Optimizing Hadoop Parameters Based on the Application Resource Consumption

Ziad Benslimane
Abstract

Optimizing Hadoop Parameters Based on the Application Resource Consumption

Ziad Benslimane

The interest in analyzing the growing amounts of data has encouraged the deployment of large scale parallel computing frameworks such as Hadoop. In other words, data analytic is the main reason behind the success of distributed systems; this is due to the fact that data might not fit on a single disk, and that processing can be very time consuming which makes parallel input analysis very useful. Hadoop relies on the MapReduce programming paradigm to distribute work among the machines; so a good balance of load will eventually influence the execution time of those kinds of applications.

This paper introduces a technique to optimize some configuration parameters using the application’s CPU utilization in order to tune Hadoop; the theories stated and proved in this paper rely on the fact that the CPUs should neither be over utilized nor under utilized; in other words, the conclusion will be a sort of an equation of the parameter to be optimized in terms of the cluster infrastructure.

The future research concerning this topic is planned to focus on tuning other Hadoop parameters and to use more accurate tools to analyze the cluster performance; moreover, it is also interesting to research any possible ways to optimize Hadoop parameters based on other consumption criteria such the input/output statistics and the network traffic.
## Table of contents

1 Introduction ........................................................................................................ 1  
   1.1 Big Data ........................................................................................................ 1  
   1.2 Data Analytic ................................................................................................. 2  
   1.3 Distributed Computing ................................................................................... 2  
   1.4 Cloud Computing ......................................................................................... 4  

2 Hadoop ............................................................................................................... 6  
   2.1 The Hadoop Architecture .............................................................................. 6  
    2.1.1 HDFS ..................................................................................................... 7  
    2.1.2 MapReduce .............................................................................................. 9  
   2.2 Advantages of Hadoop .................................................................................. 11  
   2.3 Hadoop Configuration .................................................................................. 12  

3 Optimization Ways .......................................................................................... 13  

4 Problem and Proposed Solution ...................................................................... 16  
   4.1 Statement of Problem .................................................................................... 16  
   4.2 Proposed Solution ......................................................................................... 16  
   4.3 Theoretical Assumptions ............................................................................. 17  
   4.4 Experiment Plan ........................................................................................... 18  

5 Experiment Design: Single Node Cluster ..................................................... 19  
   5.1 Test Cases .................................................................................................... 19  
    5.1.1 Grep ....................................................................................................... 19  
    5.1.2 PiEstimator ............................................................................................. 19  
    5.1.3 MRIF ...................................................................................................... 20  
   5.2 Objective ..................................................................................................... 20  
   5.3 Setup ............................................................................................................ 21  
   5.4 PiEstimator .................................................................................................. 22  
   5.5 Grep ............................................................................................................. 22  
   5.6 MRIF ............................................................................................................ 22  
   5.7 Results ......................................................................................................... 22  

6 Experiment Design: Multi-Node Cluster ....................................................... 27  
   6.1 Objective ..................................................................................................... 27  
   6.2 Setup ............................................................................................................ 27  
   6.3 PiEstimator .................................................................................................. 29  
   6.4 Grep ............................................................................................................. 29  
   6.5 MRIF ............................................................................................................ 29  
   6.6 Results ......................................................................................................... 30  
    6.6.1 PiEstimator ............................................................................................. 30  
    6.6.2 Grep ....................................................................................................... 34  
    7.6.3 MRIF ..................................................................................................... 38
7 Conclusion and Future Work ................................................................. 43
  7.1 Conclusion ..................................................................................... 43
  7.2 Future work .................................................................................... 43
Acknowledgments .................................................................................... 45
References .............................................................................................. 46
Appendix A The MapReduce Process ...................................................... 48
Appendix B The Quadratic Sieve Algorithm ........................................... 50
Resume and Research Papers Published ................................................ 52
1 Introduction

1.1 Big Data

Electronically stored data has been increasing tremendously in the last decade; this was caused by the decreasing price of hard disks as well as the convenience in saving and retrieving the data. Moreover, as the business intelligence field is concerned, there is a strong belief that storing as much data as possible will eventually help in generating more information that can be used to draw more accurate conclusions.

In most cases, companies use database management systems in order to manage the saved data; however, if that data is really huge, a typical DBMS will have trouble to process queries in a decent amount of time. For example, Wal-Mart, a famous retailer feeds their central database with more than 2.5 petabytes per hour (Cukier, 2010); this kind of example can explain the results found by the “Enterprise Strategy Group” shown in the Figure 1.1 (The unified Guide, 2011).

![Figure 1.1 Total Archived Capacity](image)
1.2 Data Analytic

Data Analytic, or simply referred to as DA, is the process of analyzing raw data in order to make useful conclusions aiming for better decisions (Search Data Management, 2008).

One of the management consultancies argues that organizations might have a lot of data generated in the past few years but taking advantage of all that is not an easy task (Data Analytics, 2011). The data might represent important information about customers and clients, that once analyzed, can offer better decisions about their future business. For example, a school library that has a limited budget and needs to buy more books needs to know some information such as: which books are the most borrowed, how many study majors make use of that book, and so on; the lesson here is that the more information you analyze the better the decision you can make.

So in order to make Data Analytic for Big Data possible, a new computing technology had to emerge in order to allow a higher reading speeding of Big Data as well as a higher capacity in random memory. This computing paradigm is known as Distributed Computing, and it will be explained in detail in the next section.

1.3 Distributed Computing

Distributed Computing refers to the execution of an application on multiple systems; this is usually done for heavy programs that need to be divided into multiple tasks and then distributed among the machines. The distributed computers communicate with each other using a network connection.

Distributed Computing allows the execution of the application to be carried in parallel assuming the tasks are independent of each other; that is no task is waiting for results from another one. This parallelism has solved the problem of Big Data analytics by enabling multiple parallel reads as well as multiple concurrent processing.

For example, The New York Times had four terabytes of scanned articles that needed to be converted to the PDF format; so the managers made use of this concept by renting multiple machines working together that ended up finishing the job in less than 24 hours (Tom, 2010). This task would have taken many days if executed on a single machine as data would have been read sequentially.

The advantages of this computing paradigm include but are not limited to the
following:

- Scalability
- Speed
- No single point of failure

Scalability basically means that multiple computers can be added to the cluster without much overhead; if you have already tens of computers working together then adding another ten should be added without much overhead.

Moreover, the speed is an advantage because many reads and computations can occur in parallel thus at a higher rate than any single machine.

The last advantage, no single point of failure, refers to the fact that even if one machine crashes, the others can still manage to continue the main job as all the data should have been replicated across multiple nodes and any remaining computation once assigned to that machine should be resumed or restarted by another one.

On the other hand, Distributed Computing has also some disadvantages such as:

- Less secure
- Debugging can be cumbersome

It is believed that a distributed set of computers is less secure than a single machine, because many extra security holes would have to be managed like the network communication, authentication at every single machine, and encryption of every replicated data.

The debugging issue is due to the likelihood of the problem occurring at any machine, so managers usually have to remote login to the systems in order to check the logs of each one of them.

The usefulness of this computing paradigm needs some kind of architecture that controls the way developers should write programs, and how does each machine exactly interact with the others. For example, the following basic questions have to be answered:

- How is the work divided?
- How is replication offered?
- How does the communication occur?

Of course, many more questions have to be addressed by the architecture being used.

There are several architectures currently used for Distributed Computing, and the most influential ones are the following: client–server, n-tier architecture, distributed
Chapter 1 Introduction

objects, loose coupling, or tight coupling (Wikipedia, 2012).

In our case, and as far as our experiments are concerned, the most important one that needs to be studied in detail is the “Tightly Coupled” architecture, which represents a cluster of machines working together by sharing the same process and by working on parts separately before merging the results back (Wikipedia, 2012). Figure 1.2 shows the basic design of a tightly-coupled architecture.

![Figure 1.2 Tightly Coupled Architecture](image)

1.4 Cloud Computing

The Distributed Computing defined previously needs a lot of resources; a company that needs to do a very large job on a set of big input might need hundreds of computers to complete such a task in a decent amount of time. The problem here is whether the company really has to buy all that equipment knowing that they only need them for few days or so; the answer here is negative with the new emerging concept of Cloud Computing.

Cloud computing is defined by the National Institute of Science and Technology to be a service offering computing capabilities as well as other resources as a utility to the client (National Institute of Standards and Technology, 2011). A good analogy here would be the way customers consume electricity; people usually just plug in devices on their walls’ outlets and get as much electricity as needed for a certain price. Figure 1.3 depicts the main evolution that led to the propagation of cloud computing as one of the most advanced computing services (Bias, 2010).
Figure 1.3 How Cloud Computing Evolved
2 Hadoop

Hadoop emerged from an old Apache project called “Nutch” concerning the creation of an open source web search engine; this project was facing great challenges such as the need for expensive hardware and high monthly expenses (Tom, 2010). A working prototype was released by 2002 but scalability was the main concern as the developers stated that the architecture would not cope with the billions of pages on the web (Tom, 2010). Later, in 2003, a paper came out describing the Google File System (GFS); this has given the Nutch group a hint that they might get use of that in order to store pages and thus to solve their scalability issue. After that, a new project was started in order to make their own open source implementation of GFS (Tom, 2010); in a similar fashion, Google released another paper describing the MapReduce programming paradigm, and the Nutch group ported all their algorithms to run in that standard. Finally, a new project emerged called Hadoop with the help and support of Yahoo (Tom, 2010).

In order to understand Hadoop, its architecture has to be analyzed in detail.

2.1 The Hadoop Architecture

Hadoop, which is an open-source framework allowing distributed computing for large data sets, is made of two components:

- The distributed file system component referred to as HDFS
- The MapReduce component

The outline of how the two components as well as the Nutch web search fit on the overall architecture is shown in Figure 2.1 (McNamara, 2011).
2.1.1 HDFS

The Hadoop Distributed File System was based on a paper published by Google about their Google File System. It is a distributed file system that runs on top of the existing file system on every machine in the cluster which makes it highly portable. HDFS runs on top of commodity hardware which makes it very experienced and robust against any kinds of failures; in other words, one of the main goals of this Hadoop component was fault tolerance (Dhruba, 2012). Another main important aspect of HDFS is the built-in support of huge data files; the project documentation claims that a normal input file is assumed to be in terms of gigabytes up to few terabytes (Dhruba, 2012).

HDFS handles data coherency by assuming that data is only written once and read multiple times; so there is no need to implement any coherence model such as the ones implemented in processor cache systems.

The HDFS is actually a master/slave architecture where the master is called the NameNode and the slave is referred to as the DataNode. The HDFS nodes can be characterized by the following (Dhruba, 2012):

- One NameNode
  - Responsible for the file system namespace
  - Holds the metadata concerning the files stored in DataNodes
  - Should have the most expensive hardware to avoid single point of failure
  - Executes operations on the file system
Chapter 2 Hadoop

- DataNodes
  - Where the blocks are stored
  - Commodity hardware since replication is offered by the software layer
  - Serves read and write requests to the clients
  - Handles block creation and deletion
  - Performs replication
  - Get the instructions from the NameNode

HDFS offers data replication by the software layer in order to distribute the input files across the DataNodes to enable the application to work on the data locally and also to avoid any loss of data once a DataNode crashes. HDFS does this by using a block system of fixed size where each file is broken to a multiple of blocks; moreover, it allows the client to change the size of the block which is 64 MB by default. The replication factor can also be specified by modifying the configuration files.

The way replication is done is one of the big advantages of HDFS; it is done in a way to assure data reliability, availability, and network utilization (Dhruba, 2012)

HDFS Communication protocols are all carried on top of TCP/IP; for example the client requests a TCP connection to the NameNode using the ClientProtocol while the
DataNodes reach it through the DataNode Protocol (Dhruba, 2012).

The HDFS architecture and the interaction between its nodes as well as the client are demonstrated in figure 2.2 (Dhruba, 2012).

2.1.2 MapReduce

MapReduce is basically a framework made to process large amounts of data in parallel by breaking the input into multiple independents splits (Wikipedia, 2012); this kind of division is required for processing tasks concurrently without having to execute things sequentially.

A program written under the MapReduce framework should operate in two phases called “map” and “reduce”.

As far as Hadoop is concerned, the MapReduce job splits the inputs and then makes use of the “map” phase to process splits in parallel; so each map task works on its share of the input without worrying about the rest of the map tasks. After that, the framework itself uses a sorter to rank the map outputs before passing them to the reduce tasks that merges them (Apache, 2012). The following is a pseudo-code for the map and reduce functions (Ho, 2008):

```java
map(input_record)
{
    ...
    emit(k1, v1)
    ....
    emit(k2, v2)
}

reduce (key, values)
{
    aggregate = initialize()
    while (values.has_next)
    {
        aggregate = merge(values.next)
    }
    collect(key, aggregate)
}
```
The number of map tasks spawned by Hadoop cannot be controlled; however, the number of reduce tasks can easily be fed to Hadoop either through the command line or by changing the configuration files.

The MapReduce framework is considered to be a master/slave architecture where the master is the JobTracker while the slave is the TaskTracker. There exists only one JobTracker per cluster while there are many TaskTrackers to allow parallelism; here are the main responsibilities of both of the MapReduce nodes (Hadoop-Wiki, 2010) (Hadoop-Wiki, 2009):

- **JobTracker**
  - Receives jobs from the clients
  - Requests location of data from the NameNode
  - Finds the nearby TaskTrackers
  - Monitors the TaskTrackers
  - Decides to which TaskTracker it should assign a certain job
  - Manages failed jobs after receiving a signal from the responsible TaskTracker
  - Updated the status of each task
  - Keeps the clients updated with the latest tasks’ information
  - Suffers from the Single point of failure issue

- **TaskTracker**
  - Accepts MapReduce tasks from the JobTracker
  - Broadcasts the number of free task slots it can handle
  - Spawns a Java Virtual Machine process for each task
  - Notifies the JobTracker about the status of each task
  - Sends continuous “alive” signals to the JobTracker

The interaction between the MapReduce nodes with each other and with the client is illustrated in Figure 2.3 (Ho, 2008).
2.2 Advantages of Hadoop

Hadoop has many advantages that make it useful for many large companies in industry today; some of those are shown below:

- Performs large scale computations
- Handles an input that is much larger than any single hard drive
- Handles failure and data congestion
- Uses a simplified programming model
- Performs automatic distribution of data
- Has a very good scalability
Keeps up with Moore’s law:

- Even if the number of transistors doubles every two years or so, reading data from hard disks to CPUs is relatively too slow. Hadoop uses multiple machines to read the data simultaneously.

### 2.3 Hadoop Configuration

Hadoop can be configured and customized according to the user’s needs; that is, the configuration files that get generated automatically after installation can be modified to suit the specific needs of applications and clusters.

The configuration files fall in two categories: read-only default configuration and site-specific configuration. The first one includes “core-default.xml”, “hdfs-default.xml”, and “mapred-default.xml”; while the second category includes “core-site.xml”, “hdfs-site.xml”, and “mapred-site.xml”.

In our case, the second category was modified accordingly in order to successfully conduct the test cases; each file includes some parameters to be changed, the Table 2.1 shows the most important fields that were interacted with while carrying the experiments:

<table>
<thead>
<tr>
<th>File</th>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapred-site.xml</td>
<td>mapred.tasktracker.map.tasks.maximum</td>
<td>The maximum number of parallel map tasks on a TaskTracker</td>
</tr>
<tr>
<td></td>
<td>mapred.tasktracker.reduce.tasks.maximum</td>
<td>The maximum number of parallel reduce tasks on a TaskTracker</td>
</tr>
<tr>
<td>Core-site.xml</td>
<td>fs.default.name</td>
<td>URI of NameNode.</td>
</tr>
<tr>
<td>conf/hdfs-site.xml</td>
<td>dfs.name.dir</td>
<td>The path where the namespace information gets stored by the NameNode</td>
</tr>
<tr>
<td></td>
<td>dfs.data.dir</td>
<td>A list of paths where the DataNode stores its blocks</td>
</tr>
</tbody>
</table>
Chapter 3 Optimization Ways

The current Hadoop optimization research is mainly conducted by Apache which is offering three different subprojects related to Hadoop: Common, HDFS, and MapReduce.

Another good source of Hadoop optimization techniques is Cloudera, which gives training based on online classes, tutorials, as well as utilities to better understand the implementation details of Hadoop (Cloudera, 2011).

Many papers are now tackling the problem of performance from different perspectives, but most papers argue that Hadoop configuration has to be changed according to the cluster’s hardware and application’s needs; the configuration includes over 180 parameters and most of them should not be kept at their default values.

In the paper titled: “Optimizing Hadoop for the Cluster” (Hansen, (n.d.)), it is argued that Hadoop’s default configuration should not be expected to be the most optimized one for all kinds of clusters and all types of applications; according to the paper, this motivation should influence Hadoop users to change the configuration files to suit their own needs. The paper suggests changing the most important two parameters in the following way:

- Number of mappers: the input file size divided by the block size
- Number of reduces: 1.75 * (number of nodes * maximum number of reducers that can run in parallel)

The outcome of this change is to enable the user to run as many mappers and reducers in parallel as possible to fully utilize the available resources.

The paper proved the efficiency of this kind of change by comparing the test results against the Hadoop default configuration as well as the Clouadera’s one, and the results were as successful as expected. As far as their experiments are concerned, they have used one application which is CPU bound and another which is I/O dependent; so the two applications differed in their consumption profiles which gave them the conclusion that different application types expect different kind of configuration parameters.

Finally, and since the paper only discussed how to manually change parameters by testing the applications on the cluster, the authors suggested that Hadoop should offer a built-in configurator in their default installation. This suggestion was also a
contributor to this thesis motivation as my main aim was to save time for Hadoop users so that they can find the optimized parameters without having to test the application on the whole cluster and all the input.

Moreover, another paper titled: “Towards Optimizing Hadoop Provisioning in the Cloud” (Kambatla, (n.d.)) gets similarly motivated by the fact that one size does not fit all and that static default parameters cannot be suitable to all kinds of applications. The paper suggests that in order to maximize resource utilization, which is to keep all resources as much busy as possible, a consumption profile history of the application is needed.

The paper gives the following steps to get a consumption signature for a newly added application and thus to determine the optimum configuration parameter for the application:

1. Generate resource consumption signature by running the application against fewer nodes and a smaller data set
2. Match the resource consumption signature with other ones previously computed and stored in a database
3. Assign a signature according to the closest match

This method was limited by the fact that a typical Hadoop task cannot be given a signature as it is divided into three overlapping phases: map, copy, and reduce. The usual case was to be able to generate a signature for each phase but due to the limitation just mentioned, the paper just handles the overall Hadoop task.

The authors have chosen to compute consumption signatures by taking into account the duration of each interval of time and the resources consumed during that range. The comparison was conducted by calculating the vector distance between two signatures, and the test application will take the parameters from its best match.

The paper finally deduces that applications with similar signatures will have similar performances and incur similar bottlenecks if given similar parameters.

Finally, an article called “Advanced Hadoop Tuning and Optimizations” (Sharma, (n.d.)) also gives a similar motivation based on the fact that out of the box configuration is not always friendly, and states that Hadoop is difficult to debug and the performance tuning is kind of a black-art.
The solution suggested by the article is to change as many parameters as possible; the most important ones being the following:

- **mapred.compress.map.output:**
  - **Definition:** Map Output Compression
  - **Default:** false
  - **Suggestion:** it should be set to true for large clusters and large jobs to avoid the overhead of compression at mappers and decompression at reducers.

- **mapred.map/reduce.tasks.speculative.execution:**
  - **Definition:** Enable and disable map or reduce speculative execution
  - **Default:** true
  - **Suggestion:** if the average task completion time is greater than 1 hour and the application requires intense calculations and high throughput then this value should be set to false in order to avoid the execution of redundant tasks

- **mapred.tasktracker.map/reduce.tasks.maximum:**
  - **Definition:** Maximum tasks of map and reduce for a TaskTracker
  - **Default:** 2
  - **Suggestion:** for large clusters is should be in this range:
    - $\left[ \frac{(\text{cores per node})}{2}, 2 * (\text{cores per node}) \right]$

- **dfs.block.size:**
  - **Definition:** File system block size
  - **Default:** 67108864 bytes
  - **Suggestion:** if the cluster is small and the data set is large, this default size will create a large number of map tasks so the block size should be made larger to avoid the overhead of having too many map tasks
4 Problem and Proposed Solution

4.1 Statement of Problem

Hadoop is very efficient when it comes to applying data analytic on big data with a huge cluster; however, every application is different in terms of resource usage and bottlenecks. The Hadoop configuration parameters should be modified according to many criteria especially the expected resource consumption of test application. The issue here is that the user should update at least the most important parameters, such as the number of map tasks to spawn for a given job as well the number of map tasks that can run in parallel. Moreover, the number of reduce tasks is totally dependent on the number of map tasks launched; that leads us to put all the effort in finding the optimum number of map tasks that Hadoop should release for a new test application.

4.2 Proposed Solution

Before getting into the details of the solution, it would be important to mention how the idea developed and what were the main motivations driving this evolutionary process.

Basically, after being introduced to Hadoop and its great influence in industry; I have been motivated to discover more about it and to see how it really performed in some real world situations. This has pushed me to read more about its great success but which has also enabled me to spot some of the challenges being faced. I was very attracted to the fact that Hadoop can handle a wide variety of applications and inputs while it can still scale to huge clusters without much trouble; however, I found out that Hadoop, obviously, cannot run in the most optimized way for all those kinds of applications, in other words, some applications might run much faster if the configuration parameters were tuned beforehand. Finally, I have found what I was looking for, a challenge to be solved by developing an innovative solution that not only optimizes Hadoop but that also reduces the time required to find those optimized parameters.

The solution presented to the problem stated is based on estimating the optimum number of map tasks as well as the subset of them that should run in parallel at any
given time and at any TaskTracker; the solution takes advantage of the possibility to run Hadoop on a single node cluster just to categorize the applications in terms of CPU usage; once that step is completed, every application should be known to either being CPU-light or CPU-heavy. After that, special theories are stated and then parameters are deduced depending on the application’s category. Finally, the applications are run with the new parameters and then compared with the default ones that Hadoop offers.

4.3 Theoretical Assumptions

The theories that are behind the experiment are related to the relativity of the task setup overhead compared to the task overall execution time; for example, if a task is very light and takes only 4 seconds to finish its execution, then the task scheduling overhead which takes 2 seconds has to be minimized. Moreover, the task setup overhead might be negligible compared to the overall execution time, and then multiple tasks can be spawned in order to distribute the load among all the CPU thus keeping all of them as busy as possible. On the other hand, some of the light tasks might be better off scheduled concurrently on a single CPU rather than distributed among other ones; the theoretical proof is that the CPU needs many parallel light tasks in order to be fully utilized while it needs only one or two heavy tasks to reach that state, so the focus here is on utilizing the CPUs without experiencing any over-utilization nor under-utilization.

The theoretical assumptions stated can be summarized as follows:

- Heavy-CPU applications
  - Schedule few parallel tasks
    - The goal is to avoid over-utilizing the CPU
  - Spawn many tasks
    - To distribute the load evenly across CPUs, as the task setup overhead is negligible

- Light-CPU applications
  - Schedule many parallel tasks
    - In order to fully utilize the CPUs
  - Spawn few tasks
    - To avoid wasting time as the task setup overhead is very influential
4.4 Experiment Plan

The plan is to first categorize the application as either CPU-light or CPU-heavy by executing it on a single node cluster; then the theoretical assumptions stated will be applied and checked during the experiments on the multi-node cluster.

The problem faced here is how to control the number of map tasks launched by Hadoop since it is automatically generated depending on the input split and the HDFS block size. The solution to this problem was to play around with the input files using the simple Unix commands “split” and “cat”; this is efficient because all our inputs are text files as will be described later in the experiment.
5 Experiment Design: Single Node Cluster

5.1 Test Cases

5.1.1 Grep

The Grep MapReduce application, which is offered with the default Hadoop installation, counts the number of occurrences of each matching string in a text file; the map task is responsible for counting the number of times each match occurred in a single line while the reducer just sums up the values. The Grep application command only takes the regular expression as input; the number of maps gets generated automatically by Hadoop depending on the input splits. This application runs two jobs in sequence, one as a counter and a second as a sorter (Hadoop-Wiki, 2009).

5.1.2 PiEstimator

The PiEstimator is one of the MapReduce applications offered by Hadoop in its default installation directory; the applications mapper tries to estimate π by generating a given amount of random numbers and checking whether they are inside or outside the circle. On the other hand, the reducer accumulates the results generated from the mappers.

For example, assuming the total number is:

- TotalPoints = insidePoints + outsidePoints

The rational approximation of the value (Area of the circle)/(Area of the square) is the fraction (insidePoints/outsidePoints) given that the area of the circle is π/4 and the area of the unit square is 1; this will lead us to the estimation of π as:

- Estimated π = 4 *(insidePoints/outsidePoints)

This application takes two input arguments: the number of map tasks to launch as well as the number of samples to check for (Apache, 2009).
5.1.3 MRIF

The Map Reduce Integer Factorization application is an implementation of the Quadratic Sieve Algorithm that does integer factorization; each map task is responsible for performing a sieve on an interval and returning the smooth factors; on the other hand, the reducer tries to find a sub-set of them whose product is a square (Javier, 2009).

The following example demonstrates how the Quadratic Sieve Algorithm can be used to factorize $n = 16843009$ (PlanetMath, (n.d.)):

First, the integer closest to its square root has to be found, which is: 4104; then the factor base is $B = \{2, 3, 5, 7, 13\}$. After that the smooth values should be calculated according to the following equation: $y_i = f(x_i) = x_i^2 - n$.

Assume $x_0$ and $x_1$ were chosen and calculated to be:
- $x_0 = 4241$
- $x_1 = 4497$

Then $y_0$ and $y_1$ are 1143072 and 3380000, respectively; this will give us:
- $x = 4241 \cdot 4497 = 19071777$
- $y = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7^1 \cdot 13^1 = 1965600$

Finally, the greatest common divisors are computed to find the two factors:
- $\gcd(x-y, n) = 257$
- $\gcd(x+y, n) = 65537$

This application takes one input, which is the integer to be factorized, and then it generates one single file that Hadoop uses as an input split so only one map task gets launched; that input file includes the full intervals to be sieved.

5.2 Objective

The aim of this experiment is to categorize applications according to their CPU consumption; all the three applications: PiEstimator, Grep, and MRIF were run with default configurations on a single node cluster to analyze the ratio of their CPU utilization time against the overall execution period.
5.3 Setup

The single node cluster experiment is carried out in a personal computer that has a dual core CPU running at a clock rate of 1.73 GHz. The latest Hadoop and OpenJDK were installed on Linux Ubuntu; SSH was properly configured as it is the way Hadoop uses to manage its nodes.

After that, Hadoop was configured by modifying the configuration files as shown in Table 5.1. Finally, and after configuring Hadoop, the NameNode was formatted and the script for starting all the nodes was launched.

Table 5.1 Properties Added To One-Node Cluster Configuration

<table>
<thead>
<tr>
<th>Configuration File</th>
<th>Property Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>core-site.xml</td>
<td>The Hadoop temporary directory:</td>
</tr>
<tr>
<td></td>
<td>&lt;property&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;name&gt;hadoop.tmp.dir&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;value&gt;/app/hadoop/tmp&lt;/value&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/property&gt;</td>
</tr>
<tr>
<td>hdfs-site.xml</td>
<td>Name of default file system</td>
</tr>
<tr>
<td></td>
<td>&lt;property&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;name&gt;fs.default.name&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;value&gt;hdfs://localhost:54310&lt;/value&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/property&gt;</td>
</tr>
<tr>
<td>Mapred-site.xml</td>
<td>Where the JobTracker is located:</td>
</tr>
<tr>
<td></td>
<td>&lt;property&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;name&gt;mapred.job.tracker&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;value&gt;localhost:54311&lt;/value&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/property&gt;</td>
</tr>
<tr>
<td></td>
<td>Replication number of each block:</td>
</tr>
<tr>
<td></td>
<td>&lt;property&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;name&gt;dfs.replication&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;value&gt;1&lt;/value&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/property&gt;</td>
</tr>
</tbody>
</table>
5.4 PiEstimator

The PiEstimator, which takes the number of map tasks as input as well as the number of samples, was assigned 1 map task and 1000 samples. The command was given to Hadoop as follows:

- bin/hadoop jar *hadoop*examples*.jar pi 1 1000

5.5 Grep

Grep was assigned to look for the string “so” in 12 text books; Hadoop automatically assigned 12 map tasks due to the number of the input splits. Grep was launched with the following command:

- bin/hadoop jar *hadoop*examples*.jar grep /user/hduser/input_12 /user/hduser/output “so”

The previous command takes the path of the input inside HDFS and the output path also. Before having the input in HDFS, the following command was run in order to copy the files from the local file system to HDFS.

- bin/hadoop dfs –copyFromLocal /home/hduser/input_12 /user/hduser/input_12

5.6 MRIF

The MRIF application, as defined earlier, generates one input file; Hadoop consequently launches 1 single map task. Moreover, the integer chosen to be factorized was: 59595959595959595959; which is chosen on purpose to be large enough so that the Quadratic Sieve Algorithm generates lots of computations.

The command was given as follows:

- ./run.sh 59595959595959595959

The run.sh script included all the required information such as the path for the Hadoop binary.

5.7 Results

The aim of this phase was to categorize applications based on their CPU utilization, so MRIF was considered to be CPU-heavy while PiEstimator and Grep were both taken
to be CPU-light.

As far as the MRIF application is concerned, the percentage was calculated by dividing the field of “CPU time spent” by the total execution time; that information was taken from the output of Hadoop MapReduce illustrated in Figure 5.1. The job took 194 seconds and the CPU spent around 171 seconds; dividing the CPU time over the total time, we get:

- CPU Utilization = CPU time / Total Time = 170.910 / 194 = 88%

The result found proves that the MRIF application is CPU-heavy.

```
12/04/06 18:18:04 INFO input.FileInputFormat: Total input paths to process: 1
12/04/06 18:18:04 INFO mapred.JobClient: Running job: job_201204061501_0009
12/04/06 18:18:05 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 18:18:21 INFO mapred.JobClient: map 2% reduce 0%
12/04/06 18:18:24 INFO mapred.JobClient: map 4% reduce 0%
12/04/06 18:20:58 INFO mapred.JobClient: map 100% reduce 0%
    ...
12/04/06 18:21:13 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 18:21:18 INFO mapred.JobClient: Job complete: job_201204061501_0009
12/04/06 18:21:18 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 18:21:18 INFO mapred.JobClient: Launched map tasks=1
12/04/06 18:21:18 INFO mapred.JobClient: CPU time spent (ms) = 170910
    ...
Factor1  372731
Factor2  15896000015989
```

Figure 5.1 Part of MapReduce Output for the MRIF Application

On the other hand, while running the PiEstimator application and checking the results, it was clear enough that this application is CPU-light; the important fields of the MapReduce output is shown in Figure 5.2. The total execution time was around 33 seconds while the “CPU time spent” field indicated 1360 ms.

- CPU Utilization = CPU time / Total Time = 1.36 / 32.753 = 4%

This result proves that the PiEstimator is a CPU-light application.
In the same fashion, the CPU utilization of the Grep application was calculated; and as described earlier this application runs two jobs sequentially. After adding the total execution times for both jobs as well the CPU time spent from the MapReduce output shown in Figure 5.3, the application was categorized as CPU-light:

- Total Execution Time = First Job + Second Job = 43 seconds + 33 seconds = 76 seconds
- Total CPU Time = CPU Time First Job + CPU Time Second Job = 20910 ms + 5990 ms = 26900 ms
- CPU Utilization = Total CPU Time / Total Execution Time = 35%
Conclusively, the single-node cluster experiment has basically allowed us to categorize the three applications in order to be able to run the multi-node cluster with new updated parameters; the overall results of the last experiments are shown in the Figure 5.4.
Figure 5.4 The Applications’ CPU Utilization
Chapter 6 Experiment Design: Multi Node Cluster

6 Experiment Design: Multi-Node Cluster

6.1 Objective

The goal of this experiment is to take advantage of the previous experiments on the one node cluster in order to optimize Hadoop configuration for the new multi-node cluster. The final aim is to get better results with the new parameters than the default Hadoop ones.

6.2 Setup

The experiments were run on a cluster of three computers offered by the University inside the computer lab; each machine had a dual core CPU running at a clock rate of 1.86 GHz. All the machines had an Ubuntu Linux pre-installed and were all connected directly to a switch. Hadoop was then installed on all the three machines and configured as follows:

- One node
  - The master node and the slave also
  - The JobTracker
  - The NameNode
  - A DataNode
  - A TaskTracker
- Two nodes
  - Slave nodes
  - DataNodes
  - TaskTrackers

This kind of node setup was accomplished in multiple steps:

1. A single node cluster was setup on each machine, in order to make sure each computer is behaving properly while interacting with Hadoop.
2. The IP address of each node was added in the hosts-file of the other ones.
3. Permissions were given to the master node to allow it to access the other two slaves using SSH; this was done by copying the public SSH key of the master to the slaves.
4. The “masters” and “slaves” files were modified to include the hostnames of the nodes used.
   a. On the master node:
      i. The master hostname “hadoop3” was added to “masters” file.
      ii. The slaves’ hostnames were added to “slaves” file; which were “hadoop2” and “hadoop4”.

5. The configurations were modified to reflect the new URI of the file system and the one for the JobTracker; as now they are no longer referred to as “localhost” but rather as “hadoop3”; the changes were made across the files on all the three nodes. The properties modified are shown in Table 6.1.

6. Finally, the nodes were started using two scripts: “start-dfs.sh” to start HDFS nodes, and “start-mapred.sh” to start the MapReduce nodes; all scripts were run on the master node (hadoop3).

Table 6.1: Updates made to the configuration files of all cluster nodes

<table>
<thead>
<tr>
<th>Configuration File</th>
<th>Property Modified</th>
<th>Property Description</th>
</tr>
</thead>
</table>
| core-site.xml      | `<property>
fs.default.name
</name>
,value>
hdfs://hadoop3:54310
</value>
</property>` | The name of the default file-system. A URI whose authority is used to determine the host, port, etc. for a file system. |
| mapred-site.xml    | `<property>
mapred.job.tracker
</name>
,value>
hadoop3:54311
</value>
</property>` | The hostname and the port number where the JobTracker resides. |
6.3 PiEstimator

The PiEstimator, which was categorized as a CPU-light application, was executed given a different number of map tasks in the command line: 1, 3, 6, and 12; moreover, it was always assigned the same number of samples to look for: 1000. This way, it was guaranteed that the only change made is the number of map tasks spawned by Hadoop.

Moreover, to check for the best number of parallel tasks, the PiEstimator was run with 12 map tasks twice; the first time with 4 parallel map tasks at each node, and the second time with no parallelism at all. The number 4 of course was not chosen randomly, it is because we have three nodes and we would like to test the application with full parallelism while still fully utilizing all CPU resources.

The field of “mapred.tasktracker.map.tasks.maximum” inside the file “mapred-site.xml” was modified and the cluster was restarted each time in order for the changes to take effect.

6.4 Grep

Grep, which also was a CPU-light application, was assigned a different number of map tasks, 1, 3, 6, and 12 but always the same string to search for: “so”. The input was 12 text books so Hadoop chose automatically 12 map tasks taking into consideration the number of input splits. In order to circumvent that, the Unix “cat” command was used to merge two files together in order to have 6 input splits; then to merge 4 files together to get 3 input splits; and finally to merge all the 12 books together to get 1 input split.

Moreover, to check parallelism for this application, the application was run twice with 12 map tasks, but once with 4 parallel map tasks at each node, and another time with no parallelism at all. In the same fashion as the previous application, the parameter field of “mapred.tasktracker.map.tasks.maximum” inside the file “mapred-site.xml” was modified and the cluster was restarted each time in order for the changes to take effect.

6.5 MRIF

The MRIF application, which was the only application categorized as CPU-heavy,
was launched with a different number of map tasks: 1, 4, 7, and 12. This application had only 1 input file, but it was necessary to split the file in order to force Hadoop to pick the number of map tasks required for the experiment. First of all, the application was run with that input file in order to generate 1 map task; then it was split using the Unix “split” command given the number of lines in order to have 4 input files, then 7 ones, and finally 12 ones.

Moreover, parallelism was checked by running the application twice with 12 map tasks but the first time with 4 parallel tasks and the second one with only 2 ones. This was chosen in order to show that even though parallelism might be good, too much of it will hurt.

6.6 Results

The results found can be better presented if shown for each application separately, so the next three sections will demonstrate the results for the PiEstimator, Grep, and MRIF.

6.6.1 PiEstimator

Figure 6.1 illustrates the results for the PiEstimator; it shows how the total execution time changes when the number of map tasks spawned is changed.
Figure 6.1 PiEstimator execution time according to the number of map tasks

Figure 6.1 proves the assumptions and theories declared before as it was shown that more map tasks will simply add more overhead for a light application that barely takes a little time to finish execution. The point here is that the best results are found by having between 1 map task and 3 ones, and that is due to the fact that the cluster has three nodes; moreover, and as declared before, the resources have to be fully utilized; in this case, assigning 1 CPU to do all the job is nearly as good in performance as assigning 3 CPUs. However, assigning more map tasks than the number of CPUs will simply add too much setup overhead. The execution time for 1 map task was exactly 32.753 while the one with 3 map tasks was 32.683; that simply mean that the overhead of assigning two more tasks to be run at the same time by two other CPUs is the same as waiting for one CPU to finish all those jobs by itself.

The most important MapReduce output fields concerning the PiEstimator for 1 map, 3 maps, 6 maps, and 12 maps are illustrated respectively in Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5.
Chapter 6 Experiment Design: Multi Node Cluster

Figure 6.2 Part of MapReduce output for running PiEstimator with 1 Map

```
# bin/hadoop jar *hadoop*examples*.jar pi 1 1000

Number of Maps = 1
Samples per Map = 1000
Starting Job
12/04/06 15:02:24 INFO mapred.FileInputFormat: Total input paths to process: 1
12/04/06 15:02:24 INFO mapred.JobClient: Running job: job_201204061501_0001
12/04/06 15:02:25 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 15:02:39 INFO mapred.JobClient: map 100% reduce 0%
12/04/06 15:02:51 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 15:02:56 INFO mapred.JobClient: Job complete: job_201204061501_0001
12/04/06 15:02:56 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 15:02:56 INFO mapred.JobClient: Launched map tasks=1
12/04/06 15:02:56 INFO mapred.JobClient: CPU time spent (ms) =1360
Job Finished in 32.753 seconds
Estimated value of Pi is 3.148000000000000
```

Figure 6.3 Part of MapReduce output for running PiEstimator with 3 Maps

```
# bin/hadoop jar *hadoop*examples*.jar pi 3 1000

Number of Maps = 3
Samples per Map = 1000
Starting Job
12/04/06 15:03:09 INFO mapred.FileInputFormat: Total input paths to process: 3
12/04/06 15:03:09 INFO mapred.JobClient: Running job: job_201204061501_0002
12/04/06 15:03:10 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 15:03:24 INFO mapred.JobClient: map 33% reduce 0%
12/04/06 15:03:25 INFO mapred.JobClient: map 100% reduce 0%
12/04/06 15:03:36 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 15:03:41 INFO mapred.JobClient: Job complete: job_201204061501_0002
12/04/06 15:03:41 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 15:03:41 INFO mapred.JobClient: Launched map tasks=3
12/04/06 15:03:41 INFO mapred.JobClient: CPU time spent (ms) =3370
12/04/06 15:03:41 INFO mapred.JobClient: Map output records=6
Job Finished in 32.683 seconds
Estimated value of Pi is 3.138000000000000
```
Chapter 6 Experiment Design: Multi Node Cluster

Figure 6.4 Part of MapReduce output for running PiEstimator with 6 Map

```bash
# bin/hadoop jar *hadoop*examples.jar pi 6 1000
```

Number of Maps = 6
Samples per Map = 1000
Starting Job
12/04/06 15:03:49 INFO mapred.FileInputFormat: Total input paths to process: 6
12/04/06 15:03:50 INFO mapred.JobClient: Running job: job_201204061501_0003
12/04/06 15:03:51 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 15:04:04 INFO mapred.JobClient: map 33% reduce 0%

12/04/06 15:04:22 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 15:04:27 INFO mapred.JobClient: Job complete; job_201204061501_0003
12/04/06 15:04:27 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 15:04:27 INFO mapred.JobClient: Launched map tasks=6
12/04/06 15:04:27 INFO mapred.JobClient: CPU time spent (ms)=5170
Job Finished in 37.747 seconds
Estimated value of Pi is 3.13800000000000000000

Figure 6.5 Part of MapReduce output for running PiEstimator with 12 Map

```bash
# bin/hadoop jar *hadoop*examples.jar pi 12 1000
```

Number of Maps = 12
Samples per Map = 1000
Starting Job
12/04/06 15:04:35 INFO mapred.FileInputFormat: Total input paths to process: 12
12/04/06 15:04:35 INFO mapred.JobClient: Running job: job_201204061501_0004
12/04/06 15:04:36 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 15:04:52 INFO mapred.JobClient: map 16% reduce 0%

12/04/06 15:05:08 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 15:05:13 INFO mapred.JobClient: Job complete; job_201204061501_0004
12/04/06 15:05:13 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 15:05:13 INFO mapred.JobClient: Launched map tasks=12
12/04/06 15:05:13 INFO mapred.JobClient: CPU time spent (ms)=17620
Job Finished in 38.236 seconds
Estimated value of Pi is 3.14800000000000000000

In the same fashion, parallelism results were also successful as the total execution time for PiEstimator with multiple parallel map tasks executed faster than without; Figure 6.6 shows how 12 map tasks with 4 parallel ones executed faster than 12 map tasks without any parallelism.
Chapter 6 Experiment Design: Multi Node Cluster

The figure also proves the fact that CPU-light applications need as much parallelism as possible, of course without over-utilizing the resources; in our cluster of three nodes and our experiment with 12 map tasks, the best thing is to use all the CPUs evenly, so that is why 3 parallel tasks were run at each node. The application once run with parallelism enabled finished executing in 43 seconds while the unparallel version executed in 54 seconds.

Figure 6.6 PiEstimator execution time based on parallelism

6.6.2 Grep

The Grep CPU-light application has also proved the previous theories and assumptions stated; it has run faster when launched with 1 or 3 map tasks rather than 6 or 12 ones. Figure 6.7 shows how the execution time is influenced by the number of map tasks spawned. As the PiEstimator application, which was also CPU-light, the execution of Grep with 1 or 3 map tasks was much better than 6 or 12 ones; again, that is due to the well balanced utilization of resources, that either uses the 3 CPUs or finished all by 1 CPU rather than launching more tasks than CPUs generating a lot of setup overhead.
Figure 6.7 Grep execution time according to the number of map tasks

The most important MapReduce output fields concerning the Grep application for 1 map, 3 maps, 6 maps, and 12 maps are illustrated respectively in Figure 6.8, Figure 6.9, Figure 6.10, and Figure 6.11.

**FIRST JOB**

12/04/03 18:35:10 INFO mapred.FileInputFormat: Total input paths to process: 1
12/04/03 18:35:11 INFO mapred.JobClient: Running job: job_201204031800_0007
12/04/03 18:35:12 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:35:25 INFO mapred.JobClient: map 50% reduce 0%
12/04/03 18:35:27 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:35:37 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:35:42 INFO mapred.JobClient: Job complete: job_201204031800_0007
12/04/03 18:35:42 INFO mapred.JobClient: Launched reduce tasks=1
12/04/03 18:35:42 INFO mapred.JobClient: Launched map tasks=1
12/04/03 18:35:42 INFO mapred.JobClient: CPU time spent (ms) =4440

**SECOND JOB**

12/04/03 18:35:42 INFO mapred.FileInputFormat: Total input paths to process: 1
12/04/03 18:35:42 INFO mapred.JobClient: Running job: job_201204031800_0008
12/04/03 18:35:43 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:35:56 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:36:08 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:36:13 INFO mapred.JobClient: Job complete: job_201204031800_0008
12/04/03 18:36:13 INFO mapred.JobClient: Launched map tasks=1
12/04/03 18:36:13 INFO mapred.JobClient: CPU time spent (ms) =6510

Figure 6.8 Part of MapReduce output for running Grep with 1 Map
Chapter 6 Experiment Design: Multi Node Cluster

**FIRST JOB**

12/04/03 18:37:59 INFO mapred.FileInputFormat: Total input paths to process: 3
12/04/03 18:37:59 INFO mapred.JobClient: Running job: job_201204031800_0009
12/04/03 18:38:00 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:38:13 INFO mapred.JobClient: map 33% reduce 0%
12/04/03 18:38:14 INFO mapred.JobClient: map 66% reduce 0%
12/04/03 18:38:15 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:38:25 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:38:30 INFO mapred.JobClient: Job complete: job_201204031800_0009
12/04/03 18:38:30 INFO mapred.JobClient: Launched reduce tasks=1
12/04/03 18:38:30 INFO mapred.JobClient: CPU time spent (ms) = 7290

**SECOND JOB**

12/04/03 18:38:30 INFO mapred.FileInputFormat: Total input paths to process: 1
12/04/03 18:38:32 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:38:45 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:38:57 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:39:02 INFO mapred.JobClient: Job complete: job_201204031800_0010
12/04/03 18:39:02 INFO mapred.JobClient: Counters: 20
12/04/03 18:39:02 INFO mapred.JobClient: Job Counters
12/04/03 18:39:02 INFO mapred.JobClient: Launched reduce tasks=1
12/04/03 18:39:02 INFO mapred.JobClient: Launched map tasks=1
12/04/03 18:39:02 INFO mapred.JobClient: CPU time spent (ms) = 5980

Figure 6.9 Part of MapReduce output for running Grep with 3 Maps

**FIRST JOB**

12/04/03 18:40:40 INFO mapred.FileInputFormat: Total input paths to process: 6
12/04/03 18:40:42 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:40:56 INFO mapred.JobClient: map 33% reduce 0%
12/04/03 18:41:00 INFO mapred.JobClient: map 66% reduce 0%
12/04/03 18:41:01 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:41:05 INFO mapred.JobClient: map 100% reduce 11%
12/04/03 18:41:14 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:41:19 INFO mapred.JobClient: Job complete: job_201204031800_0011
12/04/03 18:41:19 INFO mapred.JobClient: Launched reduce tasks=1
12/04/03 18:41:19 INFO mapred.JobClient: Launched map tasks=5
12/04/03 18:41:19 INFO mapred.JobClient: CPU time spent (ms) = 13670

**SECOND JOB**

12/04/03 18:41:19 INFO mapred.FileInputFormat: Total input paths to process: 1
12/04/03 18:41:19 INFO mapred.JobClient: Running job: job_201204031800_0012
12/04/03 18:41:20 INFO mapred.JobClient: map 0% reduce 0%
12/04/03 18:41:33 INFO mapred.JobClient: map 100% reduce 0%
12/04/03 18:41:46 INFO mapred.JobClient: map 100% reduce 100%
12/04/03 18:41:51 INFO mapred.JobClient: Job complete: job_201204031800_0012
12/04/03 18:41:51 INFO mapred.JobClient: Launched reduce tasks=1
12/04/03 18:41:51 INFO mapred.JobClient: Launched map tasks=1
12/04/03 18:41:51 INFO mapred.JobClient: CPU time spent (ms) = 6530

Figure 6.10 Part of MapReduce output for running Grep with 6 Maps
Figure 6.11 Part of MapReduce output for running Grep with 12 Maps

After checking the success of the Grep experiments concerning the number of tasks; the right level of parallelism was also checked and the results are shown in Figure 6.12. The figure basically shows that 12 map tasks with 4 parallel maps allowed at each node executes in 73 seconds while the same number of map tasks when launched sequentially spend 86 seconds to finish execution. The experiment has proved the theory stating the fact that CPUs can only be kept busy by executing multiple light map tasks as a specific CPU will be wasted and under-utilized if assigned only 1 light map task.
7.6.3 MRIF

The MRIF application has demonstrated that a heavy-CPU application should be executed with a high number of map tasks, and a low level or parallelism, exactly like what the previous theories and assumptions stated; Figure 6.13 shows the results of running MRIF with 1 map, 4 maps, 7 maps, and 12 maps. The results were as expected; a heavy-CPU application needs a high number of map tasks in order to balance the load effectively. In other words, the experimental results show that the more map tasks you have the faster the execution you get, and that is due, as stated before, to the negligible time required for setting up a task and the need to distribute the heavy load evenly as much as possible to avoid over-utilizing a certain CPU.
Figure 6.13 MRIF execution time according to the number of map tasks

The MapReduce output of MRIF when executed with 1 map, 4 maps, 7 maps, and 12 maps are illustrated in the following figure respectively: Figure 6.14, Figure 6.15, Figure 6.16, and Figure 6.17.

```
hduser@ubuntu:/mrfB ~$ sr.sh 5959595959595959595959595959595959
信息: Attempting factorization of 5959595959595959595959595959595959
信息: Factor base of size: 108
信息: Sieve of size: 1259712
12/04/06 18:18:04 INFO input.FileInputFormat: Total input paths to process: 1
12/04/06 18:18:04 INFO mapred.JobClient: Running job: job_201204061501_0009
12/04/06 18:18:05 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 18:18:21 INFO mapred.JobClient: map 2% reduce 0%
12/04/06 18:18:24 INFO mapred.JobClient: map 4% reduce 0%
....
12/04/06 18:20:43 INFO mapred.JobClient: map 93% reduce 0%
12/04/06 18:20:52 INFO mapred.JobClient: map 99% reduce 0%
12/04/06 18:20:58 INFO mapred.JobClient: map 100% reduce 0%
12/04/06 18:21:13 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 18:21:18 INFO mapred.JobClient: Job complete: job_201204061501_0009
12/04/06 18:21:18 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 18:21:18 INFO mapred.JobClient: Launched map tasks=1
12/04/06 18:21:18 INFO mapred.JobClient: CPU time spent (ms) =170910

Factor1 372731
Factor2 159890000015989
```

Figure 6.14 Part of MapReduce output for running Grep with 1 Map
Chapter 6 Experiment Design: Multi Node Cluster

Figure 6.15 Part of MapReduce output for running MRIF with 4 Maps

hduser@ubuntu:/mrif$ /run.sh 595959595959595959595959595959
信息: Attempting factorization of: 595959595959595959595959595959
信息: Factor base of size: 108
信息: Sieve of size: 1259712
12/04/06 18:35:23 INFO input.FileInputFormat: Total input paths to process: 4
12/04/06 18:35:23 INFO mapred.JobClient: Running job: job_201204061501_0010
12/04/06 18:35:24 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 18:35:40 INFO mapred.JobClient: map 29% reduce 0%
12/04/06 18:35:43 INFO mapred.JobClient: map 33% reduce 0%
....
12/04/06 18:36:35 INFO mapred.JobClient: map 98% reduce 16%
12/04/06 18:36:38 INFO mapred.JobClient: map 100% reduce 16%
12/04/06 18:36:46 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 18:36:51 INFO mapred.JobClient: Job complete: job_201204061501_0010
12/04/06 18:36:51 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 18:36:51 INFO mapred.JobClient: Launched map tasks=4
12/04/06 18:36:51 INFO mapred.JobClient: CPU time spent (ms)=164790

Factor1  21991129
Factor2  271000000271

Figure 6.16 Part of MapReduce output for running MRIF with 7 Maps

hduser@ubuntu:/mrif$ /run.sh 595959595959595959595959595959
信息: Attempting factorization of: 595959595959595959595959595959
信息: Factor base of size: 108
信息: Sieve of size: 1259712
12/04/06 18:44:03 INFO input.FileInputFormat: Total input paths to process: 7
12/04/06 18:44:03 INFO mapred.JobClient: Running job: job_201204061501_0011
12/04/06 18:44:04 INFO mapred.JobClient: map 0% reduce 0%
12/04/06 18:44:20 INFO mapred.JobClient: map 15% reduce 0%
12/04/06 18:44:22 INFO mapred.JobClient: map 20% reduce 0%
12/04/06 18:44:23 INFO mapred.JobClient: map 27% reduce 0%
....
12/04/06 18:45:06 INFO mapred.JobClient: map 98% reduce 19%
12/04/06 18:45:09 INFO mapred.JobClient: map 100% reduce 19%
12/04/06 18:45:18 INFO mapred.JobClient: map 100% reduce 100%
12/04/06 18:45:23 INFO mapred.JobClient: Job complete: job_201204061501_0011
12/04/06 18:45:23 INFO mapred.JobClient: Launched reduce tasks=1
12/04/06 18:45:23 INFO mapred.JobClient: Launched map tasks=7
12/04/06 18:45:23 INFO mapred.JobClient: CPU time spent (ms)=175040

Factor1  5718321949
Factor2  10421931491
In a similar way, the results of the parallelism tests were also successful in matching the theories and assumptions declared before. The Figure 6.18 shows that the execution time for an MRIF application with 12 map tasks spawned allowing 2 parallel tasks at each node is much faster than allowing 4 parallel maps; which was expected as too much parallelism will basically over-utilize the CPUs unlike the light-CPU applications which have to be executed with much parallelism to fully utilize a certain CPU. As shown in the related figure, the MRIF version with 2 parallel map tasks at each node executed in 70 seconds while the version with 4 parallel map tasks executed in 78 seconds.
Figure 6.18 MRIF execution time based on parallelism
7 Conclusion and Future Work

7.1 Conclusion

The problem of data analytic of huge amounts of input has been kind of solved by the Hadoop framework which splits inputs and jobs to multiple nodes in order to execute tasks in a parallel fashion; this new paradigm, which scales to thousands of nodes without much overhead, needs to be tuned and optimized for each cluster depending on the application resource consumption. In this paper, a new way of controlling the number of map tasks was introduced; which was based on changing the number of splits using simple UNIX commands. Moreover, the paper discussed how optimizing the number of map tasks based on the CPU utilization statistics can give amazing results concerning the total execution time. The theories introduced and proved in this paper can be summarized in the following statements:

- Light-CPU applications:
  - Spawn as many map tasks as the number of CPUs
    - \( \text{Num\_maps} \leq \text{Num\_CPU} \)
  - In case too many maps have to be run, run as many in parallel per node as possible to avoid any CPU under-utilization
    - \( \text{Num\_parallel\_maps} = \frac{\text{Num\_maps}}{\text{Num\_CPU}} \)

- Heavy-CPU applications:
  - Spawn at least a number of map tasks equal to the number of CPUs, but the more the better performance
    - \( \text{Num\_maps} \geq \text{Num\_CPU} \)
  - Control the level of parallelism by not exceeding 2 parallel maps on each CPU to avoid any CPU over-utilization
    - \( \text{Num\_parallel\_maps} \leq 2 \)

7.2 Future work

The current research conducted in this thesis only discusses how to optimize the number of map tasks given the CPU utilization; it is planned though, in the near future, to include a possible way of optimizing also the number of reduce tasks. Moreover,
Chapter 7 Conclusion and Future Work

there is a good chance of including “Ganglia” (Massie, 2001), which is a famous cluster monitoring tool that might display more accurate results concerning the cluster performance. If time permits, the “Cloudera” configuration tool (Cloudera, 2011) will be incorporated in order to compare their configuration parameters to the ones found in this paper.
Acknowledgments

Any kind of innovative and original work has some key players that only act behind the scenes; this thesis was the outcome of the support offered by my supervisor the professor Qin Liu as well as my co-supervisors Fion Yang and Zhu HongMing. Moreover, I would like to thank Meihui Li whose office was always open for me whenever I needed help regarding practical matters in China.

I would also like to show my admiration to the government of China for welcoming foreign students and for managing such an amazing country. I also feel the urge to show my total appreciation towards the Kingdom of Sweden for giving such a great education; besides, I would like to personally thank Anders Berglund for his professional cooperation.

I would also like to thank my parents, Driss Benslimane and Jalila Yedji, for their financial and moral support throughout my whole educational studies. Moreover, I would like to thank both of my companions, Malak Benslimane and Kenza Ouaheb, for their continuous care.
References


[20] Hansen, A., (n.d.), *Optimizing Hadoop for the Cluster*, University of Tromso


Appendix A The MapReduce Process

The example above demonstrates how the MapReduce process is applied to an input using a simple application that counts the occurrences of words. The whole process can be divided to the following steps (Rabidgremlin, n.d.):

1. Splitting
   The input gets split based on the number of lines

2. Mapping
   The mapping phase applies a function on the input split; in this case, it counts the number of occurrences of each word
3. Shuffling

In this step, all the nodes exchange the values between each other, so that the
values with the same keys are put together; in this case, each word being
counted is put together with its equivalents from the other lines.

4. Reducing

During this step, all the values with the same key get summed up in order to get the
final answer.
Appendix B The Quadratic Sieve Algorithm

The Quadratic Sieve Algorithm can be used to factorize integers; the following is a walkthrough example that demonstrates exactly how this algorithm works (Wikipedia, 2012).

1. The algorithm first received the input which is in this case:
   a. \( n = 15347 \)

2. The range to sieve for the first 100 numbers
   a. \( V = [Y(0) Y(1) Y(2) Y(3) \ldots Y(99)] \)
      Where \( Y(x) = (x + (\sqrt{n}))^2 - n \)

3. The solutions for \( x^2 \equiv n \pmod{p} \) are then found in order to get the first prime numbers for which \( n \) has a square root modulo \( p \).

<table>
<thead>
<tr>
<th>p</th>
<th>2</th>
<th>17</th>
<th>23</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>13</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

4. The algorithm performs the sieve:
   a. For each \( p \) the following equation is solved:
      i. \( Y(x) \equiv (x + (\sqrt{n}))^2 - n \equiv 0 \pmod{p} \)
         \( \Rightarrow x \equiv \sqrt{n} - 124 \pmod{p} \)
      ii. Each equation \( x \equiv a \pmod{p} \) results in \( Vx \) being divisible by \( p \) at
          \( x = a \)
          1. \( V = [1 \; 139 \; 23 \; 1 \; 61 \; 647 \; \ldots \; 17191] \)
      iii. Entries where \( V \) equals to 1 is a smooth number

<table>
<thead>
<tr>
<th>x + 124</th>
<th>Y</th>
<th>Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>29</td>
<td>( 2^0 \cdot 17^0 \cdot 23^0 \cdot 29^1 )</td>
</tr>
<tr>
<td>127</td>
<td>782</td>
<td>( 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^0 )</td>
</tr>
<tr>
<td>195</td>
<td>22678</td>
<td>( 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^1 )</td>
</tr>
</tbody>
</table>

5. Matrix processing
   a. \( 29 = 2^0 \cdot 17^0 \cdot 23^0 \cdot 29^1 \)
   b. \( 782 = 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^0 \)
   c. \( 22678 = 2^1 \cdot 17^1 \cdot 23^1 \cdot 29^1 \)

6. Matrix notation:
a. $S \equiv [(0 \ 0 \ 0 \ 1),(1 \ 1 \ 1 \ 0),(1 \ 1 \ 1 \ 1)] \equiv [0 \ 0 \ 0 \ 0] \pmod{2}$

b. $S = [1 \ 1 \ 1]$

7. The product of the three equations is then:
   - $29 \cdot 782 \cdot 22678 = 22678^2$
   - $124^3 \cdot 127^2 \cdot 195^2 = 3070860^2$

8. The algorithm finally finds:
   a. $22678^2 \equiv 3070860^2 \pmod{15347}$

9. The final results comes from the following test:
   a. $\text{GCD}(3070860 - 22678, 15347)$
      i. Factor 1 = 103
      ii. Factor 2 = 149
Resume and Research Papers Published

Resume:
Ziad Benslimane, Male, 15 September 1985

- Educational Background:
  - September 2011 - Entry to Tongji University Master of Software Engineering
  - August 2010 - Entry to Uppsala University Master of Computer Science
  - August 2009 - Graduated from University of Texas at Arlington with a Bachelor of Science in Computer Engineering

- Experience:
  - April 2012 – Current day: Intern at the Cloud Computing department of Ebay
  - June 2009 – November 2009: Software Tester at Bynari in Dallas, Texas.
  - May 2009 – August 2009: Research Assistant in the Multimedia Department (University of Texas at Arlington)

Research papers published: