International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Survey Paper on Big Data Processing and Hadoop Components

Poonam S. Patil[1], Rajesh. N. Phursule[2]

Department of Computer Engineering
JSPM's Imperial College of Engineering and Research, Pune, India

Abstract: ***As big data continues down its path of growth, a major challenge has become how to deal with the explosion of data and analysis of this data. For such data-intensive applications, the Apache Hadoop Framework has recently attracted a lot of attention. This framework Adopted MapReduce, it is a programming model and an associated implementation for processing and generating large data sets. Hadoop Provides: Distributed File System, Job scheduling, Resource Management Capabilities, and Java API for writing Application E.g. Java Map-Reduce, Streaming MapReduce, Crunch, Pig latin, Hive, Oozie etc. 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. Many real world tasks are expressible in this model, Hadoop gives the flexibility to use any language to write an algorithms. In this paper we will briefly introduce the MapReduce framework based on Hadoop and the current state-of-the-art in MapReduce algorithms for big data analysis.***

Keywords: Big data, Hadoop, MapReduce, Hive, Hbase, Distributed Data, Relational Database, NoSql

1. Introduction

A. Big Data

We have entered an era of Big Data [1]. Through better analysis of the large volumes of data that are becoming available, there is the potential for making faster advances in many scientific disciplines and improving the profitability and success of many enterprises. However, many technical challenges described in this paper must be addressed before this potential can be realized fully. The challenges include not just the obvious issues of scale, but also heterogeneity, lack of structure, error-handling, privacy, timeliness, provenance, and visualization, at all stages of the analysis pipeline from data acquisition to result interpretation. These technical challenges are common across a large variety of application domains, and therefore not cost-effective to address in the context of one domain alone. Furthermore, these challenges will require transformative solutions, and will not be addressed naturally by the next generation of industrial products. We must support and encourage fundamental research towards addressing these technical challenges if we are to achieve the promised benefits of Big Data.

The contributions of this paper are:

- An overview of Big Data[1]
- State-of-the art in Hadoop MapReduce[3] framework
- Introduction to the various components of Hadoop

B. Vs of Big Data

Volume: Volume refers to amount of data. Volume of data stored in enterprise repositories have grown from megabytes and gigabytes to petabytes.
Variety: Different types of data and sources of data. Data variety exploded from structured and legacy data stored in enterprise repositories to unstructured, semi structured, audio, video, XML etc.

Velocity: Velocity refers to the speed of data processing. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

2. Traditional Database Systems and Hadoop

A. What is a Relational Database?

Traditional RDBMS (relational database management system) have been the de facto standard for database management System. The architecture of RDBMS is such that data is organized in a highly-structured manner, following the relational model. Though, RDBMS is now considered to be a declining database technology. While the precise organization of the data keeps the warehouse very "neat", the need for the data to be well-structured actually becomes a substantial burden at extremely large volumes, resulting in performance declines as size gets bigger. Thus, RDBMS is generally not thought of as a scalable solution to meet the needs of 'big' data.

B. What is NoSQL?

NoSQL (commonly referred to as "Not Only SQL") represents a completely different framework of databases that allows for high-performance, agile processing of information at massive scale. In other words, it is a database infrastructure that has been very well-adapted to the heavy demands of big data.

The efficiency of NoSQL can be achieved because unlike relational databases that are highly structured, NoSQL databases are unstructured in nature, trading off stringent consistency requirements for speed and agility. NoSQL centers on the concept of distributed databases, where unstructured data may be stored across multiple processing nodes, and often across multiple servers. This distributed architecture allows NoSQL databases to be horizontally

scalable; as data continues to explode, just add more hardware to keep up, with no slowdown in performance. The NoSQL distributed database infrastructure has been the solution to handling some of the biggest data warehouses on the planet – i.e. the likes of Google, Amazon, and the CIA.

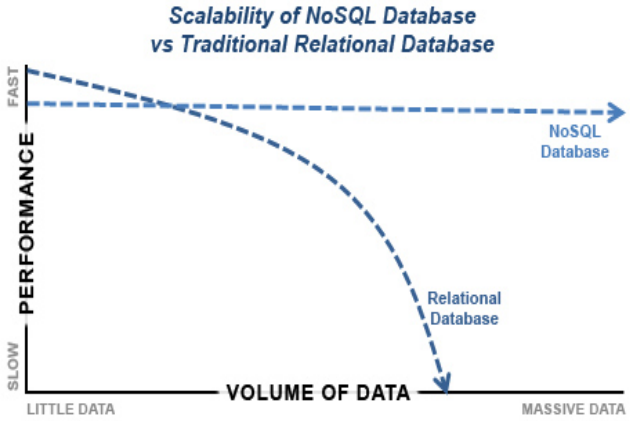

Figure 1: Traditional DB Vs NoSQL DB

C. Hadoop

Hadoop Origin: It is Created By Doug Cutting as a Part of Apache Product [9],

2004 Google publish GFS Paper,
2005 Nutch (open Source web search) uses MapReduce
2008 MapReduce becomes Apache top-level project, was lucene sub-project before
2009 Yahoo used Hadoop to start 1TB in 62sec
2013 Hadoop is used by hundreds of the companies

Hadoop is not a type of database, but rather a software ecosystem that allows for massively parallel computing. It is an enabler of certain types NoSQL distributed databases (such as HBase), which can allow for data to be spread across thousands of servers with little reduction in performance. A staple of the Hadoop ecosystem is MapReduce, a computational model that basically takes intensive data processes and spreads the computation across a potentially endless number of servers (generally referred to as a Hadoop cluster). It has been a game-changer in supporting the enormous processing needs of big data[1]; a large data procedure which might take 20 hours of processing time on a centralized relational database system, may only take 3 minutes when distributed across a large Hadoop cluster of commodity servers, all processing in parallel.

3. MapReduce Framework

A. Master-Slave Architecture

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of MapReduce[2] runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

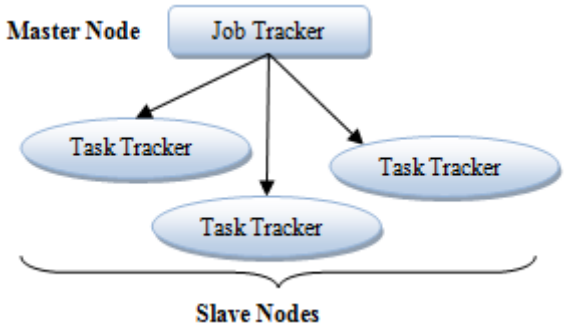

Figure 2: Hadoop Master Slave Architecture

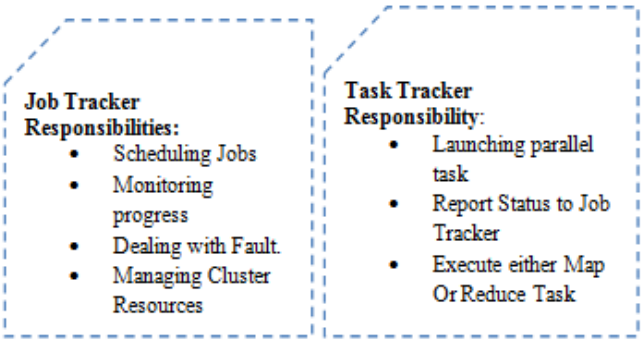

Figure 3: Hadoop Job and Task Tracker Responsibility

In the initial implementations [3] of Hadoop, MapReduce is designed as a master-slave architecture as shown in fig 2 the responsibilities of task and job tracker are shown in fig 3. The JobTracker is the master managing the cluster resources, scheduling jobs, monitoring progress and dealing with fault-tolerance. On each of the slave nodes, there exists a TaskTracker process, responsible for launching parallel tasks and reporting their status to the JobTracker. The slave nodes are statically divided into computing slots, available to execute either Map or Reduce tasks. The Hadoop community realized the limitations of this static model and recently redesigned the architecture to improve cluster utilization and scalability. The new design, YARN [7] is presented.

B. Programming Model of MapReduce

The computation [3] of MapReduce takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce. 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 these values together to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

C. 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.

Pseudo Code 1 for Word Count

```
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. In addition, the user writes code to fill in a mapreduce specification object with the names of the input and output files and optional tuning parameters. The user then invokes the MapReduce function, passing it to the specification object. The user's code is linked together with the MapReduce[2] library (implemented in C++) few simple examples of interesting programs that can be easily expressed as MapReduce computations.

Distributed Grep: 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.

Count of URL Access Frequency: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits <URL, TotalCount> pair.

ReverseWeb-Link Graph: The map function <target, source> pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>

Term-Vector per Host: A term vector summarizes the most important words that occur in a document or a set of documents as a list of <word, frequency> pairs. The map function emits a <hostname, term vector> pair for each input document (where the hostname is extracted from the URL of the document). The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits <hostname, term vector> pair.

D. 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.

map (k1,v1) → list(k2,v2)
reduce (k2,list(v2)) → list(v2)

That is, the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values.

4. Hadoop Component

There is an extensive list of products and projects that either extend Hadoop's functionality or expose some existing capability in new ways, like MapReduce for distributed data processing, HDFS for distributed file system, Hive[11] for distributed data warehouse[8] and provides sql based query language, HBase[12] or Accumulo for distributed column based database, Pig[6] provides an abstraction layer with the help of scripts in the language Pig Latin, which are translated to MapReduce jobs. Other examples of projects built on top of Hadoop include Apache Sqoop, Apache Oozie, and Apache Flume, YARN for improvement of cluster utilization and scalability etc

A. HDFS

The Hadoop Distributed File System (HDFS)[10] is a distributed file system designed to run on commodity hardware its architecture is shown in fig 4. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.HDFS is part of the Apache Hadoop Core project.

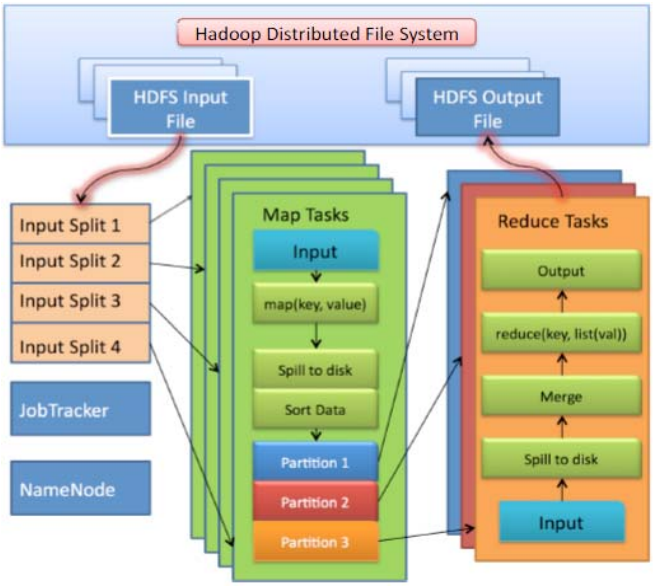

Figure 4: Hadoop MapReduce and HDFS

Hadoop MapReduce[1] jobs read their input data from HDFS and also write their output to it. HDFS has been very popular because of its scalability, reliability and capability of storing very large files. HDFS [10] applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A Map/Reduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

B. HBase

Apache HBase[8] is distributed column based database - like layer built on Hadoop designed to support billions of messages per day, HBase is massively scalable and delivers fast random writes as well as random and streaming reads. It also provides row-level atomicity guarantees, but no native cross-row transactional support. From a data model perspective, column-orientation gives extreme flexibility in storing data and wide rows allow the creation of billions of indexed values within a single table. HBase[12] is ideal for workloads that are write-intensive, need to maintain a large amount of data, large indices, and maintain the flexibility to scale out quickly.

C. Hive

Hive [8] is a technology developed at Facebook that turns Hadoop into a data warehouse [8] complete with a dialect of SQL for querying. Being a SQL dialect, HiveQL[11] is a declarative language. The architecture of Hive is shown in fig 5. In PigLatin[6], you specify the data flow, but in Hive we describe the result we want and Hive figures out how to build a data flow to achieve that result. Unlike Pig, in Hive a schema is required, but you are not limited to only one schema. Like PigLatin and the SQL, HiveQL[4] itself is a relationally complete language but it is not a Turing complete language. It can also be extended through UDFs just like Piglatin to be a Turing complete. Hive is a technology for turning the Hadoop into a data warehouse, complete with SQL dialect for querying it. Hive works in terms of tables. There are two kinds of tables you can create: managed tables whose data is managed by Hive and external tables whose data is managed outside of Hive. Another option Hive provides for speeding up queries is bucketing. Like partitioning, bucketing splits up the data by a particular column, but in bucketing you do not specify the individual values for that column that correspond to buckets, you simply say how many buckets to split the table into and let Hive figure out how to do it.

D. PIG

Pig was initially developed at Yahoo Research around 2006 but moved into the Apache Software Foundation in 2007. Pig consists of a language and an execution environment. Pig's language, called as PigLatin[6], is a data flow language - this is the kind of language in which you program by connecting things together. Pig can operate on complex data structures, even those that can have levels of nesting. Unlike SQL, Pig does not require that the data must have a schema, so it is well suited to process the unstructured data. But, Pig can still leverage the value of a schema if you want to supply one. PigLatin is relationally complete like SQL, which means it is at least as powerful as a relational algebra. Turing completeness requires conditional constructs, an infinite memory model, and looping constructs. PigLatin is not Turing complete on itself, but it can be Turing complete when extended with User--Defined Function.

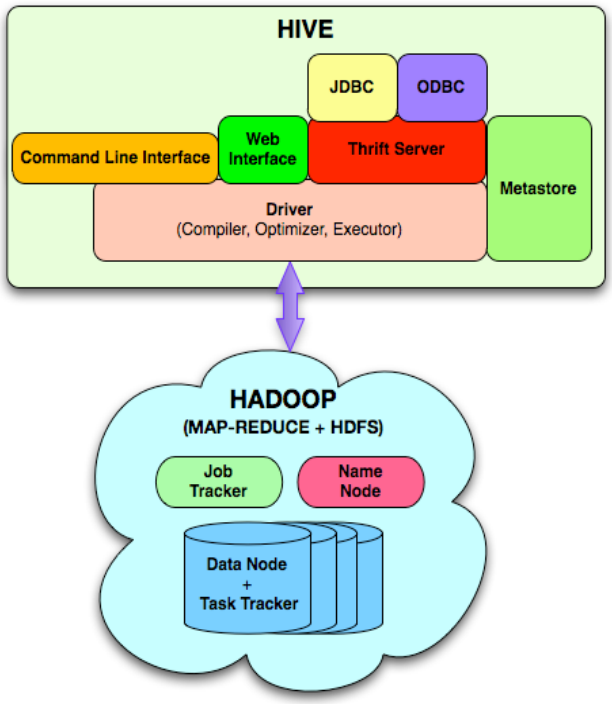

Figure 5: Hive System Architecture

There are three different ways to run Pig. You can run your PigLatin[13] code as a script, just by passing the name of your script file to the pig command. You can run it interactively through the grunt command line launched using Pig with no script argument. Finally, you can call into Pig from within Java using Pig's embedded form.

E. YARN

YARN [4], Yet Another Resource Negotiator, is included in the latest Hadoop release and its goal is to allow the system to serve as a general data-processing framework. It supports programming models [7] other than MapReduce, while also improving scalability and resource utilization. YARN makes no changes to the programming model or to HDFS. It consists of a re-designed runtime system, aiming to eliminate the bottlenecks of the master-slave architecture [4]. The responsibilities of the Job Tracker are split into two different processes, the Resource Manager and the Application Master. The Resource Manager handles resources dynamically, using the notion of containers, instead of static Map/Reduce slots. Containers are configured based on information about available memory, CPU and disk capacity. It also has a pluggable scheduler, which can use different strategies to assign tasks to available nodes. The Application Master is a framework-specific process, meaning that it allows other programming models to be executed on top of YARN, such as MPI or Spark [12]. It negotiates resources with the Resource Manager and supervises the scheduled tasks.

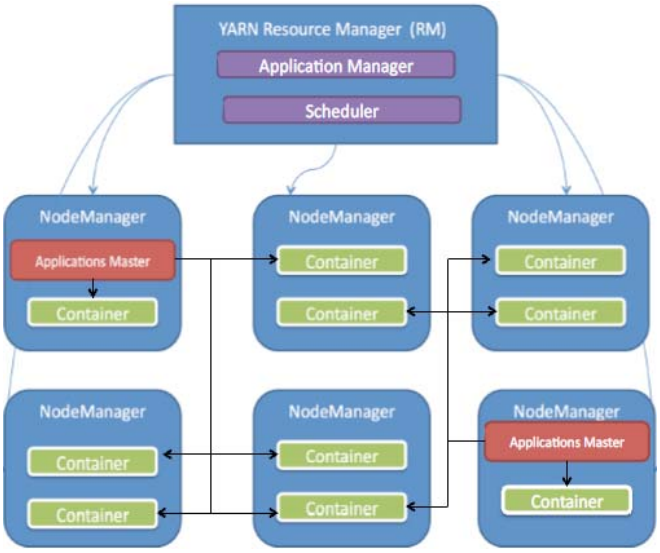

Figure 6: YARN System Architecture

Fig 6 shows the System Architecture of YARN. The lifecycle of a Map-Reduce job running in YARN [7] is as follows:

1. Hadoop MR JobClient submits the job to the YARN ResourceManager (ApplicationsManager) rather than to the Hadoop MapReduce [2] JobTracker.
2. The YARN RM-ASM negotiates the container for the MR Application Master with the Scheduler (RMS) and then launches the MR AM for the job.
3. The MR AM starts up and registers with the RMASM.
4. The Hadoop MapReduce JobClient polls the ASM to obtain information about the MR AM and then directly talks to the AM for status, counters, etc.
5. The MR AM computes input-splits and constructs resource requests for all maps and reducers to the YARN Scheduler.
6. The MR AM runs the requisite job setup APIs of the Hadoop MR Output Committer for the job.
7. The MR AM submits the resource requests for the map/reduce tasks to the YARN Scheduler (RM-S), gets containers from the RM and schedules appropriate tasks on the obtained containers by working with the Node Manager for each container.
8. The MR AM monitors the individual tasks to completion, requests alternate resource if any of the tasks fail or stop responding.
9. The MR AM also runs appropriate task cleanup code of completed tasks of the Hadoop MR Output Committer.
10. Once the entire map and reduce tasks are complete, the MR AM runs the requisite job commit APIs of the Hadoop MR Output Committer for the job.
11. The MR AM then exits since the job is complete.

5. Conclusion and Future Scope

The MapReduce programming model has been successfully used at Google for many different purposes the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. MapReduce is easy to parallelize and distribute computations and to make such computations fault tolerant. And there are extensive list of products and projects that either extend Hadoop's functionality or expose some existing capability in new ways.

MapReduce model specifically, are an active research area, still at its infancy. Currently, the interest for MapReduce is at its peak and there exist a lot of problems and challenges still to be addressed. There lies a bright future ahead for Big Data, as businesses and organizations realize more and more the value of the information they can store and analyze. Developing ways to process the vast amounts of data available drives business innovation, health discoveries, science progress and allows us to find novel ways to solve problems, which we considered very hard or even impossible in the past.

References

[1] Apache Hadoop. Available at http://hadoop.apache.org
[2] Apache HDFS. Available at http://hadoop.apache.org/hdfs
[3] Apache Hive. Available at http://hive.apache.org
[4] Apache HBase. Available at http://hbase.apache.org
[5] A community white paper developed by leading researchers across the United States "Challenges and Opportunities with Big Data"
[6] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu and Raghotham Murthy "Hive – A Petabyte Scale Data Warehouse Using Hadoop" By Facebook Data Infrastructure Team
[7] dhruba,jssarma,jgray,kannan,nicolas,hairong,krangana than,dms,aravind.menon,rash,rodrigo,amitanand.s "Apache Hadoop Goes Realtime at Facebook"

SIGMOD .'11, June 12.–16, 2011, Athens, Greece. Copyright 2011 ACM 978-1-4503-0661-4/11/06

[8] Jeffrey Dean and Sanjay Google, Inc." MapReduce: Simplified Data Processing on Large Clusters"

[9] Kyuseok Shim Seoul National University shim@ee.snu.ac.kr "MapReduce Algorithms for Big Data Analysis"

[10] Sanjeev Dhawan1, Sanjay Rathee2, Faculty of Computer Science & Engineering, Research Scholar " Big Data Analytics using Hadoop Components like Pig and Hive" AIJRSTEM 13- 131; © 2013, AIJRSTEM.

[11] Vasiliki Kalavri, Vladimir VlassovKTH The Royal Institute of Technology Stockholm, Sweden kalavri@kth.se "MapReduce: Limitations, Optimizations and Open Issues". TrustCom/ISPA/IUCC,Page1031-1038,IEEE,(2013)

[12] Vinod Kumar, Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal,Mahadev, Konar, Robert Evans,Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, Eric Baldeschwieler "Apache Hadoop YARN: Yet Another Resource Negotiator" SoCC'13, 1–3 Oct. 2013, Santa Clara, California, USA. ACM978-1-4503-2428-1