International Journal of Cloud Computing and Services Science (IJ-CLOSER)
Vol.2, No.1, February 2013, pp. 16~27
ISSN: 2089-3337

Big Data Processing with Hadoop-MapReduce in Cloud Systems

Rabi Prasad Padhy
Senior Software Engg, Oracle Corp.,
Bangalore, Karnataka, India

Article Info

Article history:

Received Oct 10th, 2012
Accepted Oct 31th, 2012

Keyword:

Big Data
Hadoop
MapReduce
Indexing
Dataset storage

ABSTRACT

Today, we're surrounded by data like oxygen. The exponential growth of data first presented challenges to cutting-edge businesses such as Google, Yahoo, Amazon, Microsoft, Facebook, Twitter etc. Data volumes to be processed by cloud applications are growing much faster than computing power. This growth demands new strategies for processing and analyzing information. Hadoop-MapReduce has become a powerful Computation Model addresses to these problems. Hadoop HDFS became more popular amongst all the Big Data tools as it is open source with flexible scalability, less total cost of ownership & allows data stores of any form without the need to have data types or schemas defined. Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. In this paper I have provided an overview, architecture and components of Hadoop, HCFS (Hadoop Cluster File System) and MapReduce programming model, its various applications and implementations in Cloud Environments.

Corresponding Author:

Rabi Prasad Padhy,
Senior Software Engineer,
Oracle Corporation, Bangalore
Karnataka , India.
Email: rabi.padhy@gmail.com

1. INTRODUCTION

Cloud computing has been driven fundamentally by the need to process an exploding quantity of data in terms of exabytes as we are approaching the Zetta Byte Era. One critical trend shines through the cloud is Big Data. Indeed, it's the core driver in cloud computing and will define the future of IT. When a company needed to store and access more data they had one of two choices. One option would be to buy a bigger machine with more CPU, RAM, disk space, etc. This is known as scaling vertically. Of course, there is a limit to how big of a machine you can actually buy and this does not work when you start talking about internet scale. The other option would be to scale horizontally. This usually meant contacting some database vendor to buy a bigger solution. These solutions do not come cheap and therefore required a significant investment. Today, the source of data generated not only by the users and applications but also "machine-generated," and such data is exponentially leading the change in the Big Data space. Dealing with big datasets in the order of terabytes or even petabytes is a challenging. In Cloud computing environment a popular data processing engine for big data is Hadoop-MapReduce due to ease-of-use, scalability, and failover properties. Hadoop-MapReduce programming model consists of data processing functions: Map and Reduce. Parallel Map tasks are run on input data which is partitioned into fixed sized blocks and produce intermediate output as a collection of <key K, value V> pairs. These pairs are shuffled across different reduce tasks based on <K, V> pairs. Each Reduce task accepts only one key at a time and process data for that key and outputs the results as <K, V> pairs. The Hadoop-MapReduce architecture consists of one Job Tracker (also called as Master) and many Task Trackers (also called as Workers). The Job Tracker receives job

submitted from user, breaks it down into map and reduce tasks, assigns the tasks to Task Trackers, monitors the progress of the Task Trackers, and finally when all the tasks are complete, reports the user about the job completion. Each Task Tracker has a fixed number of map and reduce task slots that determine how many map and reduce tasks it can run at a time. HDFS supports reliability and fault tolerance of MapReduce computation by storing and replicating the inputs and outputs of a Hadoop job.

2. BIG DATA

Big data is a collection of data sets so large and complex which is also exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the structures of our current database architectures. Big Data is typically large volume of un-structured (or semi structured) 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. The core of Big Data is Hadoop which is a platform for distributing computing problems across a number of servers. It is first developed and released as open source by Yahoo, it implements the MapReduce approach pioneered by Google in compiling its search indexes. Hadoop's MapReduce involves distributing a dataset among multiple servers and operating on the data: the "map" stage. The partial results are then recombined: the "reduce" stage. To store data, Hadoop utilizes its own distributed filesystem, HDFS, which makes data available to multiple computing nodes. Big data explosion, a result not only of increasing Internet usage by people around the world, but also the connection of billions of devices to the Internet. Eight years ago, for example, there were only around 5 exabytes of data online. Just two years ago, that amount of data passed over the Internet over the course of a single month. And recent estimates put monthly Internet data flow at around 21 exabytes of data. This explosion of data - in both its size and form - causes a multitude of challenges for both people and machines.

Figure 1. Example of Different Sources of Big Data

3. HADOOP

Hadoop is a batch processing system for a cluster of nodes that provides the underpinnings of most BigData analytic activities because it bundle two sets of functionality most needed to deal with large unstructured datasets nsmely, Distributed file system and MapReduce processing. it is a project from the Apache Software Foundation written in Java to support data intensive distributed applications. Hadoop enables applications to work with thousands of nodes and petabytes of data. The inspiration comes from Google's MapReduce and Google File System papers. Hadoop's biggest contributor has been the search giant Yahoo, where Hadoop is extensively used across the business platform. Hadoop is an umbrella of sub-projects around distributed computing and although is best known for being a runtime environment

forMapReduce programs and its distributed filesystem HDFS, the other sub-projects provide complementary services and higher level abstractions. Some of the current sub-projects are:

Core: The Hadoop core consist of a set of components and interfaces which provides access to the distributed filesystems and general I/O (Serialization, Java RPC, Persistent data structures). The core components also provide "Rack Awareness", an optimization which takes into account the geographic clustering of servers, minimizing network traffic between servers in different geographic clusters.

MapReduce: Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of computer nodes. MapReduce uses the HDFS to access file segments and to store reduced results.

HDFS: Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS is, as its name implies, a distributed file system that provides high throughput access to application data creating multiple replicas of data blocks and distributing them on compute nodes throughout a cluster to enable reliable and rapid computations.

HBase: HBase is a distributed, column-oriented database. HBase uses HDFS for its underlying storage. It maps HDFS data into a database like structure and provides Java API access to this DB. It supports batch style computations using MapReduce and point queries (random reads). HBase is used in Hadoop when random, realtime read/write access is needed. Its goal is the hosting of very large tables running on top of clusters of commodity hardware.

Pig: It is a dataflow processing (scripting) language Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs. The main characteristic of Pig programs is that their structure can be substantially parallelized enabling them to handle very large data sets, simple syntax and advanced built-in functionality provide an abstraction that makes development of Hadoop jobs quicker and easier to write than traditional Java MapReduce jobs.

ZooKeeper: It is a cluster configuration tool and distributed serialization manager useful to build large clusters of Hadoop nodes, high performance coordination service for distributed applications. ZooKeeper centralizes the services for maintaining the configuration information, naming, providing distributed synchronization, and providing group services.

Hive: Hive is a data warehouse infrastructure built on top of Hadoop. Hive provides tools to enable easy data summarization, ad-hoc querying and analysis of large datasets stored in Hadoop files. It provides a mechanism to put structure on this data and it also provides a simple query language called Hive QL, based on SQL, enabling users familiar with SQL to query this data.

Chukwa: it is used for monitoring large distributed clusters of servers. It is a data collection system for monitoring large distributed systems. Chukwa includes a flexible and powerful toolkit for displaying, monitoring and analyzing results to make the best use of the collected data.

HCatalog: It is a storage management layer for Hadoop that enables users with different data processing tools. HCatalog's table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored.

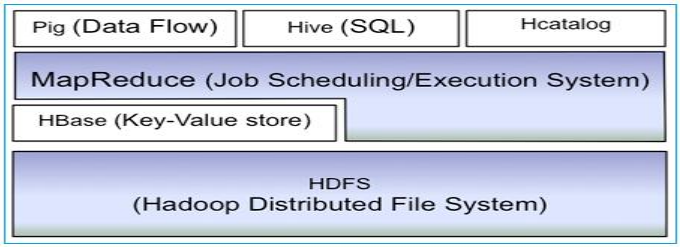

Figure 2. High Level Architechture of Hadoop

3.1. Architecture of Hadoop

Hadoop is a Map/Reduce framework that works on HDFS or on HBase. The main idea is to decompose a job into several and identical tasks that can be executed closer to the data (on the DataNode). In addition, each task is parallelized : the Map phase. Then all these intermediate results are merged into one result : the Reduce phase. In Hadoop, The JobTracker (a java process) is responsible for monitoring the job, managing the Map/Reduce phase, managing the retries in case of errors. The TaskTrackers (Java process) are running on the different DataNodes. Each TaskTracker executes the tasks of the job on the locally stored data.

The core of the Hadoop Cluster Architecture is given below.

HDFS (Hadoop Distributed File System): HDFS is the basic file storage, capable of storing a large number of large files.

MapRedure: MapReduce is the programming model by which data is analyzed using the processing resources within the cluster.

Each node in a Hadoop cluster is either a master or a slave. Slave nodes are always both a Data Node and a Task Tracker. While it is possible for the same node to be both a Name Node and a JobTracker

Name Node: Manages file system metadata and access control. There is exactly one Name Node in each cluster.

Secondary Name Node: Downloads periodic checkpoints from the name Node for fault-tolerance. There is exactly one Secondary Name Node in each cluster.

Job Tracker: Hands out tasks to the slave nodes. There is exactly one Job Tracker in each cluster.

Data Node: Holds file system data. Each data node manages its own locally-attached storage and stores a copy of some or all blocks in the file system. There are one or more Data Nodes in each cluster.

*Task Tracker***:** Slaves that carry out map and reduce tasks. There are one or more Task Trackers in each cluster.

3.2 Uses of Hadoop

- Building search index at Google, Amazon, Yahoo
- Analyzing user logs, data warehousing and analytics
- Used for large scale machine learning and data mining applications
- Legacy data processing where it requires massive computational

4. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

An HDFS cluster has two types of node operating in a master-worker pattern: a NameNode (the master) and a number of DataNodes (workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. The namenode also knows the datanodes on which all the blocks for a given file are located. Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing. Name Node decides about replication of data blocks. In a typical HDFS, block size is 64MB and replication factor is 3 (second copy on the local rack and third on the remote rack). The Figure 4 shown architecture distributed file system HDFS. Hadoop MapReduce applications use storage in a manner that is different from general-purpose computing. To read an HDFS file, client applications simply use a standard Java file input stream, as if the file was in the native filesystem. Behind the scenes, however, this stream is manipulated to retrieve data from HDFS instead. First, the Name Node is contacted to request access permission. If granted, the Name Node will translate the HDFS filename into a list of the HDFS block IDs comprising that file and a list of Data Nodes that store each block, and return the lists to the client. Next, the client opens a connection to the "closest" Data Node (based on Hadoop rack-awareness, but optimally the same node) and requests a specific block ID. That HDFS block is returned over the same connection, and the data delivered to the application. To write

data to HDFS, client applications see the HDFS file as a standard output stream. Internally, however, stream data is first fragmented into HDFS-sized blocks (64MB) and then smaller packets (64kB) by the client thread. Each packet is enqueued into a FIFO that can hold up to 5MB of data, thus decoupling the application thread from storage system latency during normal operation. A second thread is responsible for dequeuing packets from the FIFO, coordinating with the Name Node to assign HDFS block IDs and destinations, and transmitting blocks to the Data Nodes (either local or remote) for storage. A third thread manages acknowledgements from the Data Nodes that data has been committed to disk.

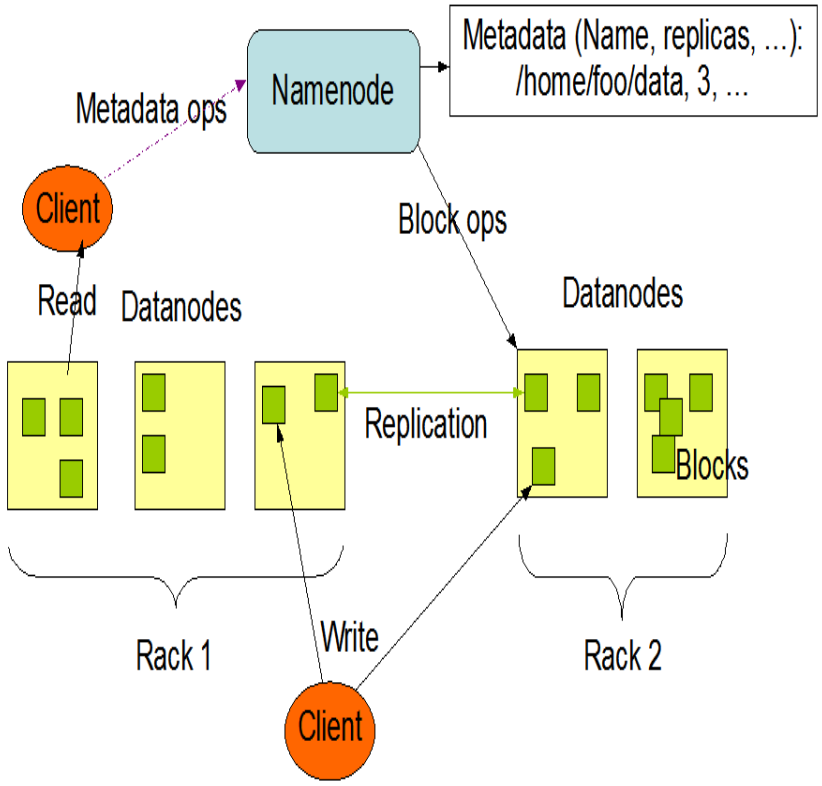

Figure 3. Hadoop Distributed Cluster File System Architechture

Source: http://hadoop.apache.org/docs/r0.17.2/hdfs_design.html

5. MAP-REDUCE

MapReduce is a data processing or parallel programming model introduced by Google. In this model, a user specifies the computation by two functions, Map and Reduce. In the mapping phase, MapReduce takes the input data and feeds each data element to the mapper. In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result. In simple terms, the mapper is meant to filter and transform the input into something that the reducer can aggregate over. The underlying MapReduce library automatically parallelizes the computation, and handles complicated issues like data distribution, load balancing and fault tolerance. Massive input, spread across many machines, need to parallelize. Moves the data, and provides scheduling, fault tolerance. The original MapReduce implementation by Google, as well as its open-source counterpart, Hadoop, is aimed for parallelizing computing in large clusters of commodity machines. Map Reduce has gained a great popularity as it gracefully and automatically achieves fault tolerance. It automatically handles the gathering of results across the multiple nodes and returns a single result or set. MapReduce model advantage is the easy scaling of data processing over multiple computing nodes.

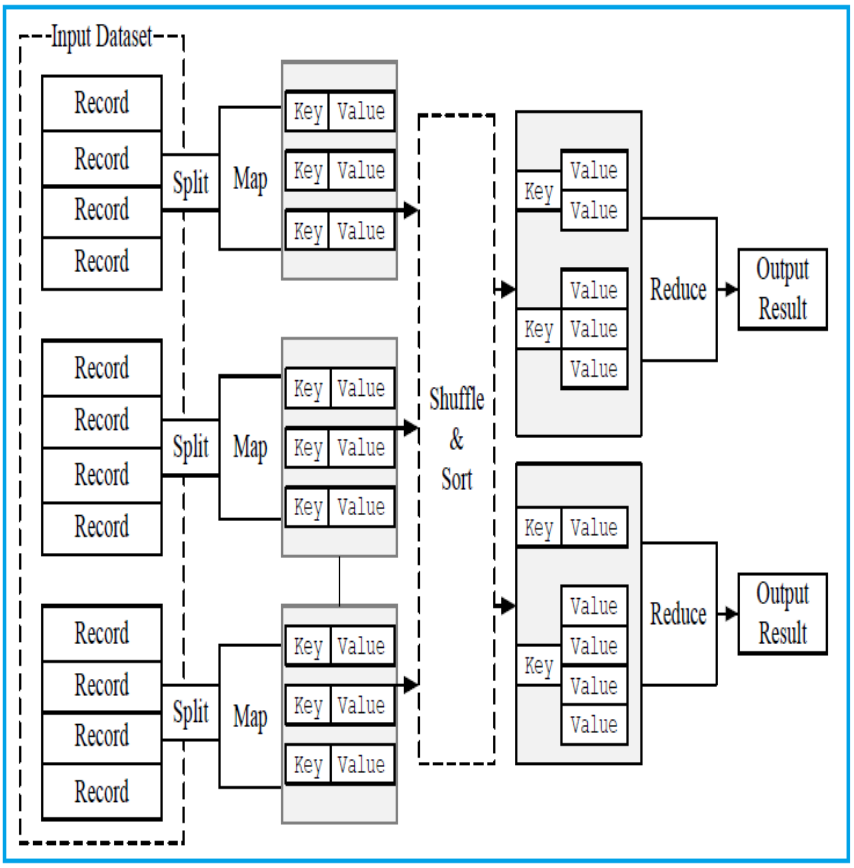

Figure 4. Execution process of MapReduce Programming Model

5.1 MapReduce Architecture & Implementation

Programming model: 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. There are three main roles: the master, mappers, and reducers. The single master acts as the coordinator responsible for task scheduling, job management, etc. MapReduce is built upon a distributed file system (DFS) which provides distributed storage. Figure 5 shows the execution process of MapReduce. The input data is split into a set of M blocks, which will be read by M mappers through DFS I/O. Each mapper will process the data by parsing the key/value pair and then generate the intermediate result that is stored in its local file system. The intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together (the shuffle phase). If the memory size is limited, an external sort might be used to deal with large amounts of data at one time. The locations of the intermediate results will be sent to the master who notifies the reducers to prepare to receive the intermediate results as their input. Reducers then use Remote Procedure Call (RPC) to read data from mappers. The user defined reduce function is then applied to the sorted data; basically, key pairs with the same key will be reduced in some way, depending on the user defined reduce function. Finally the output will be written to DFS.

Fault tolerance: MapReduce is designed to be fault tolerant because failures are common phenomena in large scale distributed computing and it includes worker failure and master failure.

Worker failure: The master pings every mapper and reducer periodically. If no response is received for a certain amount of time, the machine is marked as failed. The ongoing task and any tasks completed by this mapper will be re-assigned to another mapper and executed from the very beginning. Completed reduce tasks do not need to be re-executed because their output is stored in the global file system.

Master failure: Since the master is a single machine, the probability of master failure is very small. MapReduce will re-start the entire job if the master fails. There are currently three popular implementations of the MapReduce programming model namely Google MapReduce, Apache Hadoop, Stanford Phoenix.

5.2 Execution Process in MapReduce Programming Model

In MapReduce programming model and a MapReduce job consists of a map function, a reduce function, and When a function called the below steps of actions take place.

- MapReduce will first divide the data into N partitions with size varies from 16MB to 64MB
- Then it will start many programs on a cluster of different machines. One of program is the master program; the others are workers, which can execute their work assigned by master. Master can distribute a map task or a reduce task to an idle worker.
- If a worker is assigned a Map task, it will parse the input data partition and output the key/value pairs, then pass the pair to a user defined Map function. The map function will buffer the temporary key/value pairs in memory. The pairs will periodically be written to local disk and partitioned into P pieces. After that, the local machine will inform the master of the location of these pairs.
- If a worker is assigned a Reduce task and is informed about the location of these pairs, the Reducer will read the entire buffer by using remote procedure calls. After that, it will sort the temporary data based on the key.
- Then, the reducer will deal with all of the records. For each key and according set of values, the reducer passes key/value pairs to a user defined Reduce function. The output is the final output of this partition.
- After all of the mappers and reducers have finished their work, the master will return the result to users' programs. The output is stored in F individual files.

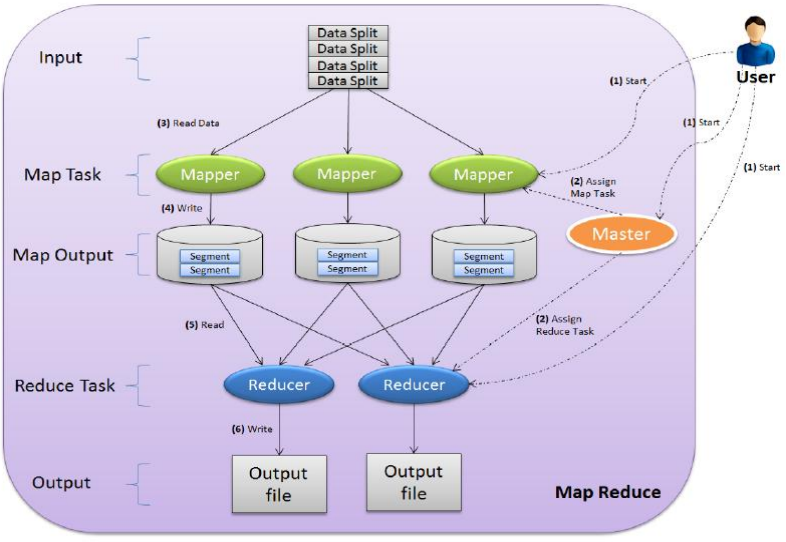

Figure 5. Architechture of MapReduce, Source: [12]

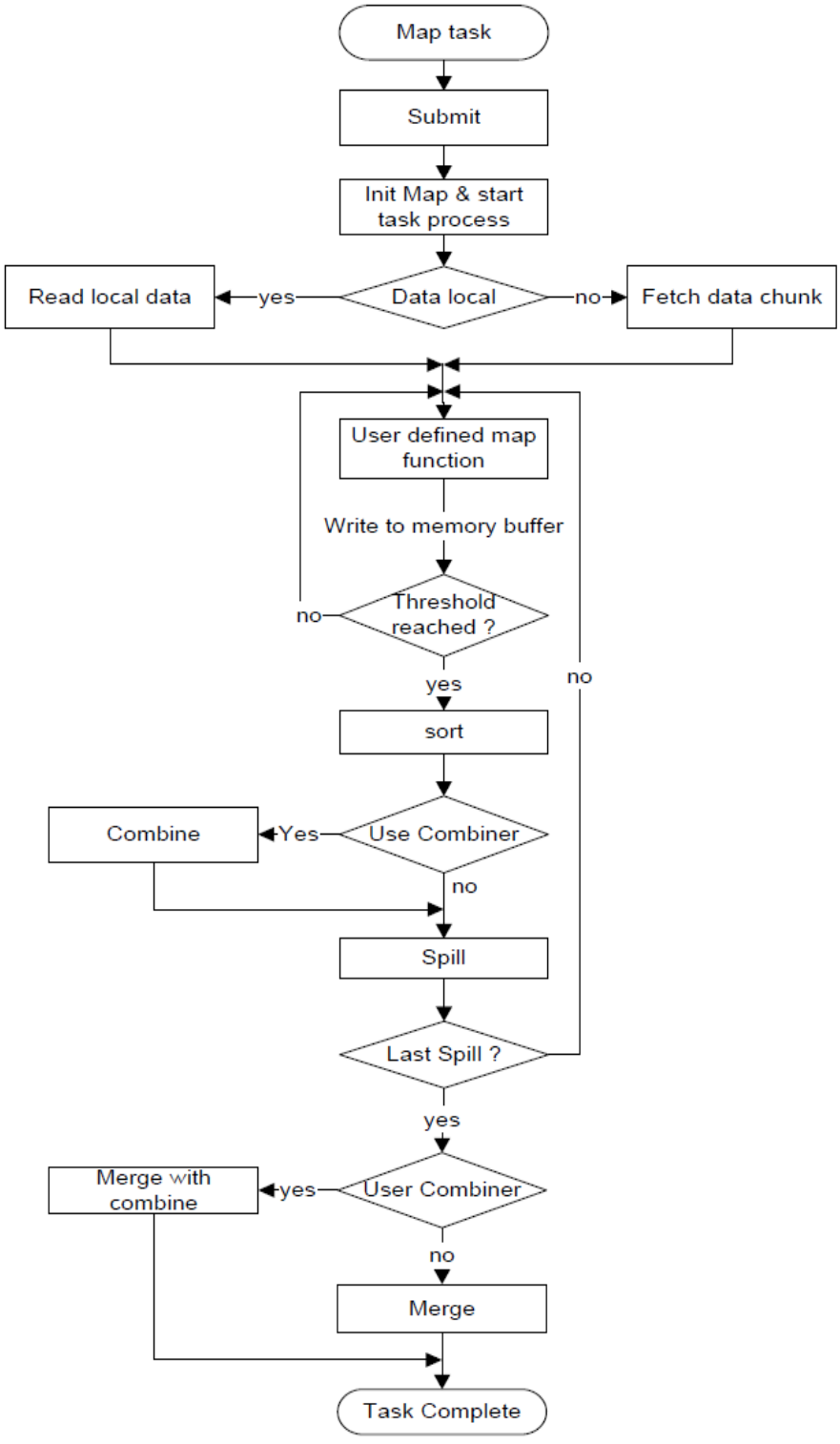

Figure 6. Flow Chart for Map task

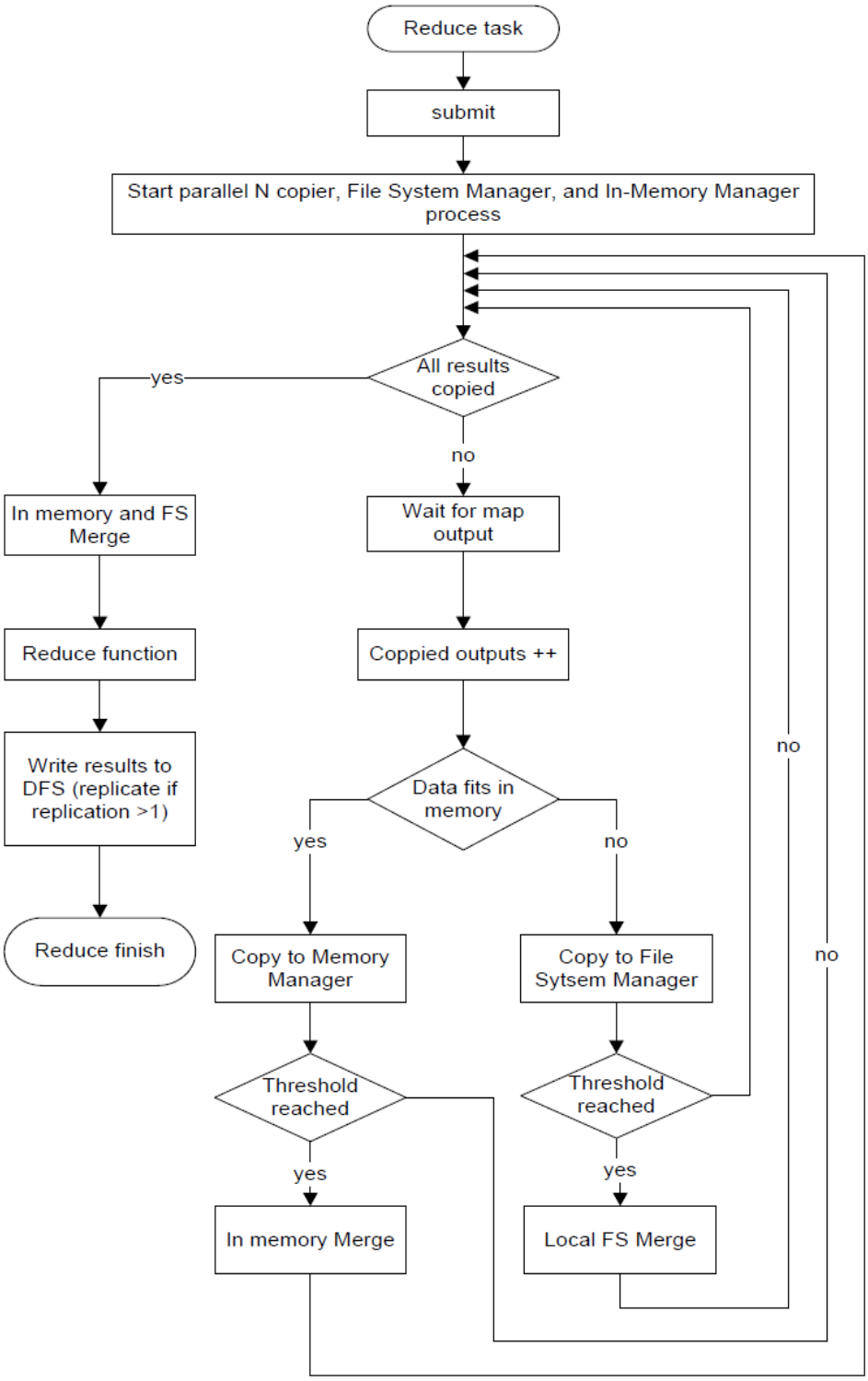

Figure 7. Flow Chart of Reduce Task

5.3 A MapReduce Progamming Model Example

In essence MapReduce is just a way to take a big task and split it into discrete task that can be done in parallel. A simple problem that is often used to explain how MapReduce works in practice consists in counting the occurrences of single words within a text. This kind of problem can be easily solved by launching a single MapReduce job as given in the below figure 8.

- Input data
- Input data are partitioned into smaller chunks of data
- For each chunk of input data, a "map task" runs which applies the map function resulting output of each map task is a collection of key-value pairs.
- The output of all map tasks is shuffled for each distinct key in the map output; a collection is created containing all corresponding values from the map output.
- For each key-collection resulting from the shuffle phase, a "reduce task" runs which applies the reduce function to the collection of values.
- The resulting output is a single key-value pair.
- The collection of all key-value pairs resulting from the reduce step is the output of the MapReduce job.

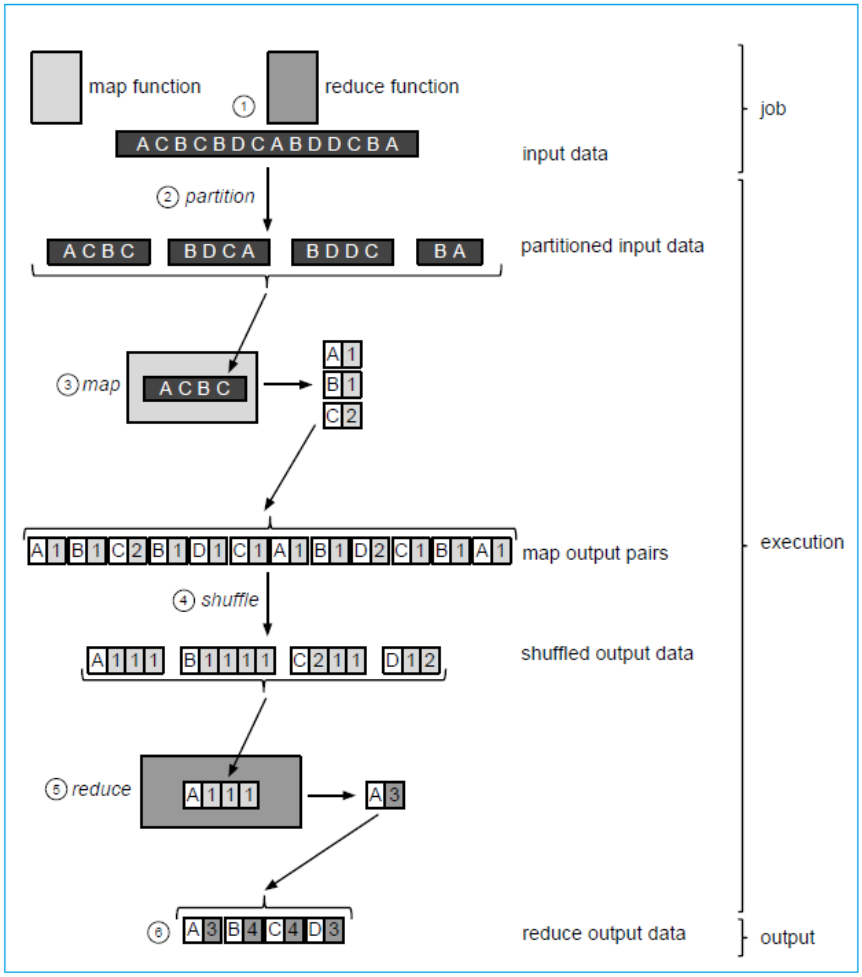

Figure 8. A MapReduce Progamming Model Example, Source: [14]

5.4 Advantages of the MapReduce

Using MapReduce, a programmer defines his job with only Map and Reduce functions, without having to specify physical distribution of his job across nodes.

Flexible: MapReduce does not have any dependency on data model and schema.
Fault tolerance: MapReduce is highly fault-tolerant.
High scalability: The best advantage of using MapReduce is high scalability.

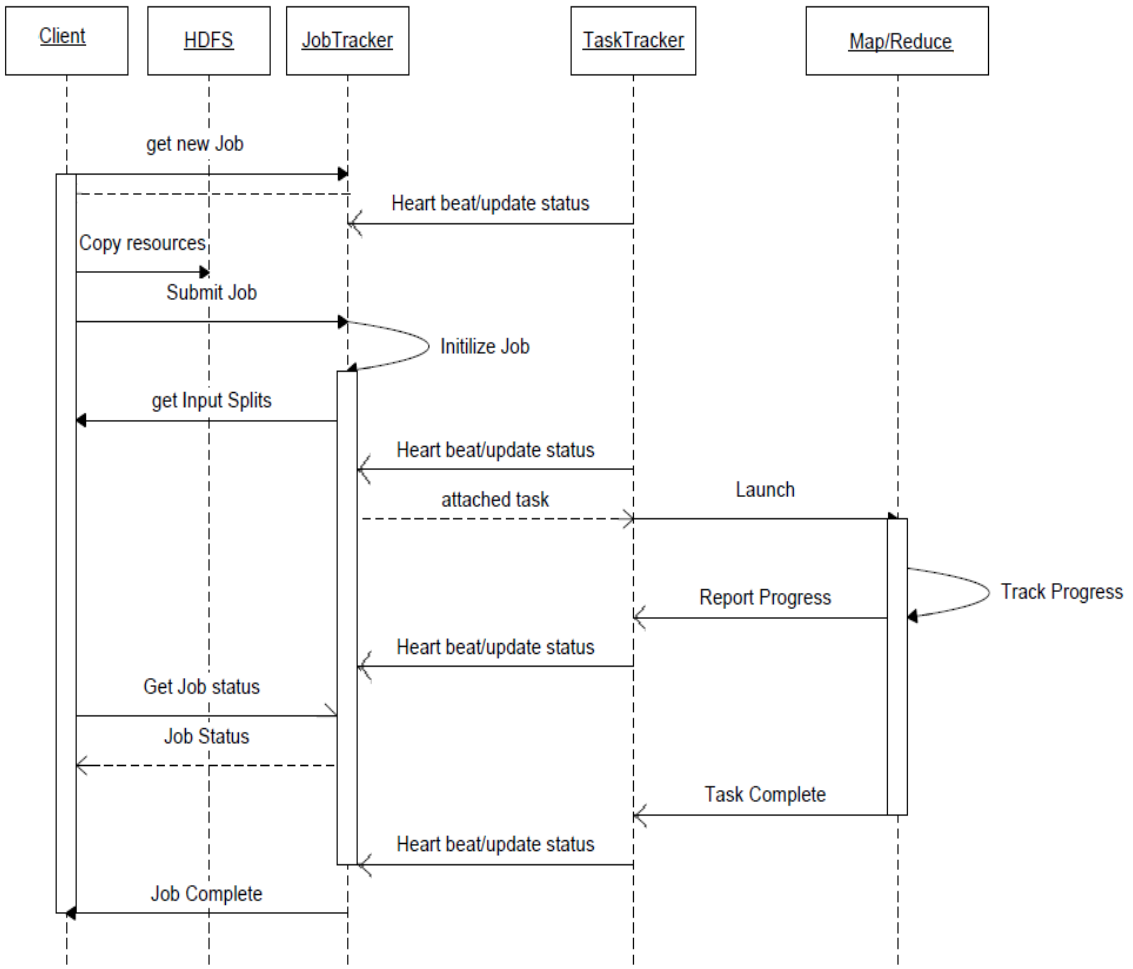

Figure 9. Sequence Diagram for Hadoop- MapReduce Programming Model

6. CONCLUSION

Cloud technology progress & increased use of the Internet are creating very large new datasets with increasing value to businesses and processing power to analyze them affordable. BigData is still in its early infancy but it is already having a profound effect on technology companies and the way we do business. The size of these datasets suggests that exploitation may well require a new category of data storage and analysis systems with different architectures. Hadoop-MapReduce programming paradigm have a substantial base in the Big Data community due to the cost-effectiveness on commodity Linux clusters, and in the cloud via data upload to cloud vendors who have implemented Hadoop/HBase. The effectiveness and ease-of-use of the MapReduce method in parallelization involves many data analysis algorithms. HDFS, the Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to these information.

REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.

[2] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In EuroSys '10:

Proceedings of the 5th European conference on Computer systems, pages 265–278, New York, NY, USA, 2010. ACM.

[3] R. Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics BMC bioinformatics,11(Suppl 12):S1, 2010.

[4] A. Pavlo et al . A comparison of approaches to large-scale data analysis. In Proceedings of the ACM SIGMOD, pages 165–178, 2009.

[5] C. Ordonez et al . Relational versus Non-Relational Database Systems for Data Warehousing. In Proceedings of the ACM DOLAP, pages 67–68, 2010.

[6] F. Chang et al . Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS),
26(2):1–26, 2008.

[7] S. Melnik et al . Dremel: interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment,3(1-2):330–339, 2010.

[8] Cassandra Query Language Documentation. http://caqel.deadcafe.org/cql-doc. [online; accessed 25-July-2012].

[9] HBase: Bigtable-like structured storage for Hadoop HDFS. http://wiki.apache.org/hadoop/hbase. [online; accessed 26-July-2012.

[10] S. Ghemawat et al . The google file system. ACM SIGOPS Operating Systems Review, 37(5):29–43, 2003.

[11] HDFS (hadoop distributed file system) architecture. http://hadoop.apache.org/common/docs/current/hdfs

[12] TCP/IP Implementation of Hadoop Acceleration by Cong Xu A thesis submitted to the Graduate Faculty of Auburn University in partial fulfillment of the requirements for the Degree of Master of Science Auburn, Alabama Aug 4, 2012

[13] R. Lammel. Google's mapreduce programming model – revisited. Science of Computer Programming, 70(1):1 – 30, 2008.

[14] 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..

[15] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1029–1040, New York, NY, USA, 2007. ACM.

[16] T. Condie et al. MapReduce online. In Proceedings of the 7th USENIX conference on Networked systems design and implementation, pages 21–21, 2010.

[17] W. Jiang et al . A Map-Reduce System with an Alternate API for Multi-core Environments. In Proceedings of the 10th IEEE/ACM CCGrid, pages 84–93, 2010.

[18] H. Yang et al. Map-reduce-merge: simplified relational data processing on large clusters. In Proceedings of the 2007 ACM SIGMOD, pages 1029–1040, 2007. "Map-Reduce Online"; Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein UC Berkeley, Khaled Elmeleegy, Russell Sears Yahoo! Research

[19] M. Grant, S. Sehrish, J. Bent, and J. Wang. "Introducing Map-reduce to High End Computing". 3rd Petascale Data Storage Workshop, Nov 2008.

[20] Y. Hung-Chih, D. Ali, H. Ruey-Lung, and D.S. Parker, "Map-Reduce-Merg e: Simplified Relation al Data Processing On Large Clusters", in Proceedings of the 2007 ACM Sigmod International Conference on Management of Data Beijing", China: acm, 2007.

[21] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing Map-reduce to High End Computing", in Petascale Data Storage Workshop", 2008. pdsw '08. 3rd, 2008, pp. 1-6.

[22] Amdahl, G. (1967). Validity of the single processor approach to achieving large-scale computing capabilities (pp. 483–485).

[23] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1–7), 107–117.

[24] Cacheda, F., Plachouras, V., & Ounis, I. (2005). A case study of distributed information retrieval architectures to index one terabyte of text. Information Processing & Management, 41(5), 1141–1161.

[25] I. Adams, D.D.E. Long, E.L. Miller, S. Pasupathy, M.W. Storer, Maximizing efficiency by trading storage for computation, in: Workshop on Hot Topics in Cloud Computing, HotCloud'09, San Diego, CA, 2009, pp. 1–5.

[26] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (2009) 599–616.

[27] Dhruba Borthaku, The Hadoop Distributed File System: Architecture and Design. Retrieved from, 2010, http://hadoop.apache.org/common/.

BIOGRAPHY OF AUTHOR

Rabi Prasad Padhy is currently working as a Senior Software Engineer - Oracle India Private Ltd. Bangalore, Karnataka, India. He has achieved his MCA degree from Berhampur University. He carries more than 8 years of extensive IT Experience with MNC's like EDS, Dell, IBM and Oracle. His area of interests includes IT Infrastructure Optimization, Virtualization, Enterprise Grid Computing, Cloud Computing and Cloud Databases. He has published several research papers in national and international journals. He is a certified professional for Oracle & Microsoft Database, Oracle Apps, Linux & Solaris System Admin and also ITIL Certified.