Performance analysis of MapReduce Programs on Hadoop cluster

Mahesh Maurya
Research Scholar, JJT University
MPSTME, Mumbai, India
maheshkmaurya@yahoo.co.in

Sunita Mahajan
MET Principal, Mumbai
MET, Mumbai, India
sunitam_ics@met.edu

Abstract: **This paper discusses various MapReduce applications like pi, wordcount, grep, Terasort. We have shown experimental results of these applications on a Hadoop cluster. In this paper, performance of above application has been shown with respect to execution time and number of nodes. We find that as the number of nodes increases the execution time decreases. This paper is basically a research study of above MapReduce applications.**

Keywords- Hadoop, MapReduce, HDFS, wordcount, grep)

1. Introduction:

Apache Hadoop is a software framework that supports data-intensive distributed applications. It enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop was derived from Google's MapReduce [2] and Google File System (GFS) papers [3].

HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework. Each node in a Hadoop instance typically has a single data node; a cluster of data nodes form the HDFS cluster. The situation is typical because each node does not require a data node to be present. Each data node serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses the TCP/IP layer for communication; clients use RPC to communicate between each other. HDFS stores large files (an ideal file size is a multiple of 64 MB), across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts [1]. MapReduce is a programming model for processing large data sets, and the name of an implementation of the model by Google. MapReduce is typically used to do distributed computing on clusters of computers [1].

This paper covers many sections. Section II covers about Hadoop File System like what is Hadoop file system, its architecture and its communications. Section III discusses about MapReduce like what is MapReduce, its architecture and communications. Section IV and V shows the experimental setup, results and observations. Section VI and VII are conclusion and references.

2. Hadoop file system (HDFS) [5]

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. [5] Hadoop [4][6][7] provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce [2] paradigm.

2.1 ARCHITECTURE [5]

A. NameNode

The HDFS namespace is a hierarchy of files and directories. Files and directories are represented on the NameNode by inodes, which record attributes like permissions, modification and access times, namespace and disk space quotas.

B. DataNodes

Each block replica on a DataNode is represented by two files in the local host's native file system. The NameNode does not directly call DataNodes. It uses replies to heartbeats to send instructions to

978-1-4673-4805-8/12/$31.00 ©2012 IEEE

the DataNodes. The instructions include commands to:

1. Replicate blocks to other nodes;
2. Remove local block replicas;
3. Re-register or to shut down the node;
4. Send an immediate block report.

C. HDFS Client

User applications access the file system using the HDFS client, a code library that exports the DFS file system interface. Similar to most conventional file systems, HDFS supports operations to read, write and delete files, and operations to create and delete directories. The user references files and directories by paths in the namespace.

D. Image and Journal [5]

The namespace image is the file system metadata that describes the organization of application data as directories and files. A persistent record of the image written to disk is called a checkpoint.

E. CheckpointNode

The CheckpointNode periodically combines the existing checkpoint and journal to create a new checkpoint and an empty journal. The CheckpointNode usually runs on a different host from the NameNode since it has the same memory requirements as the NameNode

F. BackupNode

The BackupNode is capable of creating periodic checkpoints, but in addition it maintains an nmemory, up-to-date image of the file system namespace that is always synchronized with the state of the NameNode

G. Upgrades, File System Snapshots

The purpose of creating snapshots in HDFS is to minimize potential damage to the data stored in the system during upgrades. The snapshot mechanism lets administrators persistently save the current state of the file system, so that if the upgrade results in data loss or corruption it is possible to rollback the upgrade and return HDFS to the namespace and storage state as they were at the time of the snapshot.

3. MapReduce programming: [2] [8]

MapReduce [2] [8] is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

3.1 Programming Model [2] [8]

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values.

3.2 Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents. The user would write code similar to the following pseudo-code: [2] [8]

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

The map function emits each word plus an associated count of occurrences (just ‘1’ in this simple example). The reduce function sums together all counts emitted for a particular word.

3.3 Types

Even though the previous pseudo-code is written in terms of string inputs and outputs, conceptually the map and reduce functions supplied by the user have associated types: [2] [8]

map (k1,v1) ! list(k2,v2)

reduce (k2,list(v2)) ! list(v2)

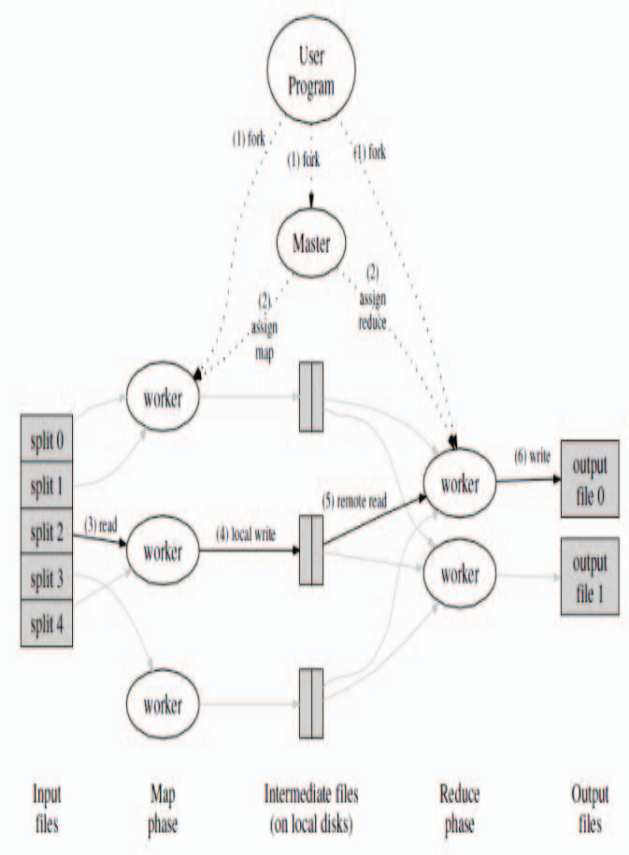

Fig.1 Overview of MapReduce Execution [2] [8]An excellent style manual for science writers is [7].

4. Experimental setup:

The experiments were carried out on the Hadoop cluster.. The hadoop infrastructure consists of one cluster having four nodes geographically distributed in one single lab. For our series of experiments we used the nodes in the Hadoop cluster, with Intel Core 2 @ 93GHZ 4 CPUs and 4 GB of RAM for each node. With a measured bandwidth for end-to-end TCP sockets of 100 MB/s, Ubuntu 11.10 and SUN JAVA jdk 1.6.0.
Experiments with MapReduce applications:
4.1 Pi: [1]
π is commonly defined as the ratio of a circle's circumference C to its diameter d:

$$\pi = \frac{C}{d}. \quad (1)$$

The ratio C/d is constant, regardless of the circle's size. For example, if a circle has twice the diameter d of another circle it will also have twice the circumference C, preserving the ratio C/d.

4.2 Wordcount: [2] [8]

Wordcount example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab.

To run the example, the command syntax is

bin/hadoop jar hadoop-*-examples.jar wordcount [-m <#maps>] [-r <#reducers>] <in-dir> <out-dir>

bin/hadoop dfs -mkdir <hdfs-dir>

bin/hadoop dfs -copyFromLocal <local-dir> <hdfs-dir>

As of version 0.17.2.1, you only need to run a command like this:

bin/hadoop dfs -copyFromLocal <local-dir> <hdfs-dir>

4.3 Grep: [2]
The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
4.4 Terasort: [1] [4]
This package consists of 3 map/reduce applications for Hadoop to compete in the annual terabyte competition.

- TeraGen is a map/reduce program to generate the data.
- TeraSort samples the input data and uses map/reduce to sort the data into a total order.
- TeraValidate is a map/reduce program that validates the output is sorted.

TeraSort is a standard map/reduce sort, except for a custom partitioner that uses a sorted list of N-1 sampled keys that define the key range for each reduce. In particular, all keys such that sample [i-1] <= key < sample[i] are sent to reduce i. TeraValidate ensures that the output is globally sorted. It creates one map per a file in the output

directory and each map ensures that each key is less than or equal to the previous one..

5. Results and observations

5.1 Pi :

Sr. No.	Time spent in execution (ms)	No. of Nodes
1	678.18	2
2	127.29	3
3	93.88	4

Table 1: Observations of pi MapReduce application

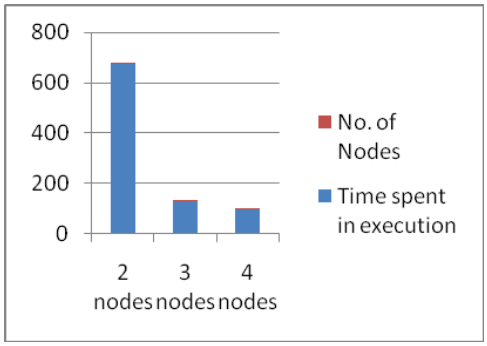

Fig 2: Results of pi MapReduce applications

5.2 Wordcount:

Sr. No.	Time spent in Execution(ms)	Average time (ms)	No. of Nodes
1	4910	6424	2
	7470		
	6690		
	6490		
	6560		
2	6760	5970	3
	6400		
	4880		
	5760		
	6050		
3	6340	5876	4
	4750		
	6180		
	6080		
	6030		

Table 2: Observations of wordcount MapReduce application

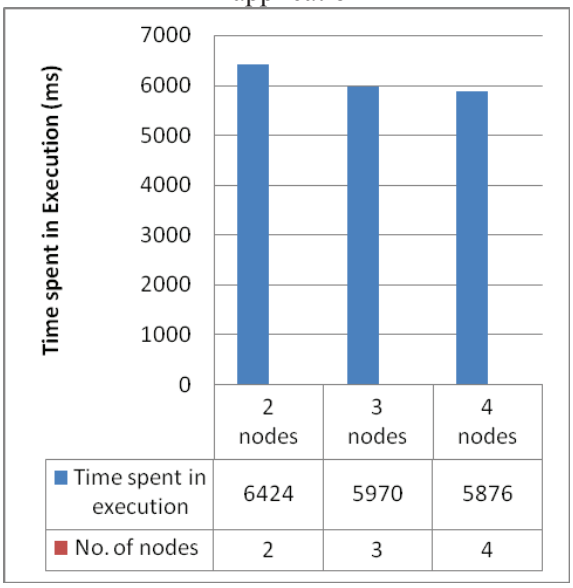

Fig.3: Results of wordcount MapReduce application

5.3 Grep:

Sr. No.	Time Spent in Execution Job 1 (ms)	Average of Job1	Time Spent in Execution Job 2 (ms)	Average of job2	No. of nodes
1	5260	5200	1630	1902	2
	5430		2310		
	5340		2140		

	4770		1530		
2	6100	4672	1080	1572	3
	3590		1750		
	5170		1650		
	3830		1810		
3	5020	4517	1110	1455	4
	4060		1880		
	4540		1300		
	4450		1530		

Table 3: Observations of grep MapReduce application

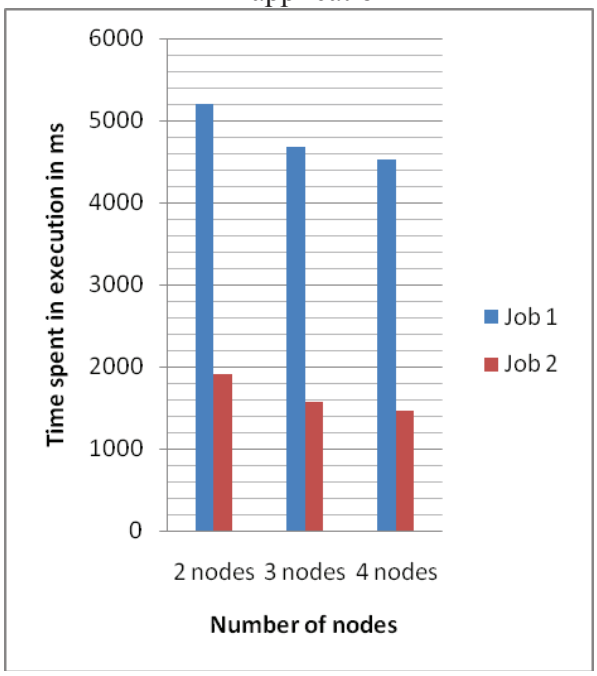

Fig.4: Results of grep MapReduce application

5.4 Terasort:

Sr. No.	Write	Read	No. of nodes
1	225683	152267	2
2	186348	127589	3
3	155840	74460	4

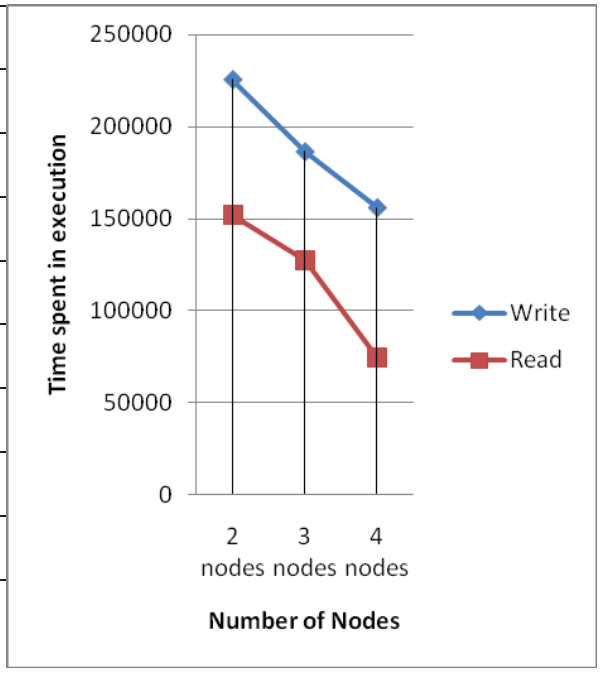

Fig.5: Results of terasort MapReduce applications

6. Conclusion:

In this paper, we have seen those MapReduce applications' observations and results. Observations show that as the number of nodes increases in cluster time spent in execution decreases. In this way it shows that MapReduce applications results totally depend on the size of Hadoop cluster. This Hadoop cluster contains four nodes i.e. one master and three slaves.

7. References:

[1] http://en.wikipedia.org/wiki/Apache_Hadoop

[2] Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing On Large Clusters", Google, Inc., Usenix Association OSDI '04: 6th Symposium on Operating Systems Design and Implementation, 2009

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung "The Google File System", Google, Sosp'03, October 19–22, 2003, Bolton Landing, New York, USA. Copyright 2003 ACM 1-58113-757-5/03/0010, 2003

[4] Apache Hadoop. http://hadoop.apache.org

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler "The Hadoop distributed File System", Yahoo! Sunnyvale, California USA, 978-1-4244-7153-9/10/$26.00 ©2010 IEEE, 2010.

[6] J. Venner, Pro Hadoop. Apress, June 22, 2009.

[7] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Yahoo! Press, June 5, 2009.

[8] Ralf Lammel "Google's Mapreduce Programming Model—Revisited", Data programmability Team Microsoft Corp. Redmond, WA, USA, This Paper Has Been Published In SCP, Http://En.Wikipedia.Org/Wiki/Mapreduce, 2008.