Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

Predicting Hadoop Parameters

Ziad Benslimane, Qin Liu
Software Engineering Department
Tongji University
Shanghai, P.R. China

Zhu Hongming
Software Engineering Department
Tongji University
Shanghai, P.R. China

***Abstract*—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 so analyzing the input in parallel is 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 predict some configuration parameters from the application's CPU utilization in order to optimize Hadoop.**

***Keywords*: Hadoop, Cloud Computing, Distributed Systems**

I. Terminology

A. *NameNode*

The NameNode is a unique HDFS node responsible for the file system namespace holding the metadata of the files stored in DataNodes; this node usually has the most expensive hardware available to avoid single point of failure [1].

B. *DataNode*

The DataNode is where the blocks are actually stored; it is usually made of commodity hardware since replication is offered by the software layer to the multiple instances running in the cluster [1].

C. *JobTracker*

The JobTracker is the node managing the MapReduce jobs by scheduling MapReduce tasks on the TaskTrackers and monitoring them [1].

D. *TaskTracker*

The TaskTracker node runs the Java Virtual Machine to execute the Map or Reduce tasks launched by the JobTracker. Multiple TaskTrackers must exist in a cluster in order to achieve parallelism [1].

II. Introduction

The decreasing price of disk storage and the increasing amount of generated data have introduced a problematic situation that data analytic has to cope with; the issue here is that the input can exceed any single hard disk and the application might be too heavy for a single CPU. The good news is that such data analytic has become possible due to the introduction of parallel computing and data distribution frameworks such as Hadoop.

Hadoop is an open source framework that allows distributed processing for huge amounts of data over a cluster and its architecture is made of two components: the Hadoop Distributed File System (HDFS) and the MapReduce framework. The first component, HDFS, runs on top of the existing file system on each node and it has a default block size of 64 MB; while the second component relies on the functional programming functions Map and Reduce. Based on this architecture, Hadoop is made of HDFS nodes and MapReduce ones; the first ones include the NameNode and multiple DataNodes, while the second category has one JobTracker and multiple TaskTrackers.

Hadoop has many advantages, including but not limited to the following [2]:

- Performing large scale computations
- Handling an input much larger than any single hard drive
- Managing failure and data congestion

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

- Using a simplified programming model
- Allowing automatic distribution of data
- Promoting very good scalability
- Keeping up with Moore's law

Hadoop splits data into chunks and then spreads them across the nodes with some required tasks to be done over them. It does that by breaking the problem into “Map"' and “Reduce” steps where the map phase splits the input and then passes sub-problems to slave nodes from which it expects an answer back and the reduce phase collects them back in order to form the answer for the original problem [3].

The main research now is about how to optimize Hadoop to get faster execution times given that every application is different in terms of resource consumption; so it is very important to do load balancing depending on the application's bottlenecks.

III. Literature Review

The current research is mainly conducted by Apache which is offering three different sub-projects related to Hadoop: Common, HDFS, and MapReduce. The "MapReduce" subproject is the most interesting for this chosen research area and it focuses on the programming model that is based on the two steps defined before: Map and Reduce [4].

Another good source of information of Hadoop is Cloudera, which gives training based on online classes, tutorials, as well as utilities to better understand the implementation details of Hadoop [5].

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” [1], 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.

Moreover, another paper titled: “Towards Optimizing Hadoop Provisioning in the Cloud” [6], 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. .

IV. Motivation

Hadoop comes with a built-in configuration that is not supposed to always give the optimal results regardless of cluster infrastructure, application type, input size, or network topology. In other words, the default parameters generated when installing Hadoop have to be tuned and optimized in order to give good results depending on the user's needs; this process however is known to be very tedious and time consuming as the user usually has to do many experiments in order to choose a good set of parameters.

The solution presented in this paper will allow a user to predict the best number of map tasks spawned as well as how many of them should be run in parallel at any given time; and that prediction will be based on the CPU statistics revealed by running the application on a single node cluster with a smaller input size in case the original one is too large to fit on a single disk.

Theoretically, after running an application on a single node cluster, it is very easy to determine the exact amount of time spent by the CPU running that specific job; so taking that into consideration, an application can be categorized by either being CPU-intensive, or CPU-light. Based on the application's category, a user can first decide on the number of map tasks to be scheduled and then on the ones to run in parallel on each node.

Predicting the number of map tasks is based on the idea that a single map task for a CPU intensive application is supposed to consume a decent amount of time, so the time required to generate an extra task would be kind of negligible; however, for a light CPU application, a map task setup overhead can be influential thus has to be minimized. On the other hand, predicting the number of parallel maps is related to the fact that the CPUs have to be kept as busy as possible; so CPU-light parallel maps have to be increased while CPU-heavy concurrent maps have to be decreased.

In order to successfully carry those experiments, and knowing that Hadoop automatically decides the number of map tasks based on the input splits, the input files were either merged or divided in order to force Hadoop to pick the exact number of Map tasks required for any given test.

V. Experiments

In this paper, the following three applications were run in a single node cluster to see their

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

CPU usage time: the PiEstimator, Grep, as well as MRIF (Map Reduce Integer Factorization). After that, they were categorized depending on their CPU usage either as CPU light or CPU intensive. Next, taking the category in consideration, the number of map tasks and the number of parallel maps were predicted and the configuration files were modified accordingly. Finally, the applications were executed using the new modified configuration files as well as the default Hadoop ones and the results were then compared and presented under the "Results" section.

A. Test Cases

PiEstimator

The PiEstimator is one of the map-reduce applications offered by Hadoop in its default installation directory; the application's mapper tries to estimate π by generating a given amount of random numbers and checking whether they are inside or outside the circle.

This application takes 2 input arguments: the number of map tasks to launch as well as the number of samples to check for [7]

Grep

The Grep map-reduce application 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 [8].

Mrif

The Map Reduce Integer Factorization application is an implementation of the Quadratic Sieve Algorithm that does integer factorization; each mapper is responsible for performing a sieve on an interval and returning the smooth factors; on the other hand, the reducer tries to find a subset of them whose product is a square [9]. This application takes one input, which is the integer to be factorized, then it generates one single file that Hadoop uses as an input split so only 1 map task gets launched.

B. First Phase: Single Node Cluster

The experiments in this phase were run on the same computer that had a dual core CPU running at a clock rate of 1.73 GHz. The PiEstimator, which takes the number of map tasks as input as well as the number of samples, was assigned 1 map and 1000 samples; On the other hand, Hadoop launched one map task for MRIF as it generated only one input file out of the integer given: 59595959595959595959; moreover, Grep was assigned to look for the string "so" and got 12 map tasks from hadoop for having 12 input text books.

The CPU usage time was revealed by the output of the MapReduce framework, and the results are presented in Figure 1.

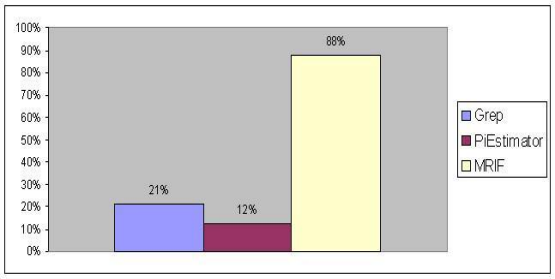

Figure 1: CPU Utilization

C. Second Phase: Multi Node Cluster

The aim of this phase is to prove that light CPU applications need high parallelism to keep a CPU busy while heavy CPU applications don't need that much; Moreover, the goal was to show that the number of tasks should be minimized for light-CPU applications due to the task setup overhead, while they should be maximized for CPU-heavy applications to distribute the load evenly across CPUs.

The experiments of this phase were conducted on a cluster of three nodes, each with a dual core CPU running at 1.7 GHz. One master node where all the commands were run had the namenode, the secondary namenode, the jobtracker, a tasktracker, and a datanode; whereas each of the other two nodes had a datanode and a tasktracker.

All the applications were executed with a changing number of map tasks as well as a changing amount of parallelism; this was done to find the optimum number of map tasks and map parallelism for the light CPU applications as well as the heavy CPU ones.

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

The amount of parallelism was changed by modifying the "mapred-site.xml" file inside the "conf" directory; the parameter field called "mapred.map.parallel" was edited to either disable parallelism by assigning a zero, or by allowing full parallelism thus assigning 4 map tasks given the number of tasks to test with was 12.

As far as the PiEstimator application is concerned, it was always run with 1000 samples and the number of map tasks was given through the command line. However, the Grep application was given the following string to look for "so", and the "cat" command had to be used in order to merge the 12 input splits thus forcing Hadoop to change the number of maps. Unlike the previous applications, the MRIF application was CPU intensive, and it was executed to factorize the following integer: 59595959595959595959; the number of map tasks was influenced by using the "split" command which splits the input file into many partitions.

VI. Results and Discussion

A. *First Phase*

The aim of this phase was to categorize applications based on their CPU utilization, so MRIF was considered to be CPU-intensive while PiEstimator and Grep were both taken to be CPU-light as Figure 1.

The percentages were calculated by dividing the field of "CPU time spent" by the total execution time; as far as the PiEstimator is concerned, the CPU ran for 4000 ms while the total execution time was 32100 ms. In a similar fashion, the Grep application executed in 31 seconds while the CPU spent only 6510 ms. Moreover, the MRIF application run for almost 194 seconds and its CPU spent around 171 seconds.

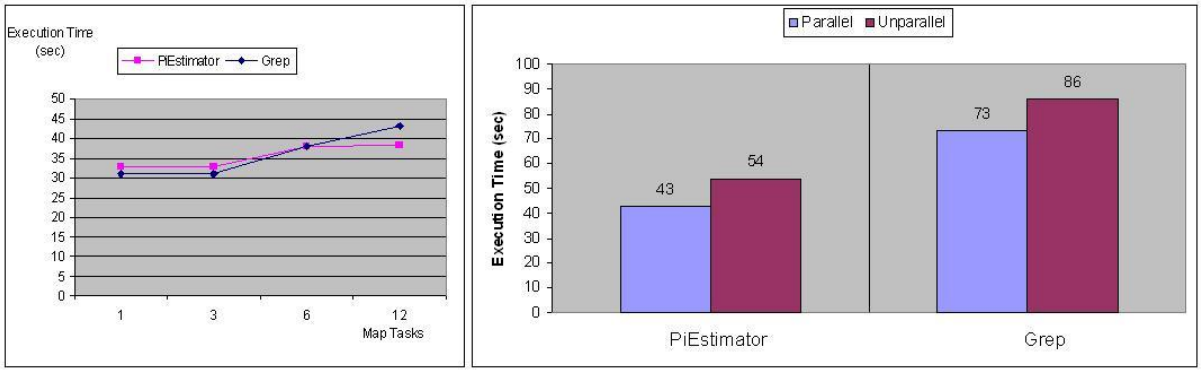

Figure 2: Execution Time for CPU-Light

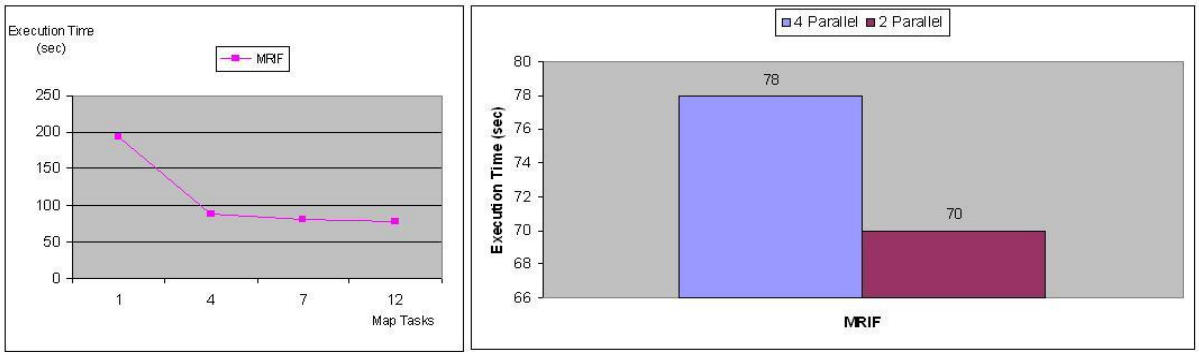

Figure 3: Execution Time for CPU-Heavy

ISBN: 978-981-07-5939-1

B. *Second Phase*

Figure 2 shows that increasing the number of map tasks for light CPU applications is not a good idea; this is due to the fact that the overhead of setting up a task is not negligible compared to the time required to finish executing a task. Both CPU light applications, PiEstimator and Grep, executed faster when given 1 or 3 map tasks rather than 6 or 12 ones. Figure 2 also shows that parallelism is good for those kinds of applications; it allows you to keep the CPU as busy as possible by executing many light tasks at the same time.

On the other hand, and as Figure 3 shows, increasing the number of map tasks for a heavy-CPU application is rather a good idea; this is related to the task setup time being negligible once compared with the huge time of executing a heavy task. So it is always better to run multiple tasks in order to distribute the balanced load across all the CPUs. Moreover, decreasing parallelism for heavy tasks is very beneficial to avoid abusing the CPUs; in other words, each CPU is already busy enough with one heavy task and should not be allowed to run many ones in parallel.

VII. Future Work

The current paper 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 predicting also the number of reduce tasks. Moreover, there is a good chance of including "Ganglia" [10], which is a famous cluster monitoring tool that might display more accurate results concerning the cluster performance. If time permits, the "Cloudera" [11] configuration tool will be incorporated in order to compare their configuration parameters to the ones predicted in this paper.

VIII. Conclusion

Even though Hadoop contributes heavily to data analytic; it needs to be optimized for each specific application and cluster. The solution discussed in this paper takes advantage of the application's CPU statistics in order to predict the number of map tasks as well as the number of parallel ones; this optimization reduces the overall execution time by applying a good load balance and by avoiding any possible CPU bottlenecks.

REFERENCES

[1] Christer Hansen. Optimizing hadoop for the cluster. Technical report, Institute for Computer Science, University of Tromsa.

[2] Zak Stone. Introduction to hadoop. Technical report, Harvard School of Engineering and Applied Sciences.

[3] Wikipedia. http://en.wikipedia.org/wiki/MapReduce, 2012. [Online; accessed 4-January-2012].

[4] The Apache Software Foundation. http://hadoop.apache.org/mapreduce/, 2011.[Online; accessed 11-November-2011].

[5] Inc. Cloudera. http://www.cloudera.com/hadoop-training/, 2011. [Online; accessed 5-November-2011].

[6] Kambatla Karthik. Towards Optimizing Hadoop Provisioning in the Cloud. Technical report, Purdue University, IBM Research Almaden.

[7] Apache. http://hadoop.apache.org/common/docs/r0.20.2/api/org/apache/hadoop/examples/PiEstimator.html,2009. [Online; accessed 17-February-2012].

[8] Hadoop Wiki. Grep Example. http://wiki.apache.org/hadoop/Grep, 2009.[Online; accessed 13-February-2012].

[9] Tordable Javier. Map Reduce for Integer Factorization. http://code.google.com/p/mapreduce-integer-factorization/, 2009. [Online; accessed 19-March-2012].

[10] Ganglia Monitoring System. http://ganglia.sourceforge.net/. [Online; accessed 25-March-2012].

[11] Inc. Cloudera. http://www.cloudera.com/, 2012. [Online; accessed 13-March-2012].