

Optimizing Hadoop for the cluster

Christer A. Hansen^{*}
Institute for Computer Science
University of Tromsø, Norway
hansen.christer@gmail.com

ABSTRACT

The total number of clusters running Hadoop increases every day. The reason for this is that companies have found a simple model that works well. The model is built to work on thousands of machines and massive data sets. MapReduce is a scalable and fault-tolerant model that hides all the dirty work for the programmers. Since Hadoop is being installed on more and more clusters, configuration becomes an important topic. This fact inspired us to find out if we are able to optimize Hadoop to run faster. We have run experiments on different configurations and found that Hadoop should be configured in a way that it utilizes the resources the cluster has to offer. The default configuration of Hadoop has proven to be slow. We use our own configuration and Cloudera's Configurator to prove that performance may increase significantly by just changing a set of parameters.

Keywords

Configuring Hadoop, Cloudera

1. INTRODUCTION

Configuring Hadoop to perform well on a cluster is like tuning a car. There are several switches and knobs that needs to be switched and turned to get it right. The downloadable Hadoop distribution comes with a default configuration. This configuration is not optimized for any given cluster. Every cluster consist more or less of a different number of machines and different hardware. This means that each Hadoop framework should be optimized for its unique cluster setup. In this paper we show the importance of adapting the Hadoop framework to work well on its cluster. Our goal was to get better performance than the default configuration. We used a special tool developed for this purpose, called Cloudera's Configurator, to give us some hints on how to configure Hadoop.

Section 2 describes some basic concepts Hadoop introduces to implement the MapReduce programming model. Section 3 describes how configuration of Hadoop is done, and the purpose of Cloudera's Configurator. Section 4 describes the environment the experiments were executed in. It also describes how we decided the number of mapper tasks and reducer tasks. The results of the experiments are included in this section as well. Section 5 describes and analyses the results we got from our experiments. Section 6 describes

^{*}This paper is written as the second mandatory assignment in the course INF-3203 - Advanced Distributed Systems offered at the University of Tromsø.

briefly some related work. Section 7 concludes this paper and suggests some future work.

2. TERMINOLOGY

The Hadoop open-source framework consists of a distributed file system and the programming model itself. The file system is called Hadoop Distributed File System (HDFS) and is similar to the Google File System (GFS) [4]. Hadoop is based on the MapReduce programming model designed by Google [3]. In this section we provide some high level terminology regarding MapReduce and Hadoop.

A typical Hadoop job is as follows: The Hadoop framework is called within a regular program. The user program the map and reduce functions, and suggests how many mapper and reducer tasks that should be executed. The input file is split up into 64MB blocks by default (but is configurable), and is distributed among the slaves. The files are stored on the same machines that execute the mapper tasks. The result of a map task is a set of intermediate key/value pairs, this data is not stored in the distributed file system but on the local file system of the slave. The locations of these intermediate key/value pairs are passed back to the master which is responsible for forwarding these locations to the reduce workers. When the reduce workers receive these locations from the master they contact the slave and reads the intermediate data. The reduce worker then iterates over the intermediate data and stores the results to an output file. Control will be given back to the user program.

The Hadoop framework introduces some concepts of workers and processes to solve these issues.

2.1 The name node

The name node is part of the masters and is responsible for the file system name space. When a file is stored in the HDFS it is split up into smaller blocks and distributed among the data nodes. The name node holds the mapping from files to blocks in memory and in a persistent meta data store on disk, but the mapping between the blocks and the data nodes they reside on is only stored in the name node's memory, not persistently [5].

2.2 The secondary name node

The name node maintains two on-disk data structures to represent the file system's state: an image file and an edit log. The image file is a checkpoint of the file system's meta data. Changes to the file system's meta data are written to the edit log. When the name node boots it reconstructs the current state by replaying the edit log. At periodic intervals

a new checkpoint is created, this is done by applying the edit log to the image. This is performed by the secondary name node, and is often performed on a separate machine.

2.3 The data node

The data nodes are responsible for storing the blocks in their local file system. When a data node starts up it immediately tells the name node which blocks it has stored locally. This is important because the name node must have a clear picture of the block distribution across the cluster.

2.4 The task tracker

The task tracker is a process located on each slave. A slave in the cluster acts as both a task tracker and a data node. The task tracker receives map or reduce jobs from the job tracker and executes them. When the job is done the task tracker informs the job tracker where for instance the intermediate data is located after a finished map task. The job tracker goes back to idle and waits for more incoming work.

2.5 The job tracker

The job tracker process is part of the masters (the name node and the job tracker does not have to be running on the same node). The job tracker is responsible for scheduling and sending out jobs to the task trackers. It must also be able to handle failing jobs and failing slaves. Shipping jobs where the data is located means that the job tracker needs to know where every single block of data is located in the cluster.

3. CONFIGURING HADOOP

Parameters deciding how the Hadoop framework should behave is given in a set of configuration files. The files are located in the `conf/` directory of the distribution. XML formatting is used for defining and setting the values of the resources. The `hadoop-default.xml` contains the default configuration and should not be modified. When change in configuration is needed it should be done by modifying the `hadoop-site.xml` file. This file overrides the `hadoop-default.xml` file. In our experiments we only configured the `hadoop-site.xml` file.

3.1 Cludera's configuration tool

Cludera is a company working on issues concerning optimization of Hadoop [1]. They provide a tool called the Cludera Configurator. The tool is a web-based application that takes input from the user and generates a set of configuration files. These files are generated to fit a specific cluster and its hardware components.

The Cludera Configurator works as follows: (1) The name of the cluster and the number of nodes must be inserted. (2) Then specific information about the name node must be inserted, such as the host name, number of cores, how much RAM, and the path to the meta data files. (3) In this section the secondary name node's meta data path must be provided. (4) In this step, information about the job tracker must be provided. (5) Hardware information about the slaves such as number of cores, RAM, and disk space must be inserted in this section. (6) Now the Configurator has all the information about the cluster it needs to create the configuration files. Configuration files are created for the client, the slaves and the name node.

We have tried Cludera's Configurator on our cluster to see if we could get any performance gain, the results can be found in the experiments section.

4. EXPERIMENTS

In this section we present our results of the performance experiments we have executed with the different configurations. We will compare the default Hadoop configuration with Cludera's configuration. We will also suggest our own configuration and compare it against the two before mentioned settings.

The selected programs used for benchmarking are WordCount and PiEstimator¹. The reason why we chose these two applications is because they are quite different from each other.

4.1 Cluster information

All the experiments were executed on a cluster that consisted of 10 machines. Each machine had two 2,66GHz Quad-Core Intel Xeon processors, 16GB of memory, one 250GB disk, and a gigabit Ethernet link.

In our cluster we run the name node, the secondary name node, and the job tracker on the same machine. The master node containing all these processes does not run any mapper or reducer tasks.

The data nodes and the task trackers are deployed on the same machine. In our cluster we have 10 machines that run as both data nodes and task trackers. We have conducted all our experiments on these 10 machines.

The experiments were executed when the machines were in idle. The machines we used were only a subset of a larger cluster, so network traffic was not controlled under the experiments.

4.2 How many Maps and Reduces?

The question about how many mapper and reducer tasks a job should consist of is important. Picking the right amount of tasks for a job can have a huge impact on the Hadoop performance. The optimum occur when every machine in the cluster has something to do at any given time when a job is executed. To provide this, load-balancing must be considered.

The mapper tasks often work on a lot of data, and the result of those tasks are passed to the reducer tasks. A reduce task is often just an aggregate function that works on a minor portion of data compared to the mapper tasks. This implies that the number of reducer tasks should not be as high as the number of mapper tasks.

The number of mappers are usually driven by the distributed file systems block size. For instance if the input file is 2GB and the block size is 64MB, the number of maps will be set to 32 (2000MB / 64MB). The reason why this is a good solution is because each mapper task will get its own block to work on. We followed this approach when we did the WordCount experiments. [2] suggests that each machine should execute between 10 and 100 mapper tasks.

Correct number of reducers must also be considered before executing a job. A function given as $1.75 \cdot (\text{number of nodes} \cdot \text{mapred.tasktracker.reduce.tasks.maximum})$ will make the faster nodes finish their initial round of reduces

¹Both these programs are located in the `hadoop-*-examples.jar` file.

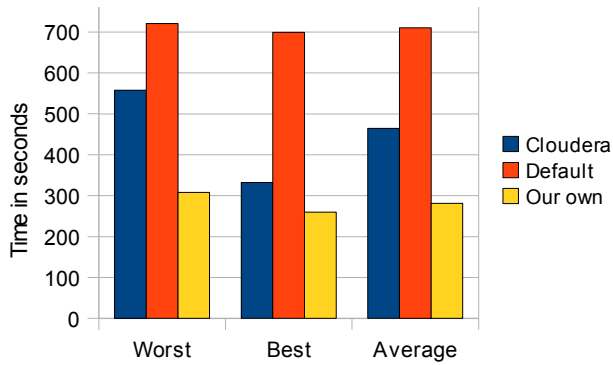


Figure 1: The WordCount results. The figure shows the worst, the best and the average times for Cloudera’s configuration, default configuration, and our own configuration.

first, when the initial round is done they immediately start a new round. Load-balancing will be provided by following this approach [2]. We followed this approach in the WordCount experiments.

4.3 PiEstimator

PiEstimator is an application that tries to estimate the value of π using the Monte-Carlo method. The approach is to generate random numbers and check whether they are on the inside or outside of a circle. A single map task is given a number of samples, this number is how many times it should try to hit the circle. The number of hits and misses are passed to a single reduce task that estimates π . PiEstimator is not dependent on input data as the WordCount application is.

In the PiEstimator experiments we told the application to execute 80 mapper tasks in total. We also told the application that each map task were to use 10 000 samples each. The logs showed after execution that in every run exactly 80 mapper tasks and 1 reduce task were executed. The results of the PiEstimator experiments are located in Figure 2.

4.4 WordCount

WordCount is a simple application that counts the number of occurrences of each word in a given input file. The role of the map function is to emit key/value pairs for each word found in the input file. The key/value pairs are sorted and aggregated locally before they are forwarded to the reducer tasks. The reduce function is responsible for counting all occurrences and writing the result to the distributed file system.

In the WordCount experiments we executed 130 mapper tasks and 67 reducer tasks for the Cloudera configuration and our own configuration. For the default configuration we specified 257 mapper tasks and 35 reducer tasks. We used a 16GB input file for each job. The results of the WordCount experiments are located in Figure 1.

5. DISCUSSION

The applications chosen for the experiments depend on different resources. The WordCount application is I/O bound, and is very CPU light because there are no expensive calculations being executed. This means that higher average

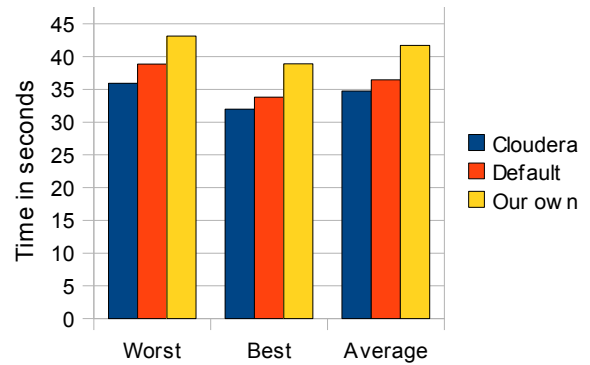


Figure 2: The PiEstimator results. The figure shows the worst, the best and the average times for Cloudera’s configuration, default configuration, and our own configuration.

reading speed will make the job run faster. The optimal execution would occur when each machine read data at maximum speed simultaneously throughout the whole job. The network is also an important resource for the WordCount application since large data sets are transferred between different tasks.

PiEstimator is unlike WordCount a CPU bound application. This means that the CPU speed decides how fast the job will be executed.

5.1 Results

The results of the experiments gave us a clear indication that the Hadoop framework should be adapted for the cluster it is running on. The results measured is the time it took to run the jobs. We will start discussing the results we got with the different configurations on the WordCount application.

The worst configuration of those tested on the WordCount application was the default configuration. The average time for this configuration was 710 seconds. Our own suggested configuration performed best, over 2,5 times faster than the default configuration, and 1,7 times faster than Cloudera’s configuration.

The configuration that performed worst on PiEstimator was our own. In the experiments we conducted, Cloudera’s configuration made the job run fastest. It was 1,2 times faster than our own configuration.

Understanding why our own configuration outperforms both other configurations in the WordCount application and not that well in PiEstimator we need to take a closer look at the configuration parameters.

- Maximum number of parallel mapper tasks per machine are set to different values in every configuration. The default configuration is set to execute maximum 2 mapper tasks in parallel while Cloudera’s configuration is set to 6. Our own configuration is set to 10.
- Maximum number of parallel reducer tasks per machine are set differently as well. The default configuration is allowed to execute 2 reducer tasks in parallel while Cloudera’s configuration and our own is set to 4.
- The number of threads at the job tracker for handling

incoming RPC calls from task trackers are set differently on the configurations. The default configuration has 10 threads running while Cloudera's configuration has 5 and our own uses 15 threads.

- The block size used in the configurations are not the same. The default configuration uses 64MB blocks while Cloudera's configuration uses 128MB, we used 128MB in our own configuration as well.
- Speculative execution for the reducer task was disabled in Cloudera's configuration while it was enabled in the other two.
- Cloudera's configuration and our own uses the fair scheduler while the default configuration does not specify one.

Both Cloudera's configuration and our own are faster than the default configuration on the WordCount application. One reason for this is the different block sizes. Cloudera's configuration and our own uses 128MB as the block size while the default configuration uses 64MB blocks. This means that a mapper task can run through 128MB data by only opening one block. In the default configuration two mapper tasks must be run to parse through the same amount of data. This is a drawback because initializing one more mapper task and opening one more file takes time. The difference in parallelism on each machine is something that also have an impact on the results. The default configuration can only run 2 mapper tasks in parallel while Cloudera's configuration runs 6 and our own runs 10. By running more mapper and reducer tasks in parallel we clearly notice a performance gain.

Our own configuration had the worst results in the PiEstimator experiments. One reason that could have caused bad performance was the high number of parallel mapper tasks per machine. Running 10 mapper tasks simultaneously on our 8 core machines may have caused a lot of context switching and actually reduced the performance of the running job.

6. RELATED WORK

The Cloudera company's business model is to help enterprises that needs Hadoop to process through their large data sets. They help the enterprises install and configure Hadoop. The Cloudera Configurator is a tool that adapts the Hadoop framework for a unique cluster.

Interesting work on optimizing the Hadoop task scheduler have been done by [6]. In their work they introduce a task scheduling algorithm, Longest Approximate Time to End (LATE). Their scheduling algorithm is designed to fit in heterogeneous environments. In heterogeneous clusters stragglers often occur. A straggler is a machine that performs poorly. The job of the scheduling algorithm is to find out when to run backup jobs, and not assigning that many jobs on slow machines. The results of their algorithm has proven to be good. In nearly all experiments the LATE scheduler outperformed the native Hadoop scheduler.

7. CONCLUSION

Adapting the Hadoop framework's configuration based on a cluster's hardware and number of nodes has proven to give increased performance. We have compared three different

configurations, the default, Cloudera's, and our own. We have seen that it is possible to get over 2 times greater performance by simply doing some small adjustments in the configuration files of Hadoop.

For future work we would like to see a configurator built in to Hadoop. This configurator should be able to extract hardware information from the cluster and automatically generate and deploy the configurations in the cluster. We would also like to see configuration adjustments happening while jobs are running, or configuration generated based on what resources an application needs the most.

8. REFERENCES

- [1] Cloudera homepage. <http://www.cloudera.com/>.
- [2] How many maps and reduces. <http://wiki.apache.org/hadoop/HowManyMapsAndReduces>.
- [3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
- [4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.
- [5] T. White. Hadoop distributed file system: Reliability. <http://tinyurl.com/9wgrg4>.
- [6] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. Technical Report UCB/EECS-2008-99, EECS Department, University of California, Berkeley, Aug 2008.