Implementation of Distributed Searching and Sorting using Hadoop MapReduce

Sanjeev Pippal
sanpippalin@gmail.com

Ankur Shukla
ankur@gmail.com

Dharmender Singh Kushwaha
dsk@mnnit.ac.in

Department of Computer Science & Engineering
MNNIT Allahabad-211004

ABSTRACT

This paper focuses on implementation of MapReduce programming model on Hadoop cluster for parallel processing of huge amount of data efficiently. There is deluge of data everywhere and we need to process these data efficiently to take decisions and to reach at some meaningful conclusion so that the information can be made useful. Processing large data set is a big problem as most of the data in unstructured. So we can't apply any algorithms since efficiency is a issue. In that case we need to process complete data set for almost every query on the data to get the desired result. If we process this huge amount of data on a single machine in linear fashion then we face problems like data storage issues and response time. So we need a mechanism which can process huge data sets in parallel fashion on distributed machines. MapReduce programming model is one such widely accepted model for parallel processing of data. While executing queries on the file containing 10 million records, it is observed that as the number of data nodes in the Hadoop cluster is increased, the minimum performance improvement obtained is over 42%. When implementing Distributed Grep Search vs Normal Implementation of Grep for a input file that has 10 million records containing user name, operation, ip address, with 3 data nodes, an improvement of over 18% is observed. For word count application, a substantial improvement of over 200% is observed with 3 or more data nodes. With our input size secondary sort outperforms insertion sort and bubble sort algorithms.

Keywords: MapReduce, Hadoop, Hadoop Distributed File System, Grep

1. INTRODUCTION

Now a day's big data is generally processed in data centers located at distributed sites. For data processing, various kinds of data processing paradigm are used. Among them MapReduce is most common data intensive computing paradigm. Hadoop is an open source implementation of MapReduce paradigm, which is used to store and process big data. Performing large-scale computation is difficult. To work with this volume of data requires distributing parts of the problem to multiple machines to handle in parallel. Whenever multiple machines are used in cooperation with one another, the probability of failures rises.

ICTCS '14, November 14 - 16 2014, Udaipur, Rajasthan, India
Copyright 2014 ACM 978-1-4503-3216-3/14/11…$15.00
http://dx.doi.org/10.1145/2677855.2677932

In a distributed environment, however, partial failures are an expected and common occurrence. Networks can experience partial or total failure if switches and routers break down. Multiple implementations or versions of client software may speak slightly different protocols from one another. Clocks may become desynchronized, lock files may not be released, parties involved in distributed atomic transactions may lose their network connections part-way through, etc. In each of these cases, the rest of the distributed system should be able to recover from the component failure or transient error condition and continue to make progress. The computations which are required to be performed over big data are conceptually straightforward. But the size of input data is typically very large. Furthermore these computations have to be distributed across hundreds or thousands of machines in order to finish the computation in a reasonable amount of time. So the issues of big data processing can be defined as:

- How to parallelize the computation.
- How to distribute the data.
- How to handle failures.

In order to solve these issues we can use functional model with user specified map and reduce operations which allows us to parallelize large computations on various distributed computing nodes. Furthermore we can use re-execution policy as primary mechanism for fault tolerance.

Our objective is to set up a hadoop cluster in fully distributed mode having one name node and two data nodes. The name node will also work as secondary name node. After cluster setup we need to write mapper and reducer functions of respective applications that will operate on the large data sets. The mapper and reducer functions are inspired by functional programming model. These functions are implemented in java and will be scheduled to run on our hadoop cluster.

2. RELATED TECHNOLOGIES

2.1 Cloud Computing

Cloud computing involves a large number of computer systems connected to a real time communication network such as Internet. Cloud computing relies on sharing computing resources rather than having local servers or personal devices to handle applications. Conceptually, users acquire computing platforms or IT infrastructures, from Clouds and execute their applications inside them. The use of multi-tenant [1,18] database designs adds to the efficiency of clouds in terms of space usage. Cloud can also be formed through volunteer ad hoc resources [2] and they are called ad hoc clouds.

2.2 Big Data

Big data [3] can be defined as a collection of data sets which is so large and complex and also exceeds the processing capacity of conventional database management systems. The data is too big and the movement is too fast or does not fit the structures of our current database architectures. Big Data is typically large volume of unstructured and structured data that gets created from various organized and unorganized applications, activities and channels such as emails, tweeter, web logs, Facebook etc.

The main difficulties with Big Data include capture, storage, search, sharing, analysis, and visualization and security[4]. The core of Big Data is Hadoop which is a platform for distributing computing problems across a number of servers. Hadoop uses HDFS as its underlying file system to store data.

2.3 MapReduce

MapReduce [5] is a data processing or parallel programming model introduced by Google[8]. MapReduce model is inspired by two functions of functional programming: Map and Reduce. With the MapReduce programming model, programmers only need to specify two functions: Map and Reduce. The Map function receives a key/value pair as input and generates intermediate key/value pairs to be further processed. The Reduce function merges all the intermediate key/value pairs associated with the same intermediate key and then generates final output. Many real world applications can be expressed using MapReduce model. The most popular implementation of MapReduce is Hadoop which is widely used to process large amount of data on large clusters built from commodity hardware.

2.4 Hadoop

Hadoop [7, 14, 15, and 16] is an open source implementation of MapReduce programming model [9]. Hadoop provides a reliable shared storage and analysis system. The storage is provided by Hadoop Distributed File System (HDFS)[10] and analysis by MapReduce. The Hadoop MapReduce framework contains a single master node and various slave nodes. Master node runs a Job tracker instance. This Job tracker instance accepts input from client node and splits the job into number of smaller tasks. Each slave node runs a Task tracker instance which accepts the tasks from master node.

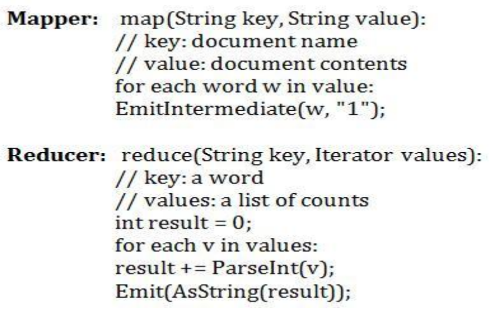
2.5 HDFS: Hadoop Distributed File System

Distributed file systems are the file systems that manage the storage across a network of machines. Hadoop comes with a distributed file system called HDFS. HDFS [3, 10] is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. HDFS cluster consists of a single Name Node and many Data Nodes. The Name Node maintains the file system tree and the meta data for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the name space image and the edit log. Name Node also determines the mapping of blocks to Data Nodes. Data Nodes are responsible for storing and retrieving blocks when they are told to and they report back to the Name Node periodically with list of blocks they are storing.

3. PROPOSED WORK

3.1 Counting Frequency of Words in a File

Word-frequency in this context means to count how often a distinct word appears in a collection of texts. Word-frequency statistics could be used to give a first measurement for the strength of expression of any literary text or to optimize text-based compression[11] algorithms. As we know that how many times a word appears in the document, we can compress the file using this information.
The Algorithm for frequency counting is as follows:

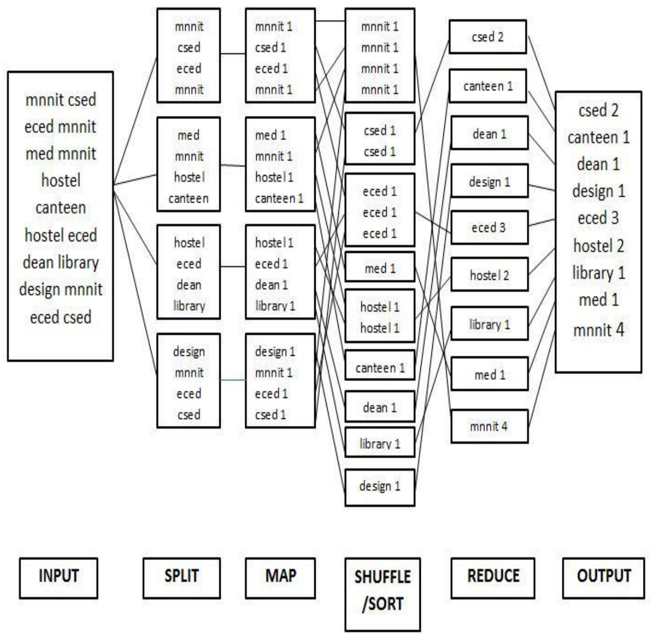


```
Mapper:   map(String key, String value):
          // key: document name
          // value: document contents
          for each word w in value:
          EmitIntermediate(w, "1");

Reducer:  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));
```

Figure 1: Word Frequency Counting Algorithm [1]

The input of the map phase is a list of text files. The whole content of each text file is represented by one large string. It is split into single words with the help of the words function of the standard library. It just uses white spaces as delimiter for the separation of distinct words. An example of overall execution of the algorithm can be depicted as follows:

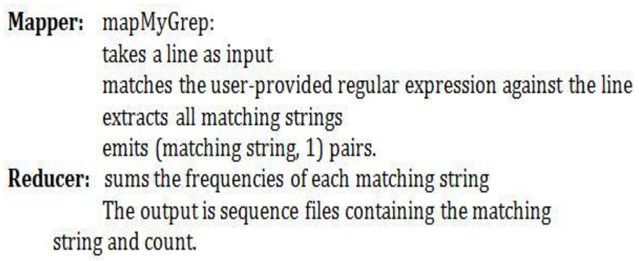


mnnit csed eced mnnit med mnnit hostel canteen hostel eced dean library design mnnit eced csed
mnnit csed eced mnnit
med mnnit hostel canteen
hostel eced dean library
design mnnit eced csed
mnnit 1 csed 1 eced 1 mnnit 1
med 1 mnnit 1 hostel 1 canteen 1
hostel 1 eced 1 dean 1 library 1
design 1 mnnit 1 eced 1 csed 1
mnnit 1 mnnit 1 mnnit 1 mnnit 1
csed 1 csed 1
eced 1 eced 1 eced 1
med 1
hostel 1 hostel 1
canteen 1
dean 1
library 1
design 1
csed 2
canteen 1
dean 1
design 1
eced 3
hostel 2
library 1
med 1
mnnit 4
csed 2 canteen 1 dean 1 design 1 eced 3 hostel 2 library 1 med 1 mnnit 4
INPUT
SPLIT
MAP
SHUFFLE /SORT
REDUCE
OUTPUT

Figure 2: Word Frequency Counting Execution Sequence Example

3.2 Distributed Grep Algorithm

Grep is a command-line utility for searching plain-text data sets for lines matching a regular expression. The distributed grep program has the same general functionality than the well-known command line program in Linux. It tests every line of the input files if it matches a given pattern which is defined by a regular expression. The input for the map phase is a tuple list whereas the first element is the filename while the second one represents the file content in a single string. The Distributed Grep Algorithm is shown in figure 3.



```
Mapper:   mapMyGrep:
          takes a line as input
          matches the user-provided regular expression against the line
          extracts all matching strings
          emits (matching string, 1) pairs.
Reducer:  sums the frequencies of each matching string
          The output is sequence files containing the matching
     string and count.
```

Figure 3: Distributed Grep Algorithm

The reduce phase just collects all lines and passes them to the user. Therefore the reduce function in the above listing does no data manipulation and only returns its input values. The efficiency of the distributed grep program is determined by the number of workers in the map phase. The overall time saving results from the use of parallel grep processes. For the reduce phase only a small number of workers or even only a single machine is needed.

4. SIMULATION RESULTS AND PERFO-RMANCE

4.1 Hadoop Multi Node Cluster

Name Node: The name node runs on port number 9000 which is configured in coresite.xml file. We can access the name node and get information using web UI by accessing the URL http://master:50060/. Name Node runs on following IP: 172.31.132.120.
Following processes run on Name Node:
• Name Node
• Job Tracker
• Secondary Name Node

Data Node: There are two data nodes in the cluster. We can access the task tracker modules of data nodes and get information using web UI by accessing the URL http://datanodehost:50060/. Name Node runs on following IP: 172.31.132.114 and 172.31.132.116. Following processes run on Data Node:
• Data Node
• Task Tracker

Task Tracker: Every data node in the cluster runs a separate java process named as Task Tracker. These task trackers get their job from job tracker which runs on Name Node. Job tracker splits the input file into number of input splits and distributes these splits to available task trackers for execution. After executing the respective job, task tracker reports back to job tracker.
Simulation of Word Frequency Counting Algorithm
Input: A text file having random collection of words.

Output: Returns a file that contains every distinct word in sorted order of the input file and a count which represents the number of occurrences of that particular word.
Simulation of Distributed Grep Algorithm
Input: A text file having 10000 twitter log records.

Output: The output is a file containing one single line which shows the number of occurrences of the regular expression along with regular expression.
Simulation of Secondary Sort Algorithm

Input: A text file having 5500000 IP addresses in dotted decimal format with decimals replaced with white spaces.
Output: Returns a file which contains the first two octets of the IP address in sorted order based on these two octets.

4.2 Analyzing Twitter Log file

Our applications can be used to analyze various real time data sets to get useful information from real time data. One such example of real time data is Twitter log file. The log file captures various information such as user names, login attempts, new tweets, ip addresses of users etc. We have taken one such log file on which our application will work.
The log file in our case has following information:
• User Name
• Operation (Login, new tweet, log out etc.)
• IP Address of the user

4.2.1 Performance of Queries on Log File:

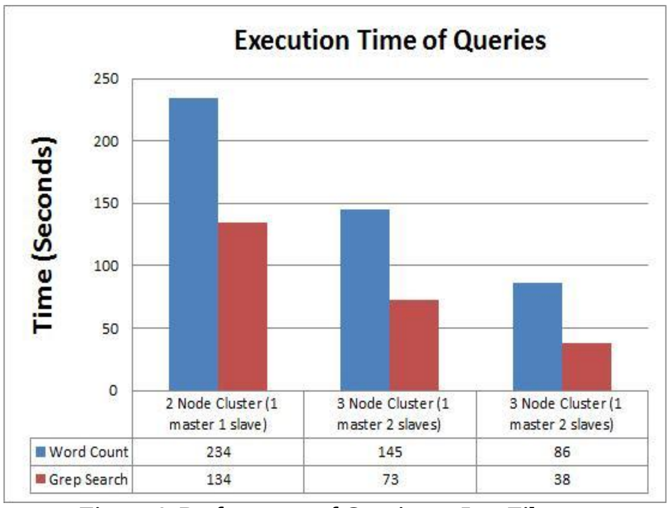
The queries are executed on the log file with different cluster configurations. The detail of query execution is as follows:
Queries: Word Count, Grep Search.

Input: The input file has 10 million records of following type:
user name, operation, ip address
The queries are executed on following three configurations:

• Configuration 1: Two node cluster, one name node and one data node.

• Configuration 2: Three node cluster, one name node and two data nodes.

• Configuration 2: Three node cluster, one name node and three data nodes.
The performance graphs of queries are is shown in figure 4.



	2 Node Cluster (1 master 1 slave)	3 Node Cluster (1 master 2 slaves)	3 Node Cluster (1 master 2 slaves)
🟦 Word Count	234	145	86
🟥 Grep Search	134	73	38

Figure 4: Performance of Queries on Log File

While executing queries on the file containing 10 million records, it is observed from figure 4 that as the number of data nodes in the Hadoop cluster are increased, the minimum performance improvement obtained is over 42%.

4.3 Performance Comparison of Distributed Grep Search with Normal Implementation on Log

File: Distributed Grep search query performs better than its normal implementation in the following scenario:

Queries to be compared: Distributed Grep Search vs Normal Implementation of Grep.

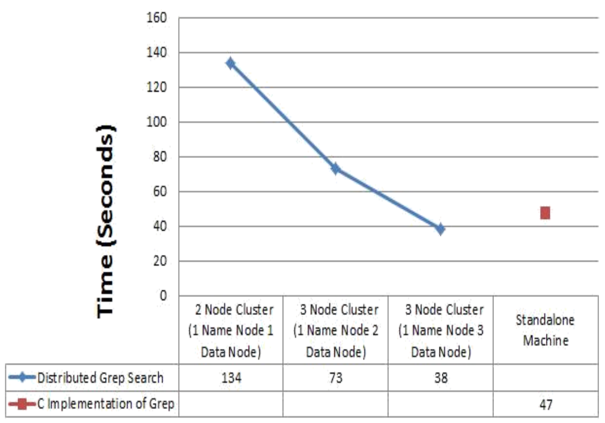
Input: The input file has 10 million records of following type:
user name, operation, ip address
The queries are executed on following three configurations:

• Configuration 1: Two node cluster, one name node and one data node.

• Configuration 2: Three node cluster, one name node and two data nodes.

• Configuration 2: Three node cluster, one name node and three data nodes.

The performance comparison graphs of queries are is shown in figure 5.



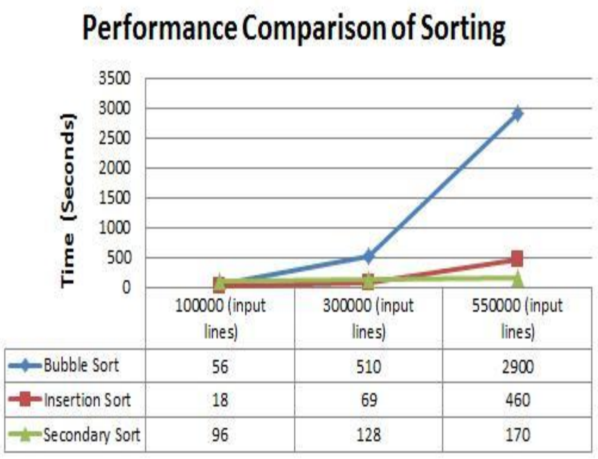
	2 Node Cluster (1 Name Node 1 Data Node)	3 Node Cluster (1 Name Node 2 Data Node)	3 Node Cluster (1 Name Node 3 Data Node)	Standalone Machine
Distributed Grep Search	134	73	38	
C Implementation of Grep				47

Figure 5: Performance Comparison of Grep Search

While comparing Distributed Grep Search with Normal Implementation on Log, it is seen that for large data size, increasing the number of data nodes in the Hadoop cluster, an improvement in response time of over 20 % is obtained as compared to the same execution on a standalone machine.

4.4 Performance Comparison of Secondary Sort with Insertion and Bubble Sort

Secondary sort algorithm works better for larger size input and more number of nodes. With our input size secondary sort outperforms insertion sort and bubble sort algorithms. Bubble sort is least efficient for larger input size. Insertion sort works better if the input is sorted up to some extent. Secondary sort does not depend on input type and performs well for larger input. Following graph in figure 6 depicts the performance comparison of these techniques for different input sizes.



	100000 (input lines)	300000 (input lines)	550000 (input lines)
Bubble Sort	56	510	2900
Insertion Sort	18	69	460
Secondary Sort	96	128	170

Figure 6: Performance Comparison of Secondary Sort

5. CONCLUSION

The overall intention of this presentation is to facilitate data intensive computing in a faster and efficient way. MapReduce paradigm is well known architecture for big data processing. When input data to be processed is very large then MapReduce model is very useful and efficient. MapReduce model effectively partitions the input data and distributes the data to various computing nodes. The computations are performed on the data in parallel manner. The processing speed depends on number of computing nodes and computing power of each node. So we can improve the processing speed by adding more nodes in existing cluster. Different nodes of the cluster get the part of input file to be processed. In this way various nodes work on different parts of input simultaneously, thereby reducing the overall execution time. While executing queries on the file containing 10 million records, it is observed that as the number of data nodes in the Hadoop cluster is increased, the minimum performance improvement obtained is over 42%. When implementing Distributed Grep Search vs Normal Implementation of Grep for a input file that has 10 million records containing user name, operation, ip address, with 3 data nodes, an improvement of over 18% is observed. For word count application, a substantial improvement of over 200% is observed with three or more data nodes. With our input size secondary sort outperforms insertion sort and bubble sort algorithms.

6. REFERENCES

[1] Pippal, S.; Sharma, V.; Mishra, S.; Kushwaha, D.S., "Secure and efficient multitenant database for an ad hoc cloud," Securing Services on the Cloud (IWSSC), 2011 1st International Workshop on , vol., no., pp.46,50, 6-8 Sept. 2011
doi: 10.1109/IWSSCloud.2011.6049024

[2] Sanjeev Pippal and Shakti Mishra and Dharmender Kushwaha " Architectural Design and Issues for Ad-Hoc Clouds", Third International conference on advances in communication, network and computing Springer doi={10.1007/978-3-642-35615-5_44}

[3] Rabi Prasad Padhy "Big Data Processing with Hadoop-MapReduce in Cloud Systems", International Journal of Cloud Computing and Services Science (IJCLOSER) Vol.2, No.1, February 2013, pp. 16 27 ISSN: 2089-3337

[4] Pippal, S.K.; Kumari, A.; Kushwaha, D.S., "CTES based secure approach for authentication and authorization of resource and service in clouds," Computer and Communication Technology (ICCCT), 2011 2nd International Conference on , vol., no., pp.444,449, 15-17 Sept. 2011
doi: 10.1109/ICCCT.2011.6075140

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler. "The Hadoop Distributed File System", In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10).

[6] Lizhe Wang, Jie Tao, Rajiv Ranjan , Holger Marten, Achim Streit , Jingying Chene, Dan Chena., "G-Hadoop: MapReduce across distributed data centers for dataintensive computing.", Future Generation Computer Systems 29 (2013): 739-750

[7] A Workload Model for MapReduce by Thomas A. de Ruiter, Master's Thesis in Computer Science, Parallel and Distributed Systems Group Faculty of Electrical Engineering, Mathematics, and Computer Science. Delft University of Technology, 2nd June 2012.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The google file system". SIGOPS Oper. Syst. Rev., 37(5):29-43, 2003.

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

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: a distributed storage system for structured data." In OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, pages 15-15, Berkeley, CA, USA, 2006. USENIX Association.

[11] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. "Pig latin: a notsoforeign language for data

processing." In SIGMOD '08: Proceedings of the 2008 ACMSIGMOD international conference on Management of data, pages 1099-1110, New York, NY, USA, 2008. ACM.

[12] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. "A comparison of approaches to large-scale data analysis". In SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of

[13] data, pages 165-178, New York, NY, USA, 2009. ACM.

[14] Apache hadoop. Website. http://hadoop.apache.org.

[15] Apache hadoop applications. http://wiki.apache.org/hadoop/PoweredBy.

[16] Apache mahout. http://lucene.apache.org/mahout/.

[17] Test data. http://www.manning.com/holmes

[18] Sanjeev Kumar Pippal, and Dharmender Singh Kushwaha. "A Simple, Adaptable and Efficient Heterogeneous Multi-Tenant Database Architecture for Ad Hoc Cloud." Journal of Cloud Computing: Advances, Systems and Applications 2013. Vol 2 no 5