00:08 -1 8 9 6.8614922375738 7.65620740076536 0.794715163191559 -3
06:00:09 -1 9 10 7.65620740076536 8.43755975710389 0.781352356338533 -
06:00:10 -1 10 11 8.43755975710389 9.20573560614934 0.768175849045445
06:00:11 -1 11 12 9.20573560614934 9.96092004527931 0.75518443912997 -
```

Figure 3.2: *Train movement input file sample.*

Given the former input, simulator internals generate the electric circuit on each instant, and solves them using modified nodal analysis. Figure 3.3 [61] shows a detailed scheme of these procedures; in it we find a preparation phase in which all the required input data is read and fragmented to be executed in a predefined number of threads. Each of the resulting threads then performs the actual simulation by means of circuit solver and an electric iterative algorithm, storing in shared memory the results that will be merged in the main thread to constitute the final output files. These output files constitute a set of fourteen distinct items that contain diverse electric parameters for a wide range of infrastructure elements; each of them contains the results corresponding to the whole simulation interval, sorted by instant.

We assessed the application as experiments become larger to get a more accurate idea of its performance and analyse properly how adaptation options might behave. We designed four experiments with variations on the simulation's initial and final time and, consequently, input data volume and memory consumption. A description of these simulations is provided in Table 3.1. Cases I and II should not yield any significant load, yet simulations III and IV, are expected to reveal the application's actual performance and limitations when dealing with large simulations, if any.

Figure 3.4 shows the execution times for the proposed experiments under the same number of threads, and Fig. 3.5 [61] reflects virtual memory consumption for them as the number of threads increases. Both graphs indicate that the simulator

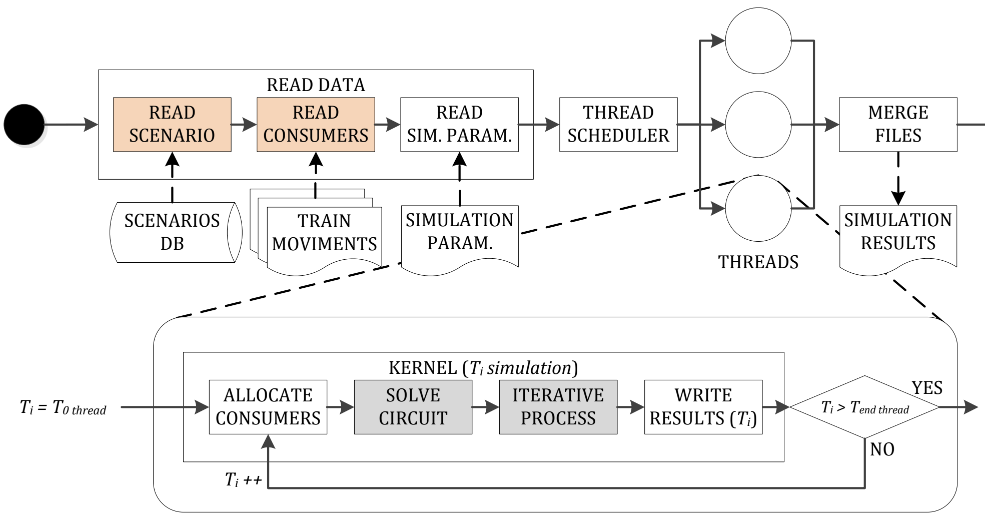


Figure 3.3: *Original application structure.*

Experiment	Simulated time (hours)	Input size (MB)
I	1	1.7
II	33	170
III	177	1228.8
IV	224	5324.8

Table 3.1: *Experiments definition.*

is not able to scale properly.

In a first place, execution times behave as expected, showing that larger cases take significantly more time to be executed; however, as the application is run in a standalone environment, there is a clear limitation in the number of threads we can use to improve performance, hence the application does not scale.

Secondly, even assuming we could use a larger number of threads, Fig. 3.5 reflects that the application's multi-thread nature might not scale properly in terms of memory consumption, as the more threads we request to increase concurrency, the more memory will be needed. This forces the host system to involve virtual memory, and this would lead to thrashing and huge performance loss.

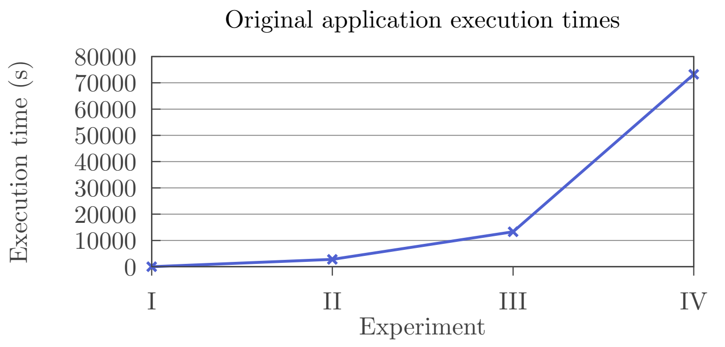


Figure 3.4: *Original application's execution times.*

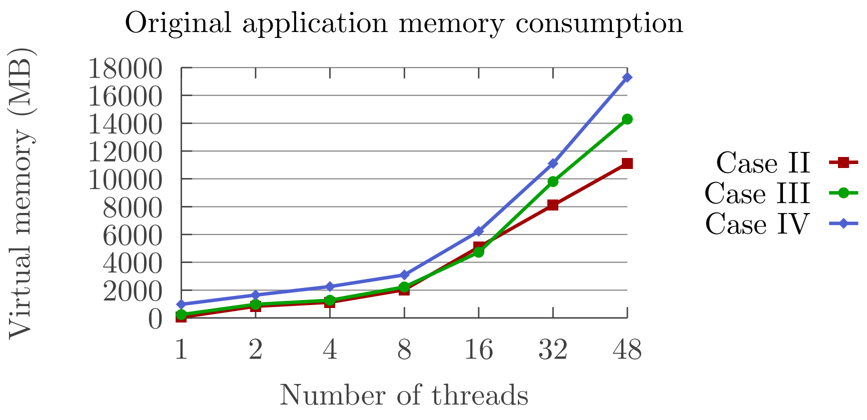


Figure 3.5: *Original application's virtual memory consumption.*

3.2 Solution Selection

This project is aimed to migrate this simulator to the Cloud in order to allow it to tackle larger experiments, following the lines of researchers that have shown these practices to be viable, beneficial and cost-effective, as seen in Sec. 2.3.2.

We could consider several options to perform the requested adaptation: bundling and virtualization of the application itself, code adaptation to build a distributed algorithm or wrapping into a parallel programming model. Since the simulator already presents a task-like structure with independent computations that can be executed in parallel, based on the input time interval and the number of requested threads, we consider a data-centric approach the most suitable solution. This is justified by the fact that we only need to divide the input data and wrap the current simulation library with an autonomous entity to partition the problem into smaller simulations that can run autonomously in different nodes.

However, migrating from a shared memory application to a data-centric model is not trivial, specially if code reusage maximisation is desired. The key idea to achieve this is to find a way to gather all the input data that a simulation partition needs, so that no further communication is needed with other subsimulations, besides post-processing tasks. From this point of view, the simulator would only see the exact information it needs for processing a limited time span, without interacting with other parallel executions that would be working on disjoint intervals. Finally, if the simulator only needs a subset of the input data, we can schedule the different executions in the actual node in which the required information is located, which constitutes the data-centric paradigm. With this proposal, we would minimise node interaction and communication overhead; furthermore, this approach does not include the virtualization overhead between the application and the execution framework that other methods add.

MapReduce, introduced in Sec. 2.4, fits naturally our data-centric and loosely-coupled adaptation proposal, hence we consider it to be the most suitable parallel programming model to use in this project. Hadoop's implementation is our selection given its popularity, community support and, mostly, its robust distributed file system that allows automatic load balance.

3.3 Requirements

In this section provide a detailed description of the application's requirements, which are limitimited to the replication of the original simulator functionality and the desired cloud deployment-related features.

Starting from the user requirements, which constitute an informal reference of the client's expected behaviour of the product, we derive the software requirements that guide the design process with specific information on the system's functionality and other characteristics. The retrieved requirements will be structured under the following schema:

1. **User requirements:**

   (a) **Capacity:** They describe the expected system functionality as in use cases.

   (b) **Restriction:** They specify constraints or conditions the system must fulfil.

2. **Software requirements:**

   (a) **Functional:**

      i. **Functional:** They describe the basic system functionality and purpose while minimizing ambiguity.

      ii. **Inverse:** These are the requirements that limit the functionality of the application to clarify its scope.

   (b) **Non-Functional:**

      i. **Performance:** Relative to the minimum required performance of the resulting system.

      ii. **Interface:** Relative to the applications user interface.

      iii. **Scalability:** Relative to the ability of the system to adapt to increasing work loads.

      iv. **Platform:** They specify the underlying software and hardware platforms in which the system will operate.

      v. **Operation:** They specify the way in which the system will perform its functionality.

Each requirement table will contain the following information:

- **Name:** Requirement name.
- **Code:** Unique code for each requirement, following these format conventions:
  - For user requirements, the format will be UR-XYY, where X indicates the requirement subtype –C, for capacity requirements, and R, for restrictions– and YY corresponds to the requirement number under its subcategory.
  - For software requirements, the format SR-X-YZZ will be used, where X indicates if it is a functional (F) or non-functional (NF) requirement, Y represents its subcategory –functional (F), inverse (I), performance (P), interface (UI), scalability (S), platform (PL) or operation (O)– and ZZ constitutes its number.

- **Type:** Indicates the category in which the requirement would be placed according to the previously described schema.
- **Origin:** Constitutes the requirement source. It might be the user, another requirement or other stakeholders involved in the project.
- **Priority:** Requirement's priority regarding implementation. The higher, the sooner it should be implemented.
- **Necessity:** It is a measure of the importance of the requirement, as its implementation can be optional or mandatory.
- **Stability:** Indicates the requirement variability through the development process.
- **Description:** Detailed explanation of the requirement.

3.3.1 User Requirements

Name	Adapted application's functionality
Code	UR-C01
Type	Capacity
Origin	User
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application's functionality shall be the same as the presented by the original application.

Table 3.2: *User requirement UR-C01*

Name	Optimization for cloud environments
Code	UR-C02
Type	Capacity
Origin	User
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application shall be optimized for its execution on a virtualized cluster in the Cloud.

Table 3.3: *User requirement UR-C02*

Name	Hadoop MapReduce as application platform
Code	UR-R01
Type	Restriction
Origin	User
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application shall run on top of Hadoop MapReduce the presented in multiple nodes.

Table 3.4: *User requirement UR-R01*

Name	Linux as underlying OS
Code	UR-R02
Type	Restriction
Origin	User
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application shall be designed for Linux systems.

Table 3.5: *User requirement UR-R02*

Name	Minimal code manipulation
Code	UR-R03
Type	Restriction
Origin	User
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application shall be reuse as much code as possible, minimizing the transformation impact in the original code.

Table 3.6: *User requirement UR-R03*

3.3.2 Functional Requirements

Name	Identical mathematical model
Code	SR-F-F01
Type	Functional
Origin	UR-C01
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application will perform the same mathematical operations that the original simulator. Hence, it shall output the same results given identical input data.

Table 3.7: *Functional requirement SR-F-F01*

Name	Identical input
Code	SR-F-F02
Type	Functional
Origin	UR-C01
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application will be provided with the same input format that the original simulator uses.

Table 3.8: *Functional requirement SR-F-F02*

Name	Identical output
Code	SR-F-F03
Type	Functional
Origin	UR-C01
Priority	High
Necessity	Mandatory
Stability	High
Description	The adapted application shall provide its output with the same format that the original simulator uses.

Table 3.9: *Functional requirement SR-F-F03*

3.3.3 Non-Functional Requirements

Name	MapReduce programming model
Code	SR-NF-PL01
Type	Platform
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	High
Description	The application will be designed following the MapReduce paradigm for a multi-node environment.

Table 3.10: *Non-Functional requirement SR-NF-PL01*

Name	Hadoop 2.2.0 version
Code	SR-NF-PL02
Type	Platform
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	High
Description	The final version of the application will run on Hadoop 2.2.0.

Table 3.11: *Non-Functional requirement SR-NF-PL02*

Name	Linux distribution Ubuntu 12.04
Code	SR-NF-PL03
Type	Platform
Origin	UR-R02
Priority	High
Necessity	Mandatory
Stability	High
Description	The final version of the application will be deployed on the Ubuntu Linux distribution, version 12.04.

Table 3.12: *Non-Functional requirement SR-NF-PL03*

Name	Virtual cluster infrastructure
Code	SR-NF-PL04
Type	Platform
Origin	UR-C02
Priority	High
Necessity	Mandatory
Stability	High
Description	The final version of the application will be deployed on a virtual cluster on top of Amazon EC2.

Table 3.13: *Non-Functional requirement SR-NF-PL04*

Name	Installation packaging
Code	SR-NF-PL05
Type	Platform
Origin	UR-R02
Priority	Medium
Necessity	Mandatory
Stability	High
Description	The final version of the application will be packed along with a pre-configured Hadoop and all the required libraries needed for a successful deployment.

Table 3.14: *Non-Functional requirement SR-NF-PL05*

Name	Platform robustness
Code	SR-NF-PL06
Type	Platform
Origin	Developer
Priority	Medium
Necessity	Mandatory
Stability	High
Description	The application shall be executed in an infrastructure with a mean time between failures greater than one week.

Table 3.15: *Non-Functional requirement SR-NF-PL06*

Name	Code manipulation limitations
Code	SR-NF-O01
Type	Operation
Origin	UR-R03
Priority	High
Necessity	Mandatory
Stability	High
Description	Developers shall not delete, insert and modify more than a 20% of the total number of code lines that belong to the original application.

Table 3.16: *Non-Functional requirement SR-NF-O01*

Name	Dedicated cluster
Code	SR-NF-O02
Type	Operation
Origin	Developer
Priority	High
Necessity	Mandatory
Stability	High
Description	The application will run in dedicated nodes that only execute the additional required software along with the simulator. This is necessary to maintain the performance measurements accurate.

Table 3.17: *Non-Functional requirement SR-NF-O02*

Name	Verification in a standalone environment
Code	SR-NF-S01
Type	Scalability
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	High
Description	The application will be tested on a single standalone machine.

Table 3.18: *Non-Functional requirement SR-NF-S01*

Name	Validation and evaluation in a physical cluster
Code	SR-NF-S02
Type	Scalability
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	Medium
Description	The application will be tested on a multi-node cluster of 10 nodes, at most.

Table 3.19: *Non-Functional requirement SR-NF-S02*

Name	Validation and evaluation in a virtual cluster
Code	SR-NF-S03
Type	Scalability
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	Medium
Description	The application will be tested on a multi-node virtual cluster of 10 nodes, at most.

Table 3.20: *Non-Functional requirement SR-NF-S03*

Name	Worst-case expected performance
Code	SR-NF-P01
Type	Performance
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	Medium
Description	The adaptation shall not exceed a 110% of the original application's execution time in the worst case, for identical input.

Table 3.21: *Non-Functional requirement SR-NF-P01*

Name	Best-case expected performance
Code	SR-NF-P02
Type	Performance
Origin	UR-R01
Priority	High
Necessity	Mandatory
Stability	Medium
Description	The adaptation should improve, at least, a 10% of the original application's execution time in the best case, for identical input.

Table 3.22: *Non-Functional requirement SR-NF-P02*

CHAPTER 4

DESIGN

4.1 Adaptation Overview

Our purpose is to divide a simulation into smaller parts that can run with the same original but on a fragment of the full data set, so that we can parallelise the executions and lower the hardware requirements for each. As seen in Sec. 3.1, the given application already divides the whole simulation time interval into smaller portions, which are processed independently by each thread. Inspired by this fact and considering the key idea of collecting only the information that a subsimulation needs, in which we insisted in 3.2, we translate the application's data flow into a sequence of MapReduce jobs. This procedure is described in the following paragraphs.

Figure 4.1 is a high-level description of the adaptation model. The original train movement files, which we know that are indexed by simulation instant, are processed by a first MapReduce job with the purpose of gathering all the trains' parameters that are involved in each specific instant. This job eliminates the need of reading the whole data set as the original simulator does before scheduling the computing threads.

The resulting intermediate output is fed to a second job that executes the actual simulation autonomously, with the help of the infrastructure file. It finally merges and writes the results to the distributed file system also in an independent way, i.e.

several nodes can perform merging tasks concurrently.

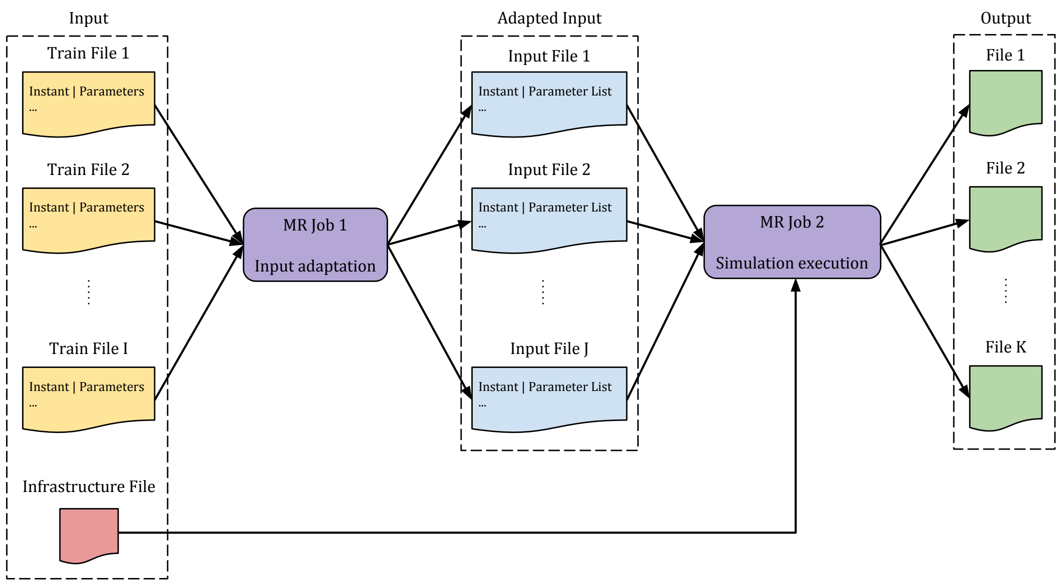


Figure 4.1: *Top-level adaptation design.*

4.2 Detailed Design

4.2.1 MapReduce Jobs Internals

Figure 4.2 shows the internal theoretical design of the MapReduce jobs. The following paragraphs explain this figure in detail, since it is key to understand why this adaptation meets our goals.

Input data adaptation

As previously stated in Sec. 3.1, the original input files are indexed by an temporal independent variable, t_i. This specific feature is critical for our adaptation, for we can execute the simulation for each instant independently as long as we manage to gather all the parameters involved in that instant and feed them to the simulator.

As we mentioned before, the first step to distribute the simulation is to transform the input files in a set of records that contain the instant, t_i, and all the parameters needed by the simulator at that specific time. This constitutes the first MapReduce

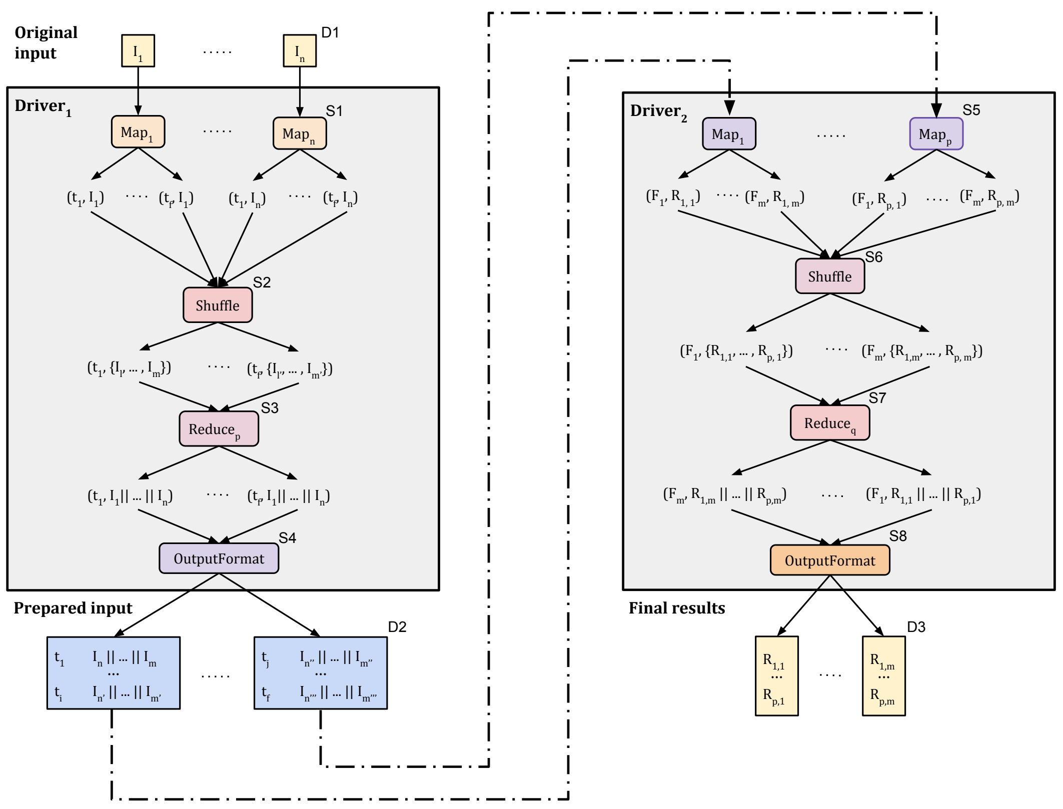


Figure 4.2: *MapReduce theoretical design for the adapted application.*

job, displayed as $Driver_1$ in the figure. We will now explain the tasks involved in this job:

***D1* – Data inputs** The MapReduce framework is in charge of dividing the original train movement files into n input splits, $I_1, \dots, I_n$, which are the input for $Driver_1$. Split records are constituted by the temporal independent variable t_i, with $i \in [1, p]$, and the parameters for the specific train at each instant. The variable t_i will act as key for the following procedures.

***S1* – Adaptation maps** Mappers read their assigned input split, parse each record and emit $(key, value)$ pairs containing the simulation index t_i and the corresponding parameters' values, resulting in a set of records $\{(t_i, I_j); i \in [1, p], j \in [1, n]\}$, as shown in the image.

***S2* – Adaptation shuffle** In this step all the values that correspond to the same key are gathered, thus resulting in p value sets, one per index to be simulated.

The intention is to provide reducers with a list of parameters per time sample that can be gathered afterwards.

***S3* – Adaptation reduce** Reducers concatenate the value list –this operation is expressed with the symbol $||$ in the figure– and output $(key, values)$ pairs where $values$ is the resulting string of the concatenation process.

***S4* – Adaptation output formatting** Finally, the pairs produced by the reducers in the previous stage are written to disc as string records that contain both the instant and all the train parameters that affect the simulation at that instant.

***D2* – Adapted data** The resulting adapted data is stored in a set of files that contain all the required information for the execution of a simulation per record in the next stage.

Independent simulations execution

Once all the data has been transformed, we are ready to execute the simulation kernel on each of the resulting records. It is important to remember that at this point this data is already distributed across the data nodes, since these files are stored on top of a block-based distributed file system that guarantees balance and forces replication. The steps involved are described below.

***S5* – Simulation maps** The string records generated by $Driver_1$ are parsed to obtain the value of the independent variable and the corresponding concatenated parameters; this information is fed to the original simulation algorithm, which is fully executed in this stage, so that we get the results for a specific key value, $R_{i,k}$. Since the simulator outputs several files, we include a file identifier F_k as key to output the file's content as value, which also has t_i injected as secondary key for further ordering tasks. Figure 4.3 shows a scheme of this procedure.

***S6* – Simulation shuffle** Simulation results are organized so that all the content that belongs to a specific F_k is listed together, with the purpose of concatenating it. This builds a set of records $\{(F_k, t_i || R_{i,k}); i \in [1, p], k \in [1, m]\}$.

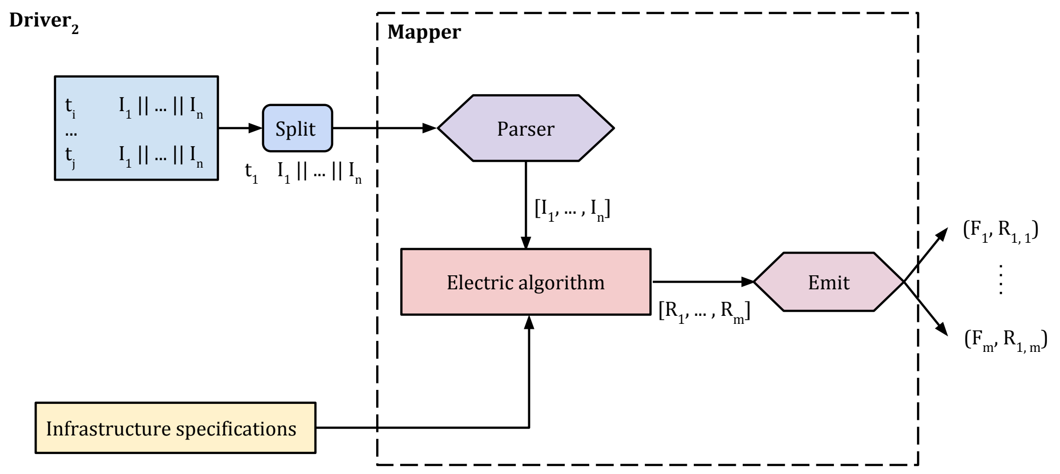


Figure 4.3: *$Driver_2$ map procedure scheme.*

***S7* – Simulation reduce** Reducers first obtain the embedded t_i for every value in the list and order the content according to it; they merge the resulting ordered content list for every F_k by concatenating it, outputting the file identifier as key and the final content as value.

***S8* – Simulation output formatting** In this stage, the reducers output is written by a custom output format in order to arrange the simulation's results files as the original application. Since Hadoop tends to write several $(key, value)$ pairs in the same file, the output format is also in charge of forcing the framework to write a file per F_k with the proper contents.

***D3* – Final data** The final data is composed of a set of files that contain the file identifier and the ordered content like the original simulation, in which each output file contains the results for the whole interval of the simulation.

CHAPTER 5

IMPLEMENTATION AND DEPLOYMENT

5.1 Application Implementation Details

The provided design we discussed in the previous chapter gives concrete guidelines on how abstract elements of the application will interact with each other. This chapter describes the next stage of the development process in which we materialize the design in a working implementation.

5.1.1 I/O Structures

We have insisted extensively in the importance of input and output files for the whole adaptation design and the actual MapReduce platform, hence we found mandatory to mention the interfaces the resulting simulator uses to read and write data.

The mappers basic input unit in the Hadoop Mapreduce framework is the *InputSplit*, which represents a set of *InputRecords* and embodies an input file block. Hadoop will launch a map task per input split and each map will process the corresponding records that are contained in it. We decided to use *TextInputFormat* (a text-based object) input records in this implementation, since the original application's input files were already in text format.

All the intermediate and final output belongs to that data type as well; however, the final results deserve special consideration as they needed special manipulation to emulate the original simulator's output. The *OutputFormat* Hadoop class is the responsible for writing data to disks in the proper format. We were able to redirect each output file's contents to a specific directory by manipulating the way in which output files' path is written by creating a user defined subclass of *MultipleTextOutputFormat*. This class would check the key value and decide under which path to write the value for every reducer output fragment; this way we have all the output information that belongs to the same file in the same location.

5.1.2 Map and Reduce Procedures

The design in 4.2 was implemented using the Hadoop Pipes API for C++ in order to reuse the original application code, written in that programming language. We provide the pseudo-code for the the resulting map and reduce procedures considering the following:

- The framework is in charge of feeding the methods with *MapContext* and *ReduceContext* as parameters. These contain diverse platform metrics, user-defined counters and all the necessary input: the input split as a record list and the result of the shuffle phase, respectively.
- The API contains all the necessary methods to access context data, represented as *get_input* for the maps and *get_next_value* for the reduce input value list.
- The *emit* function passes the platform the output pairs that were generated during the procedure.

Algorithm 1 Adaptation map algorithm

```
Require: All input data is stored in HDFS
 1: procedure ADAPTATION MAP($context$)
 2:     $line \leftarrow get\_input(context)$
 3:     if $is\_record(line)$ then                ▷ Avoids processing input file headers
 4:         $key \leftarrow get\_key(line)$
 5:         $value \leftarrow get\_value(line)$
 6:         $emit(key, value)$
 7:     end if
 8:     return
 9: end procedure
```

Algorithm 2 Adaptation reduce algorithm

```
 1: procedure ADAPTATION REDUCE($context$)
 2:     $initialize(line)$
 3:     while $has\_next\_value(context)$ do
 4:         if $is\_empty(line)$ then
 5:             $line \leftarrow next\_value(context)$
 6:         else
 7:             $line \leftarrow concatenate(line, separator, next\_value(context))$
 8:         end if
 9:     end while
10:     $emit(get\_key(context), line)$
11:     return
12: end procedure
```

Algorithm 3 Simulation map algorithm

```
Require: The adaptation job was executed and the adapted data is stored in HDFS
 1: procedure SIMULATION MAP($context$)
 2:     // Retrieve key and parameter list, which contains an item per train
 3:     $line \leftarrow get\_input(context)$
 4:     $instant \leftarrow get\_key(line)$
 5:     $parameter\_list \leftarrow get\_value\_list(line)$
 6:     // Read infrastructure
 7:     $infrastructure \leftarrow read\_infrastructure()$
 8:     // Execute simulation and output results
 9:     $result\_list \leftarrow simulate(instant, parameter\_list, infrastructure)$
10:     while $has\_next(result\_list)$ do
11:         $result \leftarrow next(result\_list)$
12:         $value \leftarrow concatenate(instant, get\_file\_content(result))$
13:         $emit(get\_file\_identifier(result), value)$
14:     end while
15:     return
16: end procedure
```

Algorithm 4 Simulation reduce algorithm

```
 1: procedure SIMULATION REDUCE($context$)
 2:     // Order the result list by instant
 3:     $initialize(map)$
 4:     while $has\_next\_value(context)$ do
 5:         $value \leftarrow next\_value(context)$
 6:         $instant \leftarrow get\_instant(value)$
 7:         $content \leftarrow get\_content(value)$
 8:         $insert(map, (instant, content))$
 9:     end while
10:     $sort(map)$
         // Concatenate results
11:     $initialize(file)$
12:     while $has\_next(map)$ do
13:         $concatenate(file, get\_content(next(map)))$
14:     end while
15:     $emit(get\_key(context), file)$
16:     return
17: end procedure
```

5.2 Platform Configuration and Deployment

As part of the iterative methodology followed, we first deployed a prototype of the application in a standalone Hadoop pseudo-cluster. This virtualized environment was the first step to verify the application's functionality against the original simulator. This first version was written and configured for Hadoop 1.1.2, which runs MRv1.

Once the code was shown correct, we migrated the application to a single-node cluster running Hadoop 2.2.0 for further testing. The final version of the application's code was deployed in a physical cluster at ARCOS dependencies and a virtual cluster running on Amazon EC2. In the following sections we provide configuration guidelines as well as deployment details for this version.

5.2.1 Hadoop Configuration

HDFS

Despite this the file system can be configured to assist in robustness, security and high availability tasks, we decided not to enable these options in this version as the application will always run under a private and controlled environment. However, we maintained the replication feature to the default value, 3, to ensure scheduled tasks would always find the data they need locally, which helps to improve performance and error recovery.

YARN

YARN's resource configuration shall be defined in Hadoop's *yarn-site.xml* configuration file. In this section we describe the necessary computations to obtain each configuration value for the purposes of this application.

First of all, we must asses our nodes to acquire the basic parameters that we need for this task, this includes the following, in a per-node basis[1]:

- Amount of RAM, R_t.
- Number of cores, C.

[1]We can configure each node independently to allow heterogeneity since the configuration file is read in each node on node manager startup.

As Hadoop shares resources with the OS, we must reserve a proper amount of memory for it. Let R_{OS} be the needed memory to run the OS, hence the memory left for the Hadoop framework in each node would be

$$R_H = R_t - R_{OS} \tag{5.1}$$

A common heuristic for container scheduling indicates that the number of containers should not exceed two containers per core. Our tests showed that this can be be increased up to four containers, at least for the first job, hence the maximum number of containers, c_{max}, can be expressed as

$$c_{max} = 4 \cdot C \tag{5.2}$$

Therefore, the minimum amount of memory we can assign to each container would be

$$R_{min} = \frac{R_H}{c_{max}} \tag{5.3}$$

Additionally, we must assign a virtual memory ratio related to the amount of physical memory, V.

Finally, we can assign these values in the configuration file. Table 5.1 matches the involved YARN configuration parameters with their proper value.

Parameter	**Value**
yarn.nodemanager.resource.memory-mb	R_H
yarn.schedule.minimum.allocation.mb	R_{min}
yarn.nodemanager.vmem-pmem-ratio	V
yarn.nodemanager.resource.cpu-vcores	C

Table 5.1: *YARN configuration parameters values related to the underlying hardware.*

MapReduce Framework

Mappers and reducers can be configured to request a specific amount of resources independently, allowing to tailor de container size to the task that will be executed and the Java heap size limit. We overwrite these configurations in the application execution script at run time so that we can manipulate these values without the need to relaunch the MapReduce framework.

Table 5.2 reflects an heuristic configuration planning for both jobs as result of our experiments with large test cases. The first job does not perform any resource-intensive task, so we can maximise the number of containers by requesting the minimum container size, R_{min}, for both map and reduce. However, the second job executes the memory-bound algorithm in the map stage and makes use of a very large buffer to store the file contents in the reduce phase. This indicates that both tasks will require a larger container in terms of memory.

Parameter	**Job 1**	**Job 2**
mapreduce.map.memory.mb	R_{min}	$min\{3 \cdot R_{min}, R_H\}$
mapreduce.map.java.opts	$0.8 \cdot R_{min}$	$0.8 \cdot min\{3 \cdot R_{min}, R_H\}$
mapreduce.reduce.memory.mb	R_{min}	$min\{6 \cdot R_{min}, R_H\}$
mapreduce.reduce.java.opts	$0.8 \cdot R_{min}$	$0.8min\{6 \cdot R_{min}, R_H\}$
mapreduce.job.reduces	c_{max}	$\lfloor \log_2 c_{max} \rfloor$

Table 5.2: *MapReduce configuration parameters values related to the underlying hardware, optimized for large test cases.*

Regarding the number of reducers, the more we have in the first job the more input splits will be generated, increasing the concurrency for the following job. However, a large number of reducers can significantly affect performance if the amount of data to be processed and the gained performance in the following step do not counterbalance the framework overhead. In the second job, a low number of reducers might result in excessive disk spilling, thrashing and container crashing in case resource usage exceeds the limits.

Other Parameters

We forced reducers to wait for at least the 85% of the mappers to finish before start processing their output in order to minimize the shuffle overload and maximize the available resources at the map phase, which is especially relevant in the second job. We also made use of Hadoop's Distributed Cache to allow read-only access to the common infrastructure file for every node.

5.2.2 Deployment

Figure 5.1 shows the final deployment scheme. The client machine is expected to have the application's input data in its own database; that input is copied HDFS on application execution. The simulation is launched by the client through a CLI SSH session with the master node; the execution script is the only component of the system that interacts with both the client and the Hadoop platform. Finally, Hadoop components handle job execution and data transmission during the simulation, as usual.

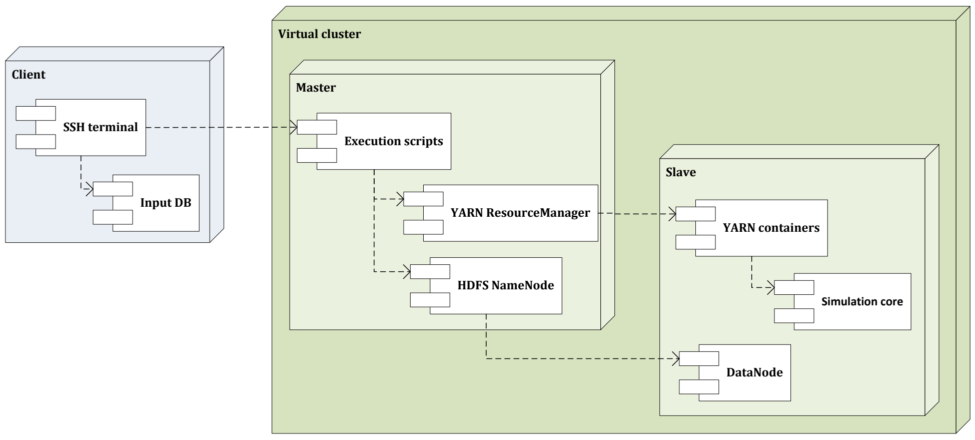


Figure 5.1: *Deployment UML diagram.*

One of the application's requirements is to provide a straightforward deployment method for end-users. We decided to include Hadoop MapReduce, the transformed simulator, the configuration files and a set of scripts to interact with the system into a snapshot of a working preconfigured installation of the final version running on the Cloud. This snapshot was exported to an AMI that can be loaded directly into any raw instance. This AMI is available on Amazon EC2 and includes de following bash scripts:

- A startup script to automatically configure the platform in every node according to the user-defined configuration. It also launches all HDFS, YARN and Mapreduce necessary services.
- An application workflow script that handles the MapReduce job pipeline and interaction with Hadoop components. It includes experiment definition, input

upload to HDFS from the client, job configuration, the transformation and simulation execution and output retrieval to the client's local storage.

- A shutdown script to clean the system of temporal files, intermediate data old logs and HDFS folders. After cleaning, the script finalises Hadoop services.

CHAPTER 6

VERIFICATION, VALIDATION AND EVALUATION

This chapter describes the verification and validation tests that were executed to corroborate the system's proper functioning and its compliance against the requirements defined in Sec. 3.3, respectively. It shows the test cases specifications and their corresponding results and, as part of the objectives of this thesis, contains an extensive discussion on the performance evaluation of the transformed application –running in both a physical and virtual cluster– versus the original simulator.

6.1 Test Plan Specification

As introduced in Sec. 5.2, we followed an iterative and incremental deployment approach to isolate the errors resulting from the implementation of the actual application code from deployment and configuration issues. The tests and evaluation experiments performed to the system will be specified according to these implementation stages to provide a clear idea of how the application was developed over time; these are summarized in Tab. 6.1 and detailed as follows:

***I* – Pseudo-cluster stage** The objective of this step if to verify the initial prototype that resulted from a basic implementation of the MapReduce design provided in Sec. 4.2 on a simplified working context; therefore, we deployed the application on Hadoop MRv1 1.1.2 in pseudo-cluster mode on a standalone desktop PC. This stage was critical to ensure that we were getting the proper output results from the adapted simulator before moving to more complex environments and dealing with large experiments that would be very hard to verify against the original application. Further optimizations in the map and reduce processes of both jobs were also tested in this environment before replicating them in the final application.

***II* – Single-node physical cluster stage** Once we proved the simulator adaptation was working effectively, we migrated to Hadoop MRv2 2.2.0 to benefit from YARN's resource management capabilities and flexibility. This system was deployed in a single-node real cluster to simplify the execution of the same black-box tests that were performed in the pseudo-cluster and set a milestone for the application's subsequent distribution, since from this point we can assume the simulator is verified and its internal behaviour will not be affected by parallelisation across different nodes. Furthermore, we performed the first experiments with large data sets to ensure the application's robustness.

***III* – Multi-node physical cluster stage** In this stage the simulator was properly verified and validated against its expected functionality. This step constitutes the link between running the application in a local cluster and moving it to the Cloud, since virtualization should not affect a properly configured platform nor the actual application. In this stage we also verified the application's ability to communicate between several nodes and achieve the expected result, and we evaluated its scalability under the configuration plan described in Sec. 5.2.1 for the same experiments that we used to analyse the original application.

***IV* – Multi-node virtual cluster stage** Finally, the final application was deployed and configured for a virtual cluster running in the Cloud. Here we performed the same tests and evaluations that we executed in the physical cluster to validate the pending requirements and compare their performance. Additionally, we built a ready-to-use AMI for easy deployment as requested, and verified the cluster's functionality when launching the application from it.

Test type	**Stage**	**Goal**
White box	I	Verify that the possible paths in the code function as expected.
Functional	I	Validate that the application covers the functionality described in the requirements.
Black box	I, II	Verify that the methods provide the expected output for a given input.
Integration (code)	I, II	Verify that all code modules are visible and function properly after integration.
Integration (deployment)	II, III, IV	Verify that all the elements in the system are visible, communicate properly with each other and function properly after integration.
Performance	II, III, IV	Validate system's performance requirements.
End-to-end	IV	Validate that the full system covers the requirements in its real environment.

Table 6.1: *Tests types performed at each stage of the development process.*

6.2 Execution Environments

As we explained in the previous section, the final application is meant to be executed in the Cloud on top of a virtual cluster running Hadoop. Additionally, we have to measure its performance in a physical cluster to have a reference of the execution times without the virtualization and communication overhead introduced by cloud migration. These two environments have significant differences regarding the resources they possess. In this section we will describe the execution environments that we considered for the evaluation phase, and build a platform configuration plan for each following the guidelines in 5.2.1.

Additionally, for the sake of completeness, we include the hardware settings of the node in which the original application was analysed in terms of performance and memory consumption. This information is relevant for further evaluation of the resulting application.

6.2.1 Original Application Evaluation Node

We tested the original multi-thread application's memory consumption and performance on one of the nodes of the ARCOS Tucán. It consisted of 24 Xeon E7 cores, one local disk and 110GB of RAM.

6.2.2 Physical Cluster

We recurred to the ARCOS Tucán cluster to test the application in a physical distributed setting. Our typical test environment is composed of one node identical to the used to assess the original application. The intention of using only one node is to avoid variations that may arise from heterogeneous configuration, resource differences or network latency [46]. This isolation favours the multi-thread application, which is especially designed to perform in standalone environments, yet it allows to focus the evaluation phase on the actual limiting factors that may affect scalability in large test cases like I/O, memory consumption and CPU usage.

Nevertheless, it is important to remark that this decision does not limit the scalability of the application and that other tests –besides performance measurements– were executed on a higher number of nodes to validate scalability requirements.

The configuration for this environment resulted from applying the recommended guidelines for $R_t = 110$, $R_{OS} = 5$, $V = 2.1$ and $C = 24$. The results for the YARN framework are detailed in Tab. 6.2, while the MapReduce job configurations are shown in Tab. 6.3.

Parameter	**Value**
yarn.nodemanager.resource.memory-mb	107520 (105 GB)
yarn.schedule.minimum.allocation.mb	1126 (1.1 GB)
yarn.nodemanager.vmem-pmem-ratio	2.1
yarn.nodemanager.resource.cpu-vcores	24

Table 6.2: *YARN configuration parameters values for the physical cluster execution environment.*

Parameter	**Job 1**	**Job 2**
mapreduce.map.memory.mb	1126 (1.1 GB)	3379 (3.3 GB)
mapreduce.map.java.opts	922 (0.9 GB)	2662 (2.6 GB)
mapreduce.reduce.memory.mb	1126 (1.1 GB)	6758 (6.6 GB)
mapreduce.reduce.java.opts	922 (0.9 GB)	5407 (5.3 GB)
mapreduce.job.reduces	96	6

Table 6.3: *MapReduce configuration parameters values for the physical cluster execution environment.*

6.2.3 Virtual Cluster

The selected Amazon EC2 instances were one general purpose *m1.medium* node as dedicated master and five memory optimized *m2.xlarge* machines as slaves, resulting in a total of ten CPUs, five local disks and 85.5GB of RAM available for job execution. Tables 6.4 and 6.5 contain the analogous computations that were calculated for the physical cluster, but considering $R_t = 17.1$, $R_{OS} = 2.1$, $V = 2.1$ and $C = 2$.

Parameter	Value
yarn.nodemanager.resource.memory-mb	15360 (15 GB)
yarn.schedule.minimum.allocation.mb	1843 (1.8 GB)
yarn.nodemanager.vmem-pmem-ratio	2.1
yarn.nodemanager.resource.cpu-vcores	2

Table 6.4: *YARN configuration parameters values for the physical cluster execution environment.*

Parameter	Job 1	Job 2
mapreduce.map.memory.mb	1843 (1.8 GB)	5530 (5.4 GB)
mapreduce.map.java.opts	1434 (1.4 GB)	4403 (4.3 GB)
mapreduce.reduce.memory.mb	1843 (1.8 GB)	1160 (10.8 GB)
mapreduce.reduce.java.opts	1434 (1.4 GB)	8806 (8.6)
mapreduce.job.reduces	8	3

Table 6.5: *MapReduce configuration parameters values for the physical cluster execution environment.*

6.3 Tests and Results

In this section we provide detailed formalisations of the performance and end-to-end tests. We focus on end-to-end tests for they summarise the verification tests conducted in previous development stages of this application, as they validate the whole functionality of the final version of the system. We also describe the performance tests in order to asses the system's performance on the different execution environments for further evaluation.

The tables below are composed of the following fields:

- **Name:** Test name.
- **Code:** Unique code for each test in the format XXX-YY, where XXX indicates the test type –EET for end-to end tests and PT for performance tests– and YY indicates the test number in that category.
- **Type:** Test type.

- **Requirement:** Constitutes the requirement that the test validates or relates to.
- **Environment:** Execution environment, among the ones described in the previous section, in which the test was performed.
- **Objective:** Motivation to perform the test, whether it is to force a particular behaviour in the system or corroborate its expected response.
- **Precondition:** System status and conditions that must be fulfilled before executing the test.
- **Procedure:** Steps that need to be executed to conduct the test.
- **Postcondition:** System status after the test was executed.
- **Acceptance:** Criteria to follow to consider the test as passed.
- **Evaluation:** Whether the application passes the test or not according to the acceptance criteria.

6.3.1 Performance Tests

Name	Performance with large experiment.
Code	PT-01
Type	Performance
Requirement	SR-NF-P03
Environment	Single-node cluster
Objective	Verify that the application resists heavy workloads.
Precondition	1. The user is logged in the master/slave node by SSH. 2. The user has the input data in the node. 3. The Hadoop framework is running.
Procedure	1. Modify the script to indicate experiment name and input data location in the node. 2. Run the simulation.
Postcondition	1. The application was executed and the output data remains in the node.
Acceptance	The simulation performs with no errors.
Evaluation	Passed.

Table 6.6: *Performance test PT-01*

Name	Performance with large experiment.
Code	PT-02
Type	Performance
Requirement	SR-NF-P03
Environment	Multi-node cluster
Objective	Verify that the application resists heavy workloads in a distributed setting.
Precondition	1. The user is logged in the master node by SSH. 2. The user has the input data in the node. 3. The Hadoop framework is running.
Procedure	1. Modify the script to indicate experiment name and input data location in the node. 2. Run the simulation.
Postcondition	1. The application was executed and the output data remains in the node.
Acceptance	The simulation performs with no errors.
Evaluation	Passed.

Table 6.7: *Performance test PT-02*

Name	Performance with best case (large) experiment.
Code	PT-03
Type	Performance
Requirement	SR-NF-P03
Environment	Cloud
Objective	Verify that the application resists heavy workloads in a virtualized setting and validate performance constraints.
Precondition	1. The user is logged in the master node by SSH. 2. The user has the input data in the node. 3. The Hadoop framework is running.
Procedure	1. Modify the script to indicate experiment name and input data location in the node. 2. Run the simulation and measure execution time.
Postcondition	1. The application was executed and the output data remains in the node.
Acceptance	The simulation performs with no errors and the execution time is less than 65900s.
Evaluation	Passed with an execution time of 55064s.

Table 6.8: *Performance test PT-03*

Name	Performance with worst case (very small) experiment.
Code	PT-04
Type	Performance
Requirement	SR-NF-P02
Environment	Cloud
Objective	Validate performance constraints.
Precondition	1. The user is logged in the master node by SSH. 2. The user has the input data in the node. 3. The Hadoop framework is running.
Procedure	1. Modify the script to indicate experiment name and input data location in the node. 2. Run the simulation and measure execution time.
Postcondition	1. The application was executed and the output data remains in the node.
Acceptance	The simulation performs with no errors and the execution time is less than 10s.
Evaluation	Failed with an execution time of 97s due to framework overhead. The next experiment in size passes the test, See Sec. 6.4 for a discussion on this result.

Table 6.9: *Performance test PT-04*

6.3.2 End-to-End Tests

Name	Deployment via AMI.
Code	EET-01
Type	End-to-end
Environment	Cloud
Requirement	SR-NF-PL05
Objective	Validate that the end user is able to successfully deploy the packaged application and platform.
Precondition	1. The user is logged in a client machine able to connect to EC2 by SSH. 2. The user has been provided with the AMI.
Procedure	1. Launch the desired number of EC2 instances. 2. Select the AMI as base image. 3. Login via SSH to the master node. 4. Execute the startup script
Postcondition	1. The Hadoop platform is properly launched in all the user's instances
Acceptance	Hadoop related processes are properly launched in the master and slaves with no errors.
Evaluation	Passed.

Table 6.10: *End-to-end test EET-01*

Name	Simulation execution via AMI.
Code	EET-02
Type	End-to-end
Requirement	SR-NF-PL05
Environment	Cloud
Objective	Validate that the end user is able to successfully run the packaged application.
Precondition	1. The user is logged in a client machine able to connect to EC2 by SSH. 2. The user has the input data in the client. 3. The startup script has been executed.
Procedure	1. Modify the script to indicate experiment name and input data location in the client. 2. Run the execution script.
Postcondition	1. The application was executed and the output data was transferred to the client.
Acceptance	The data is properly transmitted and the simulation was executed with no errors.
Evaluation	Passed.

Table 6.11: *End-to-end test EET-02*

Name	Simulation functionality validation.
Code	EET-03
Type	End-to-end
Requirement	SR-F-F01, SR-F-F02, SR-F-F03
Environment	Cloud
Objective	Validate that the adapted application functionality is the same than the original application.
Precondition	1. The user is logged in a client machine able to connect to EC2 by SSH. 2. The user has the original simulator data in the client. 3. The startup script has been executed.
Procedure	1. Modify the script to indicate experiment name and input data location in the client. 2. Run the execution script.
Postcondition	1. The application was executed and the output data was transferred to the client.
Acceptance	The output data corresponds to the output of the original simulation, holding the same results.
Evaluation	Passed.

Table 6.12: *End-to-end test EET-03*

Name	Physical cluster scalability.
Code	EET-04
Type	End-to-end
Requirement	SR-NF-S02
Environment	Multi-node cluster
Objective	Validate that the adapted application is able to scale to the required number of cluster nodes.
Precondition	1. The user is logged in a client machine able to connect to EC2 by SSH. 2. The user has the original simulator data in the client. 3. The startup script has been executed.
Procedure	1. Modify the script to indicate experiment name and input data location in the client. 2. Run the execution script.
Postcondition	1. The application was executed and the output data was transferred to the client.
Acceptance	The simulation runs with no errors and all the reserved nodes were involved in the computations.
Evaluation	Passed.

Table 6.13: *End-to-end test EET-04*

Name	Virtual cluster scalability.
Code	EET-05
Type	End-to-end
Requirement	SR-NF-S03
Environment	Cloud
Objective	Validate that the adapted application is able to scale to the required number of virtual cluster nodes.
Precondition	1. The user is logged in a client machine able to connect to EC2 by SSH. 2. The user has the original simulator data in the client. 3. The startup script has been executed.
Procedure	1. Modify the script to indicate experiment name and input data location in the client. 2. Run the execution script.
Postcondition	1. The application was executed and the output data was transferred to the client.
Acceptance	The simulation runs with no errors and all the reserved nodes were involved in the computations.
Evaluation	Passed.

Table 6.14: *End-to-end test EET-05*

6.4 Performance Evaluation

As we already discussed in Section 3.1, the original multi-thread application's memory usage suggests a lack of scalability in a standalone environment. We will now analyse whether the adapted simulator behaves as expected in relation to performance and scalability by examining the results we obtained in the performance tests, for the same experiments used to analyse the original application. We will also discuss the effects of each of its execution phases in the overall execution time. These results are summarised in Fig. 6.1.

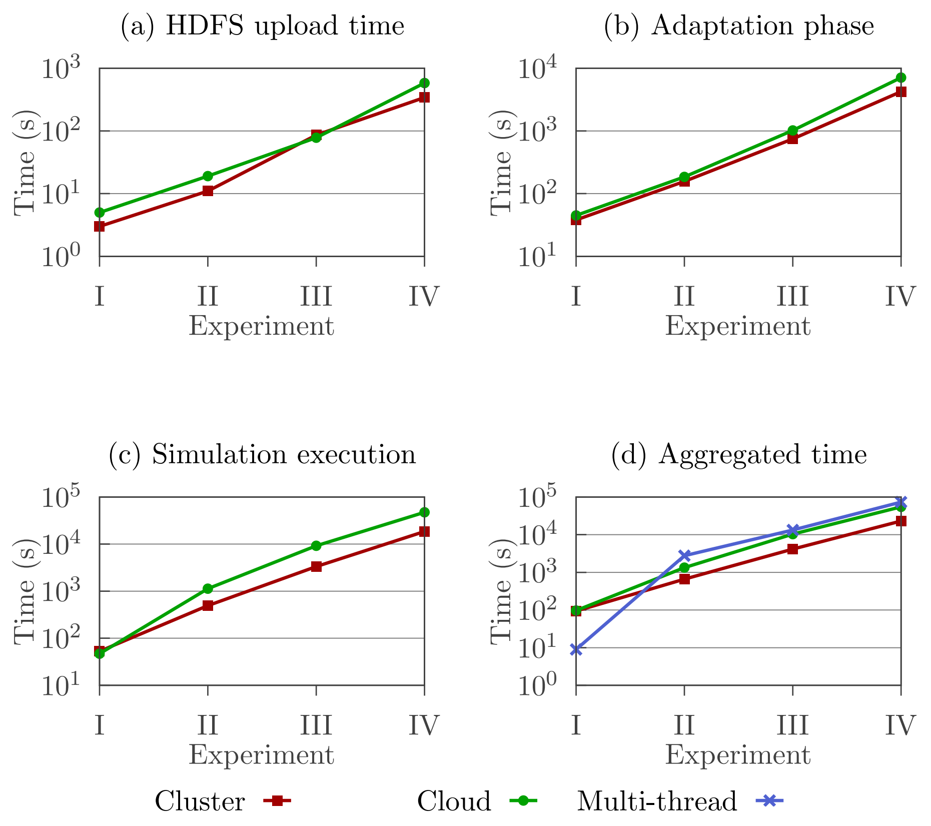


Figure 6.1: *Time results for the adapted and original application, in logarithmic scale.*

6.4.1 Stage Analysis

There are three critical phases in the execution of the adapted simulator: the input upload to HDFS from the client, the adaptation job execution and the parallel simulation execution. We refer to each phase according to its numeration in Fig.6.1; these are the observations we extract from the results:

(a) HDFS upload time

Train movement files compose the input for the first MapReduce job, thus must be previously uploaded to HDFS. This process is time-consuming since replication, block splitting and balance among the nodes must be accomplished, which may yield a considerable impact in the overall performance. In (a) we can see that this previous task does not add up a significant overhead; moreover, we can see that EC2's high-end network capabilities seem to compensate the added network latency between the nodes, so data distribution time is not significantly affected by adding more nodes.

(b) Adaptation phase

The data adaptation phase –graph (b)– takes longer on the cloud as a result of the selected instances characteristics, because the ratio between their memory and number of cores favours the memory-bound simulation execution phase instead of this one, and the configuration between jobs remains the same. This is also supported by the single-node cluster results, in which we notice a higher performance given the larger number of cores that can execute more mappers simultaneously at this stage.

(c) Simulation execution

The algorithm execution stage, (c), is the most determinant phase in the whole process for all of the execution environments, since it is most time-consuming step in the whole workflow. As we see in the figure, the application migration from the single-node environment to the virtual cluster supposed a performance hit for the latter; this is reasonable since the total resources in the cloud were considerably less than the held by the real cluster. Nevertheless, we notice that the execution times evolve in the same way for large test cases –i.e. the graphs show the same tendency in both cases–; this indicates that the virtualization and node intercommunication overheads did not affect performance significantly (recall that the physical cluster measurements were obtained from a single-node cluster to permit this comparison).

We consider the output retrieval to be optional so we do not take it into account for evaluation purposes; moreover, the input upload stage already illustrates the data transmission overhead resulting from virtualization to a remote cluster.

6.4.2 Final Evaluation

We will now evaluate the overall performance of the application by aggregating the execution times obtained in the previous stages and comparing them with the original application. This results in the graph labelled as (d) in 6.1. Additionally, we provide a speed-up graph in Fig. 6.2.

In the figure we observe that the obtained performance with MRv2 in both the single-node cluster and the elastic cloud shows remarkably better results than the original multi-thread application. We must note that the time representation is shown in logarithmic scale in Fig. 6.1, hence the difference is more significant that what it may seem at a glance. Moreover, we could further improve performance by increasing the number of available job containers, which can be achieved by either getting larger instances or adding more machines to the virtual cluster.

The shared memory simulator's results might be caused by the bottleneck constituted by the physical memory and the disk; the latter is particularly critical, as all threads write their results to disk while they perform their computations in the original simulator. The smallest experiment is an interesting exception, since it reflects how the MapReduce framework's overhead significantly affects the time taken to complete such a small simulation compared to the original application benchmark.

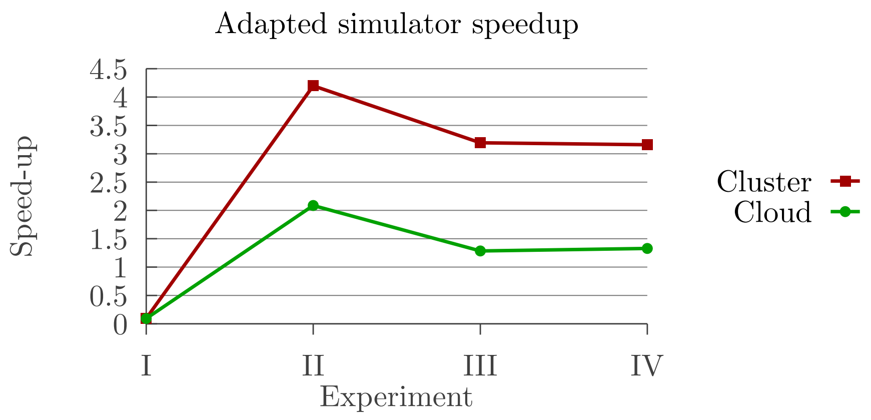


Figure 6.2: *Adapted application speed-up against the original simulator.*

In conclusion, the results obtained in these experiments prove that our proposed data-centric design is suitable for cloud migration of the original simulator. In addition, a side effect of this adaptation is that we overcome the large memory requirements and heavy I/O usage of the multi-thread application; this results in an attractive and stable performance gain for large test cases.

CHAPTER 7

BUDGET

This chapter provides some highlights in the project's development planification and life-cycle. Additionally, it details the costs generated by this project –including staff, equipment and materials– and shows an illustrative offer proposal that one could effectively sell to an interested client, which reflects potential risks and benefits.

7.1 Life Cycle

As this project is the result of a research initiative of the Computer Science Department, many unexpected difficulties emerged during the implementation, deployment and testing stages of its development.

To alleviate these issues, we followed a spiral development model that allowed us to refine our system for each target platform incrementally, starting from a standalone prototype and reaching the virtualized cluster deployment goal with a production-ready application, after evaluating the application in both a single-node and a fully distributed physical cluster. Figure 7.1 summarizes the former stages.

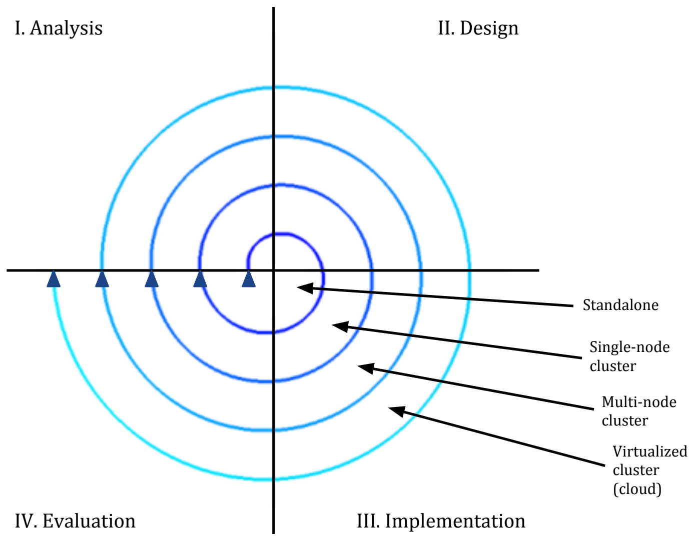


Figure 7.1: *Application development life cycle.*

7.2 Project Costs

In this section we breakdown the project's costs. Table 7.1 summarises key characteristics of the project that will be considered during the cost computation process.

Project information	
Title	Adaptation, deployment and evaluation of a railway simulator in cloud environments
Author	Silvina Caíno Lores
Department	Computer Science
Start date	1st of June of 2013
Duration	12 months
Indirect costs ratio	20%
Total budget	43,107.34€

Table 7.1: *Project information.*

Table 7.2 reflects the costs originated from staff hiring; it is built considering that:

- 1 man-month = 131.25 hours
- Maximum annual dedication: 12 man-month (1575 hours)
- Maximum annual dedication for Universidad Carlos III research personnel: 8.8 man-month (1155 hours)

Personnel				
Name and surname	**Category**	**Dedication (man-month)**	**Man-month cost(€)**	**Cost (€)**
Silvina Caíno Lores	Engineer	9	2,152.25	19,370.25
Alberto García Fernández	Research engineer	3	4,379.21	13,137.63
Total				32,507.88

Table 7.2: *Direct personnel costs.*

Table 7.3 shows that the personnel is aware and conform with the previous cost specifications[1].

Personnel consent		
Name and surname	**Identification number**	**Signature**
Silvina Caíno Lores	-	-
Alberto García Fernández	-	-

Table 7.3: *Personnel consent declaration.*

Table 7.4 describes the direct costs that emerged from equipment acquisition and usage. The chargeable cost, C, is obtained by computing $C = \frac{d}{D} \cdot c \cdot u$, where d is the number of months during which the equipment was utilised, D it the deprecation period in months, c is the equipment cost, and u is the project dedication ratio.

Equipment					
Description	**Cost (€)**	**Project dedication (%)**	**Dedication (months)**	**Deprecation period (months)**	**Chargeable cost (€)**
Desktop PC	850.00	100	12	60	170.00
Laptop PC	700.00	50	6	60	35.00
ARCOS Tucán cluster	35,000.00	20	8	40	1,400.00
Amazon EC2	75.00	100	4	40	7.50
Printer	80.00	15	12	60	2.40
Total					**1,614.90**

Table 7.4: *Direct equipment costs.*

Finally, Tab. 7.5 represents other direct costs that do not belong to any of the previous categories, including daily personnel expenses, commuting costs and office supplies.

As Tab. 7.6 indicates, the total cost of the project amounts to 43,107.34€.

[1]Identification numbers and signatures are omitted to protect staff's privacy.

Other direct costs		
Description	**Company**	**Chargeable cost (€)**
Expenses	Universidad Carlos III	1,200.00
Commuting	Universidad Carlos III	540.00
Office supplies	Universidad Carlos III	60.00
Total		**1,800.00**

Table 7.5: *Other direct costs.*

Costs summary	
Personnel	32,507.88€
Amortization	1,614.90€
Operating expenses	1,800.00€
Indirect costs	7,184.56€
Total	**43,107.34€**

Table 7.6: *Costs summary.*

7.3 Project Offer Proposal

A sample offer proposal is detailed below in Tab. 7.7. It includes estimated risks, expected benefits and taxes along with the computed total cost of the project. According to the previous criteria, the final amount for this project in case of sale to a third-party client is *seventy-one thousand nine hundred and eighty-one* Euro (71,981€).

Offer proposal			
Concept	**Increment (%)**	**Partial value (€)**	**Aggregated cost (€)**
Project	-	43,107.34	43,107.34
Risk	20	8,621.47	51,728.81
Benefits	15	7,759.32	59.488.13
Taxes	21	12,492.51	71,980.64
Total			**71,980.64**

Table 7.7: *Offer proposal breakdown.*

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this chapter we will specify the conclusions extracted from the whole development process and the resulting application evaluation. We cover positive aspects, such as the objectives met and contributions that are derived from this project, and negative aspects, like issues in the current version of the application and the difficulties we had to overcome in the different phases of the project.

To conclude, we detail several interesting research lines and improvements for future work.

8.1 Met Objectives and Other Positive Aspects

Scientific simulations, like the one we were provided with, have been traditionally related with HPC infrastructures such as supercomputers, clusters and grids. However, their resource-intensive nature limits the execution possibilities and achieved performance for large test cases and complex simulations on such environments, for the available hardware is also constrained.

Cloud Computing is becoming a popular alternative given its flexibility for on-demand resource provisioning. However, the common programming paradigms used for scientific computations, such as MPI, suffer from significant overhead because of the added virtualization and the lack of fast node interconnections.

In this project we proposed a paradigm shift from a multi-thread scheme to a data-centric model in order to overcome these limitations (*Objective 1*) mainly by minimising node interaction. By applying our MapReduce-based design the original simulator we were able to transform the program into a highly scalable MapReduce application that re-uses the same simulation library, while distributing the simulation load across as many nodes are desired (*Objective 2*).

We tested the resulting application on Hadoop MapReduce in both a physical and virtual cluster. The latter ran on top of Amazon EC2 (*Objective 3*) after being deployed using an AMI built for this specific application, with the purpose of gathering all the necessary software in a single deployment unit (*Objective 4*). The results we obtained were contrasted with the original application's performance (*Objective 5*); we found that we can even reduce the original application's simulation time with our adaptation. To sum up, we can state that we have met all the goals described in Sec. 1.2.

Additionally, we consider as very positive the successful usage of open source software, for it reduces the project's costs. Moreover, a side effect of our adaptation is that we need less resources to achieve better performance, which reduces equipment costs and also increases application's efficiency and sustainability. Finally, by breaking the dependence on local infrastructure, we can spread simulation scenarios of different sizes in a more flexible way, allowing the user to choose where to run the application –in the Cloud, in the local cluster or in standalone mode– to optimise cost and performance.

At a personal level, this project constituted an exceptional opportunity to participate in a real research initiative. Two papers resulted from it: *"Breaking data dependences in numerical simulations using Map-Reduce"*, accepted to participate in the SARTECO XXV Parallelism Congress, and *"A Cloudification Methodology for Numerical Simulations"*, currently under review.

8.2 Development Difficulties and Other Negative Aspects

The core of this project resides in the MapReduce application design. This task is critical since it affects most of the objectives of the project, therefore we had to conduct extensive research in order to ensure the model would be valid.

Besides the theoretical difficulty, this project had another deeply problematic characteristic: code reusage. The simulator was fully built in C++ when this adaptation was proposed, thus we were forced to use Hadoop Pipes API; integration between the API and the existing code was a difficult task since we aimed to minimise the modifications in the original code. Moreover, the simulator was found to present performance issues that had to be fixed before migrating to the transformed model; the same occurred with Pipes itself, so we had to include workarounds in the map and reduce code in order to avoid these issues.

Integration with the cluster environment was another obstacle, for the Tucán cluster had to be configured to support the specific software and versions required by Hadoop. Hadoop itself had to be recompiled in order to match the architecture of the original application, which implied further compatibility issues with the installed libraries of the host machines. These issues were also found during cloud initial deployment, since the instances provided by Amazon EC2 only contain a minimal OS installation.

Regarding job execution, platform configuration was the challenge after the simulator's functionality was verified. We found an heuristic to configure YARN and MapReduce frameworks by trial and error, after noticing that failures were mostly related to the lack of physical or virtual memory in the containers. We are looking into find a way to benchmark the original application to make this process automatic or, at least, reduce the number of tests needed to find a suitable configuration.

Regarding the application's behaviour, the current version might fail if the dedicated memory per container is sufficient for the platform –so no YARN errors are seen by the user– but it leads to thrashing in the node for specific input records. This issue is under study and will be tacked in future work.

Finally, we did not provide any multi-tenancy or security features in this system, as in this version it is expected to be run by a trusted user on a controlled environment, such as our private cluster or a virtual cluster with a security group properly configured to accept traffic from our client machine exclusively.

8.3 Future Work

The promising results we got with this application in terms of performance and scalability support that cloud migration is a viable solution to improve simulator's ability to execute larger experiments. As we previously stated, the MapReduce model introduced in 4.2 is one of the key aspects to consider for future improvement. We believe we can extend the model to a wider range of simulations, which can greatly benefit from the paradigm shift as the power consumption railway simulator did. Research lines regarding this aspect include:

- Multi-key mechanisms to deal with more complex simulations, which may hold several independent variables and input files types.
- Usage of optional MapReduce stages, such as *partition*, *combine* and *merge*, to manipulate simulations with a different structure or improve our model's performance.
- Direct job chaining as pipeline to avoid extensive usage of intermediate output files and disk spills, in order to improve performance.

Concerning the actual implementation of the model, the following improvements would also help to achieve better execution times and scalability:

- Platform and simulation library analysis to determine optimal configuration automatically.
- Support for on-the-fly YARN reconfiguration in a per-job basis.
- Performance optimization by means of a more suitable cloud instance mix.
- Automatic instance selection based on the platform's configuration parameters.
- Automatic cost analysis to select the most suitable environment –local cluster, cloud or standalone– for a specific use case.

Other features that could improve the system consist of:

- Multi-tenant execution of the Hadoop framework, i.e. allow different users to run jobs on the same Hadoop instance.
- Secure platform usage by means of native Hadoop security features such as authentication, data encryption and HDFS permission enforcement.

To finish, we would like to mention our current efforts to fix the known issues and deploy this system into an federated virtual cluster consisting of instances that belong to different cloud providers (currently, an OpenStack private cloud and Amazon EC2). With this approach, we can make the resulting simulation even more elastic and cost-efficient, since we could request nodes to a different provider only if our local cloud is not able to handle the simulation load.

BIBLIOGRAPHY

[1] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/o performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 40:1–40:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654100

[2] The ASCAC Subcommittee on Exascale Computing, "The opportunities and challenges of exascale computing," U.S. Department of Energy, Tech. Rep., 2010.

[3] K. Yelick, S. Coghlan, B. Draney, R. S. Canon *et al.*, "The magellan report on cloud computing for science," *US Department of Energy, Washington DC, USA, Tech. Rep*, 2011.

[4] P. Mell and T. Grance, "The nist definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[5] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, 2012.

[6] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable cloud applications—from theory to practice," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1417–1430, 2013.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] T. White, *Hadoop: The Definitive Guide: The Definitive Guide.* O'Reilly Media, 2009.

[9] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in *Proc. of the First Workshop on Hot Topics in Cloud Computing*, 2009, p. 118.

[10] A. Nash and D. Huerlimann, "Railroad simulation using opentrack," *Computers in railways IX*, pp. 45–54, 2004.

[11] A. Garcia, C. Gomez, F. Garcia-Carballeira, and J. Carretero, "Enhancing the structure of railway infrastructure simulators," in *International Conference on Engineering and Applied Sciences Optimization*, 2014.

[12] S. Iwnicki, *Handbook of railway vehicle dynamics.* CRC press, 2006.

[13] M. Nejlaoui, Z. Affi, A. Houidi, and L. Romdhane, "Analytical modeling of rail vehicle safety and comfort in short radius curved tracks," *Comptes Rendus Mecanique*, vol. 337, no. 5, pp. 303–311, 2009.

[14] F. Kiessling, R. Puschmann, A. Schmieder, and E. Schneider, *Contact Lines for Electric Railways. Planning, Design, Implementation, Maintenance*, 2009.

[15] R. Saa, A. Garcia, C. Gomez, J. Carretero, and F. Garcia-Carballeira, "An ontology-driven decision support system for high-performance and cost-optimized design of complex railway portal frames," *Expert Systems with Applications*, vol. 39, no. 10, pp. 8784–8792, 2012.

[16] C. H. Bae, "A simulation study of installation locations and capacity of regenerative absorption inverters in {DC} 1500v electric railways system," *Simulation Modelling Practice and Theory*, vol. 17, no. 5, pp. 829 – 838, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X09000161

[17] G. Poetsch, J. EVANS, R. Meisinger, W. Kortüm, W. Baldauf, A. Veitl, and J. Wallaschek, "Pantograph/catenary dynamics and control," *Vehicle System Dynamics*, vol. 28, no. 2-3, pp. 159–195, 1997.

[18] N. Cuartero, E. Arias, T. Rojo, F. Cuartero, and P. Tendero, "Calpe and indica: Two success stories," in *ASME 2012 International Mechanical Engineering Congress and Exposition.* American Society of Mechanical Engineers, 2012, pp. 271–281.

[19] V. Kindratenko and P. Trancoso, "Trends in high-performance computing," *Computing in Science Engineering*, vol. 13, no. 3, pp. 92–95, May 2011.

[20] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W.-m. Hwu, "Gpu clusters for high-performance computing," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on.* IEEE, 2009, pp. 1–8.

[21] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 47–. [Online]. Available: http://dx.doi.org/10.1109/SC.2004.26

[22] J. J. Dongarra, H. W. Meuer, E. Strohmaier *et al.*, "Top500 supercomputer sites," *Supercomputer*, vol. 13, pp. 89–111, 1997.

[23] J. Dongarra and P. Luszczek, "Linpack benchmark," *Encyclopedia of Parallel Computing*, pp. 1033–1036, 2011.

[24] Top500 Supercomputer Sites, "Top500 list," Nov. 2013. [Online]. Available: http://www.top500.org/list/2013/11/#.U6AGKJWn1QI

[25] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray User's Group (CUG)*, 2010.

[26] Top500 Organization, "Graph500 list," Nov. 2013. [Online]. Available: http://www.graph500.org/results_nov_2013

[27] W.-c. Feng and K. W. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.

[28] Green500 Organization, “Green500 list,” Nov. 2013. [Online]. Available: http://www.green500.org/lists/green201311

[29] B. Subramaniam, W. Saunders, T. Scogland, and W.-c. Feng, “Trends in energy-efficient computing: A perspective from the green500,” in *Green Computing Conference (IGCC), 2013 International.* IEEE, 2013, pp. 1–8.

[30] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina *et al.*, “The opportunities and challenges of exascale computing,” *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (November 2010)*, 2010.

[31] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, “Exascale computing study: Technology challenges in achieving exascale systems,” *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[32] S. Hemmert, “Green hpc: From nice to necessity,” *Computing in Science Engineering*, vol. 12, no. 6, pp. 8–10, Nov 2010.

[33] T. G. Peter Mell, “The nist definition of cloud computing,” National Institute of Standards and Technology, Tech. Rep., 2011.

[34] H. Hacigumus, B. Iyer, and S. Mehrotra, “Providing database as a service,” in *Data Engineering, 2002. Proceedings. 18th International Conference on*, 2002, pp. 29–38.

[35] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. R. Madden, H. Balakrishnan, and N. Zeldovich, “Relational cloud: A database-as-a-service for the cloud,” in *Conference on Innovative Data Systems Research, CIDR 2011, January 9-12, 2011 Asilomar, California*, 2011.

[36] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, “Naas: network-as-a-service in the cloud,” in *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228285

[37] D. H. Sharma, C. Dhote, and M. M. Potey, "Security-as-a-service from clouds: A comprehensive analysis," *International Journal of Computer Applications*, vol. 67, no. 3, pp. 15–18, 2013.

[38] I. Ari and N. Muhtaroglu, "Design and implementation of a cloud computing service for finite element analysis," *Advances in Engineering Software*, vol. 60–61, no. 0, pp. 122 – 135, 2013, cIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[39] B. Xiaoyong, "High performance computing for finite element in cloud," in *Future Computer Sciences and Application (ICFCSA), 2011 International Conference on*, 2011, pp. 51–53.

[40] M. AbdelBaky, M. Parashar, H. Kim, K. Jordan, V. Sachdeva, J. Sexton, H. Jamjoom, Z.-Y. Shae, G. Pencheva, R. Tavakoli, and M. Wheeler, "Enabling high-performance computing as a service," *Computer*, vol. 45, no. 10, pp. 72–80, 2012.

[41] "Amazon elastic mapreduce (amazon emr)." [Online]. Available: http://aws.amazon.com/es/elasticmapreduce/

[42] A. Raveendran, T. Bicer, and G. Agrawal, "A framework for elastic execution of existing mpi programs," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 2011, pp. 940–947.

[43] J. Earl, S. Conway, and J. Wu, "A new approach to hpc public clouds: The sgi cyclone hpc cloud," IDC, Tech. Rep., 2010. [Online]. Available: http://www.sgi.com/pdfs/4215.pdf

[44] L. Shi, H. Chen, J. Sun, and K. Li, "vcuda: Gpu-accelerated high-performance computing in virtual machines," *Computers, IEEE Transactions on*, vol. 61, no. 6, pp. 804–816, June 2012.

[45] Office of Advanced Scientific Computing Research, "The magellan report on cloud computing for science," U.S. Department of Energy, Tech. Rep., 2011.

[46] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling, "Scientific workflow applications on amazon ec2," in *E-Science Workshops, 2009 5th IEEE International Conference on*, Dec 2009, pp. 59–66.

[47] Z. Hill and M. Humphrey, "A quantitative analysis of high performance computing with amazon's ec2 infrastructure: The death of the local cluster?" in *Grid Computing, 2009 10th IEEE/ACM International Conference on*, Oct 2009, pp. 26–33.

[48] G. D'Angelo, "Parallel and distributed simulation from many cores to the public cloud," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, July 2011, pp. 14–23.

[49] S. N. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using mapreduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184 – 192, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11001075

[50] D. Yu, J. Wang, B. Hu, J. Liu, X. Zhang, K. He, and L.-J. Zhang, "A practical architecture of cloudification of legacy applications," in *Services (SERVICES), 2011 IEEE World Congress on*, July 2011, pp. 17–24.

[51] S. Srirama, V. Ivanistsev, P. Jakovits, and C. Willmore, "Direct migration of scientific computing experiments to the cloud," in *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, July 2013, pp. 27–34.

[52] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on.* IEEE, 2010, pp. 159–168.

[53] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, Dec 2008, pp. 277–284.

[54] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10, 2010, pp. 810–818. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851593

[55] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data.* ACM, 2007, pp. 1029–1040.

[56] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[57] H. Liu and D. Orban, "Cloud mapreduce: A mapreduce implementation on top of a cloud operating system," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.* IEEE Computer Society, 2011, pp. 464–474.

[58] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Nov 2010, pp. 565–572.

[59] K. Yamazaki, R. Kawashima, S. Saito, and H. Matsuo, "Implementation and evaluation of the jobtracker initiative task scheduling on hadoop," in *Computing and Networking (CANDAR), 2013 First International Symposium on*, Dec 2013, pp. 622–626.

[60] The Apache Software Foundation, "Apache hadoop 2.2.0," Oct. 2013. [Online]. Available: http://hadoop.apache.org/docs/stable/

[61] S. Caino, A. Garcia, F. Garcia-Carballeira, and J. Carretero, "Breaking data dependencias in numerical simulations using mapreduce," in *XXV Jornadas de Paralelismo*, 2014.