# Hadoop Scalability and Performance Testing in Heterogeneous Clusters

**Fernando G. Tinetti**[1], **Ignacio Real**[2], **Rodrigo Jaramillo**[2], **and Damián Barry**[2]
[1]III-LIDI, Facultad de Informática, UNLP,
Comisión de Inv. Científicas de la Prov. de Bs. As.
La Plata 1900, Argentina
[2]LINVI, Departamento de Informática, Facultad de Ingeniería, UNPSJB,
Puerto Madryn 9120, Argentina

**Abstract**— *This paper aims to evaluate cluster configurations using Hadoop in order to check parallelization performance and scalability in information retrieval. This evaluation will establish the necessary capabilities that should be taken into account specifically on a Distributed File System (HDFS: Hadoop Distributed File System), from the perspective of storage and indexing techniques, and queriy distribution, parallelization, scalability, and performance in heterogeneous environments. The software architecture will be designed and evaluated as either centralized or distributed, and the relevant experiments will be carried out establishing the performance improvement for each architecture.*

**Keywords:** Big Data, Information Retrieval, HDFS, MapReduce, Cluster, Parallelization, Scalability, Performance

## 1. Introduction

The amount of information is continuously growing: social networking, content management systems (CMS) and portals in general and as collaboration platforms in particular, data within organizations generated either by production systems or by digitizing existing information. Data usually measured in gigabytes a few years ago is now measured un terabytes and petabytes [5] [6]. Data as well as relevant applications typically require more resources than those available on a single computer. The challenge is therefore to produce and handle computing infrastructure that allows to take advantage (harnessing) of existing computing platforms, usually heterogeneous. Thus, several computers wroking collaboratively, would reach availability and scalability to cope with the currently needed information processing [7] [8]. Reusing low-cost equipment allows to address the aforementioned problem, and requires techniques of distributed systems, where each computing system has local storage and computation facilities so that processing and access can be distributed and balanced in a heterogeneous cluster [9]. A set of desirable properties for an information sharing and data reocovering system in a heterogeneous and scalable environment a could be defined [7] [10] [11]: high performance, fault tolerance and heterogeneous computing. Moreover, the NoSQL solutions for managing large volumes of data are typically based on the usage of a heterogeneous system.

There are several techniques for configuring heterogeneous computing environments. We have concentrated our work in the framework programmed in Java called Hadoop to store and process large amounts of data in clusters [1] [2]. HDFS besides being a distributed file system, scalable and portable, solves availability and reliability issues by replicating data in multiple computers [9].

### 1.1 Hypothesis

The amount of data that humans are capable of generating and storing hinders the analysis and information processing in general. Processing/analysis in this field is commonly referred to as *Big Data* applications [3]. Several problems are involved, two of the most complex ones could be *reduced* to the following questions:

- 1. How to store and protect the large volume of available data?
- 2. How to process and evaluate data in an acceptable period of time?

Specifically with regard to the latter question, the hypothesis from wich we work is the existence of a performance and scalability characterization of each heterogeneous cluster. This characterization is given in the context of the different techniques for handling large volumes of data while varying the number of nodes that comprise it.

### 1.2 Contribution

At least, we check a real usage of the infrastructure, Hadoop, proposing a design that facilitates scalability. This design could be especially used by organizations and agencies that need to handle large volumes of information, as in national or local governments or private sector companies. Another significant contribution lies in the utility that provide this type of architecture in the field of research and development.

### 1.3 Goals

We have defined a set of objectives guiding the work reported in this paper:

1) Design different scalable architectures for a Hadoop cluster varying the number of nodes in order to analyze processing time.
2) Select bibliographic material and generate a knowledge based on the techniques and methods used in partitioning, replication, and distribution in the Apache Hadoop infrastructure.
3) Set parameters and evaluate different architectures for optimizing Hadoop configuration.

# 2. Hadoop

Hadoop is a framework that allows to build a cluster architecture, providing parallel recovery of information and replication. Also, Hadoop implement a simple way to add and/or drop cluster nodes, improving scalability, minimizing the likelihood of failure nodes containing distributed data. Developed in the Java programming language, by the community of free software Apache, the Hadoop architecture is composed of three main components:

- The Hadoop Distributed File System*HDFS*, using a master/slave architecture, as shown in Fig. 1.
- The MapReduce framework, which allows the programmer to split and parallelize complex calculations in any number of computers.
- The Hadoop Common, a set of tools for integrating Hadoop subprojects.

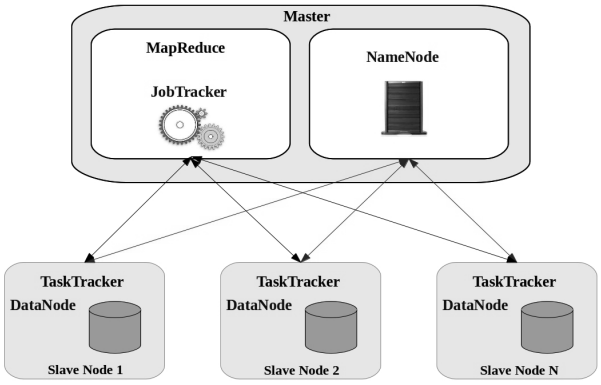


Fig. 1: Hadoop Architecture

The two main components, the HDFS and MapReduce, define a stable, robust, and flexible framework for distributed applications, making it possible to work with multiple nodes and process large amounts of information.

The HDFS is designed to provide high performance and data reliability on heterogeneous hardware. MapReduce allows the development of parallel processing applications, focused on scalability. Queries on distributed data could be distributed as well, thus enhancing performance via parallel/distributed processing. Both (HDFS-MapReduce) are designed to analyze large amounts of structured and unstructured data.

## 2.1 Hadoop DFS

HDFS implements a master/slave architecture as shown in Fig. 2, where: a) NameNode is the master process, b) DataNodes are the slave processes, and c) the master process is replicated in a Secondary NameNode. The HDFS keeps separately metadata (in the NameNode) and data (in the DataNodes). System (data) reliability is achieved by replicating files in multiple DataNodes, which also allows faster transfer rates and access. All files stored in HDFS system are divided into blocks, whose size usually is between 64 MB and 128 MB. A Hadoop cluster can consist of thou-

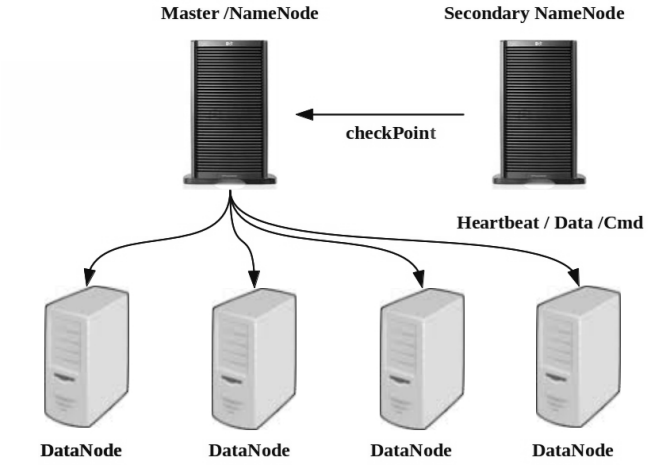


Fig. 2: HDFS Architecture

sands DataNodes wich respond to different read and write requests from clients, also maintaining block replication. The DataNodes regularly send information to the NameNode of its blocks to validate consistency with other DataNodes. A cluster may consist of thousands of DataNodes, each of which stores a portion of (possibly replicated) data. Each of the DataNodes is likely to fail. The HDFS provides rapid and automatic failover recovery via replication and the metadata contained in the NameNode.

During normal operation the DataNode sends signals (heartbeats) to NameNode every three seconds by default. If the NameNode does not receive a defined number of the heartbeats from a DataNode, the DataNode is considered out of service. Then the NameNode triggers the creation of new replicas of the DataNode out of service in other (not failing) DataNodes. Heartbeats also contain information about the total storage capacity, the fraction of storage that is in use, and the number of files or data transfer in progress.

### 2.2 Hadoop MapReduce

MapReduce allows Hadoop the parallel processing on large volumes of data through multiple nodes in a cluster, the data to be processed may be stored in the HDFS. The execution of a MapReduce process usually divides the input data into a set of independent chunks of information that are processed by the Map tasks in parallel. Then, the results of the map tasks are classified and will be the input to Reduce tasks. Typically both the input data and output data are stored in the file system.

MapReduce is based on the Master/Slave architecture, similar to that of the HDFS, as shown in Fig. 3. The Master runs the so called JobTracker and slaves run the TaskTrackers. The JobTracker is responsible for the management and control of all sumitted jobs. Also, it is responsible for task distribution and management of available TaskTrackers, trying to keep the job as close to the data as possible. The JobTracker takes into account the machines (nodes) that are close to and/or contain the data needed.

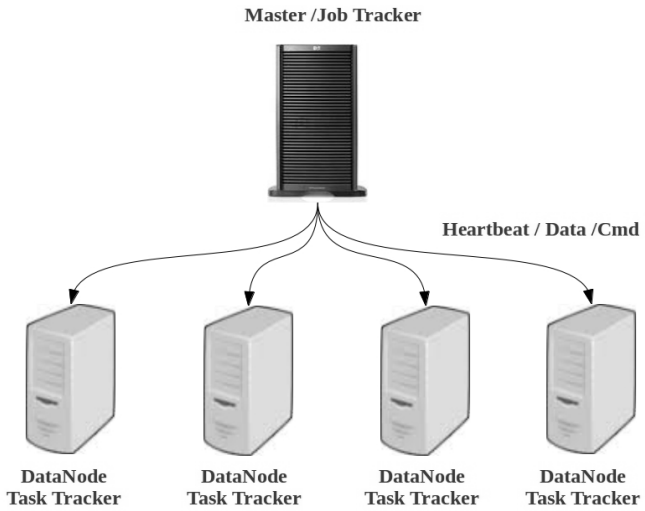


Fig. 3: MapReduce Architecture

MapReduce is based on key/value pairs, which are processed in (are the input to) Map tasks. Every Map task returns a list of pairs in a different domain data. All the pairs (generated by the Map tasks) with the same key are grouped and processed by a Reduce task.

MapReduce handles fault tolerance in a similar way to that described for the HDFS: each TaskTracker process reports regularly its status to the JobTracker process. If over a period of time the JobTracker process has not received any report from a TaskTracker process, the TaskTracker process is considered as not running. In case of failure, the task is reassigned to a different TaskTraker process.

## 3. Design and Implementation of Experiments

Different cluster configurations were evaluated from the point of view of scalability and (*raw*) performance. We also used two benchmarks, each used for measuring different performance metrics. Hardware and benchmarks are described in the following subsections.

### 3.1 Computers-Hardware

We specifically focused our work on heterogeneous computing cluster configurations, using the computers detailed below:

1) Name: **Master**
   Processor: Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
   Memory: 10 GB
   SATA Disk: 500 GB

2) Name: **Slave1**
   Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
   Memory: 16 GB
   SATA Disk: 500 GB

3) Name: **Slave2**
   Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
   Memory: 8 GB
   SDisk: 1 TB

4) Name: **Slave3**
   Processor: Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz
   Memory: 8 GB
   SATA Disk: 1TB

The computers were used for different testing scenarios: a centralized one and three cluster-like installations. Initially, the Master was used as a standalone centralized Hadoop installation, including a Master and a DataNode. We will use this installation as the departure point for testing the Hadoop software as well as measuring a non-distributed environment. The different cluster installations (from *1* to 4 computers) were made up just taking advantage of the previous installation by adding one more computer including one more DataNode. The same number of Map and Reduce tasks per node is maintained in all the experiments.

The Hadoop client process (which is not part of the main Hadoop infrastucture shown in Fig. 1 before) in every experiment was run on

- Name: **Client**
  Processor:AMD Turion(tm) X2 Dual-Core Mobile
  Memory: 4 GB

SATA Disk: 320 GB

### 3.2 Benchmarks-Software

We used two well-known tests provided by the Hadoop software: **TestDFSIO** and **TeraSort**. TestDFSIO is aimed at assessing the performance of the cluster/Hadoop installation and TeraSort is focused on scalability and parallelization.

The Hadoop **TestDFSIO** benchmark is used for reading and writing files in the HDFS, indicating number and size of files. TestDFSIO provides timing information, performance, and the average I/O speed. Basically, TerstDFSIO is useful for:

- Measurement tasks such as stress tests on HDFS.
- Discovering bottlenecks in the network.
- Evaluating the hardware performance.
- Checking the operating system configuration and Hadoop cluster machines in general.

In short, TestDFSIO gives a first impression of how fast the cluster works in terms of I/O. This test runs MapReduce jobs it is a MapReduce program that reads/writes random data from large files. Each Map task performs the same operation in a separate file and informs speed to a Reduce task, which is programmed to collect and summarize all measurements, given in MB/seg.

The Hadoop **Terasort** benchmark is designed to assess the performance and scalability of a Hadoop installation. It is specifically designed to check the distrubution of processes in the cluster using Map Reduce. TeraSort execution actually involves the execution of three MapReduce programs:

TeraGen for data generation

TeraSort for sorting the generated data

TeraValidate for sorted data validation

The TeraGen program writes data to disk just like testDFSIO-write creates random data. TeraSort sorting performance is based on the way that divides data between mappers/reducers and how data is collected and written by the partitioner. A partitioner is implemented for achieving a balanced workload. The partitioner uses an ordered list of N-1 sample keys that define the range of keys for each Reduce. In particular, a key is sent to the i-th Reduce if it resides within a range such that sample[i-1] $<=$ key $<$ sample[i], this ensures that the ith Reduce output is less than the output of the (i+1)-th Reduce. The TeraValidate program ensures that the output is globally sorted out by controlling (in the output data) that each key is less than or equal to the previous one.

## 4. Results

A TestDFSIO preliminary test was carried out for assesing Hadoop I/O performance of different cluster configurations. This test was run increasing from 1 to 14 the number of files of size 1 GB each (i.e. from 1 to 14 GB). Cluster architecture was also increased from 1 to 4 nodes, as shown in Fig. 4. Read operation results were taken into account for this run, with default settings, which imply

- BufferSize: 1000000 bytes
- Replication factor: 3
- Number of tasks in parallel by node: 2
- Block size: 64 MB

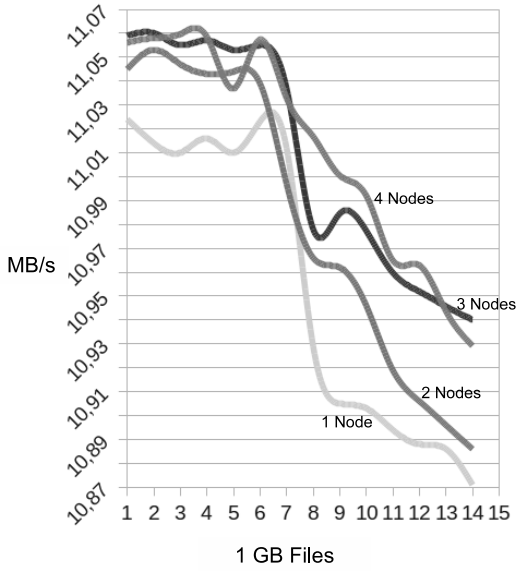


Fig. 4: TestDFSIO Read Performance

Fig. 4 shows how the performance decreases as the number of files (and involved data) increases, due to several factors among wich we can mention network traffic, local disk accesses, etc. Results are labelled as 1...4 Nodes from the standalone case to the complete cluster, with 4 nodes (computers) running the experiment. It is worth mentioning that heterogeneity does not play an important role and is almost negligible. However, when using the 4 computers (4 Nodes), performance is slightly penalized as compared to the case in which only 3 computers (3 Nodes) are used for the 14 files.

TeraSort results are the ones we are really interested in, because parallel computing is directly involved. Data to be sorted has to be generated, and we chose to follow the Fibonacci sequence $\times 10^7$. Therefore:

- In the first run $1 \times 10^7$ records or rows are generated, where each row is 100 bytes in size.
- $2 \times 10^7$, $3 \times 10^7$, $5 \times 10^7$, $8 \times 10^7$ and $13 \times 10^7$ records are then generated.

i.e. from 10 to 130 millions of records to be sorted. And given that each record is 100 bytes long, the total amount of data is among 1 GB to 13 GB. For each cluster configuration (from a standalone Master to the complete 4 nodes cluster), TeraGen was first executed to generate the data serie. Once

the data is generated, TeraSort is run in set of 10 identical experiments and the average runtime is taken as the result, just to avoid transient experiment "noise". Finally TeraValidate was run to confirm that the data were actually sorted. For each test configuration TeraGen was first executed to generate the first data series, and TeraSort was run in a sequence of 10 repetitions, thereby strengthen the statistical results and obtain representative values. TeraValidate was always used to confirm that the data were actually sorted. Table 1 shows measured runtime for each experiment, i.e. varying the number of computers mand the amount of data

Table 1: Summary of TeraSort Results

| | 1 Node | 2 Nodes | 3 Nodes | 4 Nodes |
|---|---|---|---|---|
| 1 GB | 71 | 80 | 68 | 81 |
| 2 GB | 114 | 148 | 113 | 102 |
| 3 GB | 247 | 210 | 169 | 163 |
| 5 GB | 577 | 373 | 258 | 210 |
| 8 GB | 1042 | 885 | 504 | 316 |
| 13 GB | 2386 | 1888 | 1098 | 574 |

to be sorted, where "1 Node" represents the standalone configuration (only the Master node is running), "2 Nodes" represents the configuration with the master and Slave1 running, and son on. There are several interesting details which can be quantified with the values shown in Table 1:

- A larger amount of data to be sorted implies increasing the runtime, as could be expected *a priori*.
- For small size data sets (e.g. 1 GB or 2 GB) using more computers does not imply a performance improvement. More specifically, the runtime using the Master and Slave1 increases the runtime for 1 GB and 2 GB data to be sorted.
- For large size data sets (e.g. 8 GB or 13 GB) using more computers always imply a performance improvement. The improvement depends on several factors such as centralized to distributed (1 Node and 2 Nodes), or where the added computer is relatively less powerfull than those already running in the cluster (3 Nodes and 4 Nodes, the 4th node is the least powerfull one in the cluster).
- For intermediate size data sets of those experimented with (e.g. 3 GB and 5 GB) gains are difficult to evaluate, and some more specific experiments should be carried out even with other benchmark/s.
- Given that Hadoop is expected to handle TB of information, all of the results could be considered highly encouraging, since even handling small size data sets it is possible to obtain performance gains using heterogenous computers.
- Some results regarding performance enhancements are linearly related, while others not. Relative computing power of each node has not been calculated, so the values cannot be strictically analyzed/evaluated from a numeric point of view. We have to continue our experiments (at least) in that line of work.

Fig. 5 shows the values of Table 1 graphically, focused on experiments runtime (on the vertical axis) depending on the amount of data to be sorted (on the horizontal axis). Some scalability details can be identified in Fig. 5, since

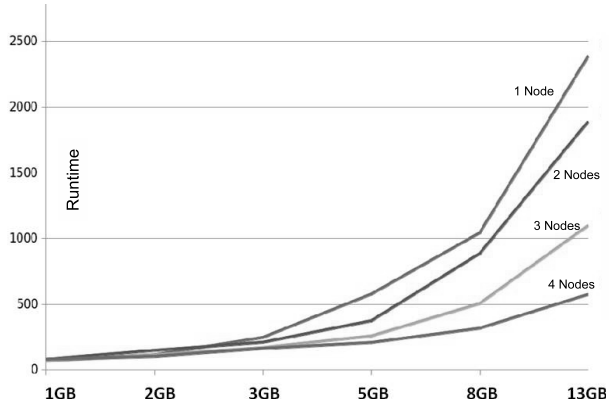


Fig. 5: TeraSort Scalability

scaling (increasing) data clearly implies increased runtime. At this time, it should be recalled that sorting is an $O(n^2)$ problem in general. Also, Fig. 5 clearly shows that increasing the number of nodes in a cluster the runtime is reduced using Hadoop with the proper configuration of HDFS and MapReduce.

# 5. Conclusions

Even when we have several problems in the Hadoop installation and configuration stages (mainly due to lack of documentation at the time we began this research some years ago) we have set a successful environment for experimentation with Hadoop in heterogeneous clusters. The installation and configuration could be replicated in every cluster (either homogeneous or heterogenous).

We have used TestDFSIO as a departure point: Hadoop I/O performance. Furthermore, we have found that TestDFSIO does not provide any information about parallel computing and/or scalability. At most, TestDFSIO could be used for node failure experimentation, varying the replication factor and injecting node failures in different scenarios.

We have used TeraSort for performance analysis in general, and performance scalability in particular. At this point, the MapReduce programming model and its conceptual basis are the most relevant. We have analyzed the processes involved in a Hadoop job so that we were able to determine the correct amount of Map and Reduce tasks per node and to properly configure MapReduce parameters. Our experiments show that processing times decreases as the cluster nodes are added. Clearly, MapReduce does not solve everything, is a solution for those problems that fit the model and can be parallelized. One of the important results of experimentation

is that having small size data sets to process (maybe up to 5 GB specifically for sorting) would suggest to avoid MapReduce at all (see Fig. 5). More specifically: adding computers to a cluster would not add real/proportional performance gains.

From using both, TestDFSIO and TeraSort it is possible to conclude that almost inexpensive infrastructure (basically: low-cost computers) enables the processing of large volumes of data. And the obtained performance in general terms, besides being linked to the implemented hardware, also depends on the correct configuration.

We have several lines of further work, taking advantage of the Hadoop experience we have acquired:

- Relative computing power evaluation, as aforementioned. We should design specific experiment scenarios and analysis.
- We should try gathering more computers, scalability in the tens, hundreds and thousands of nodes would give a better idea. Our results are exiting, but we know we have a limited number of available computers, and it is caused by our limited budget.
- We should try others benchmarks, by adapting current ones or developing new ones. An initial idea would be to identify different application areas, and use one or more benchmark per application area.

# References

[1] S. Guo, Hadoop Operations and Cluster Management Cookbook, Packt Publishing, 2013.
[2] A. Holmes, Hadoop in Practice, 2nd. Ed., Manning Publications 2014.
[3] P. C. Zikopoulos, C. Eaton, D. deRoos, T. Deutsch, G. Laspis Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Osborne Media, 2012.
[4] F. G. Tinetti, D. Barry, I. Aita, F. Páez, Distributed Search on Large NoSQL Databases, PDPTA2011, 2011.
[5] N. Scola, WhiteHouse.gov Goes Drupal [Updated], [Online]. Available: http://techpresident.com/blog-entry/whitehousegov-goes-drupal
[6] C. Henderson, Building scalable web sites, O'Reilly Media, Inc., 2006.
[7] O. M. Tamer, P. Valduriez, Principles of distributed database systems, Springer Science & Business Media, 2011.
[8] A. K. Elmagarmid, M. Rusinkiewicz, A. Sheth, Management of heterogeneous and autonomous database systems, Morgan Kaufmann, 1998.
[9] J. Venner, S. Wadkar, M. Siddalingaiah, Pro Apache Hadoop, Apress, 2nd ed., 2014.
[10] D. Taniar, C. H. C. Leung, W. Rahayu, S. Goel, High performance parallel database processing and grid databases, John Wiley & Sons, 2008.
[11] R. Ho, Scalable System Design Patterns, Pragmatic Programming Techniques, [Online]. Available: http://horicky.blogspot.com/2010/10/scalable-system-design-patterns.html 2010.