Processing of Business Intelligence using Hadoop Framework

Borivoje Milosevic[1], Srdjan Jovkovic[2], Marko Jankovic
[1]College of Applied Technical Sciences University Nis
A. Medvedeva 20, Nis 18000, Serbia
{borivojemilosevic@yahoo. com}

[2]College of Applied Technical
Sciences University Nis, A. Medvedeva 20
Nish 18000, Serbia
{srdjansms11@gmail.com}

ABSTRACT: *To ensure the distributed processing of big data volume, we have developed the Hadoop software with the help of the programming interfaces. To identify the failure if any during the application level, we have designed to receive signals. With the help of MapReduce program model, we have developed the Hadoop framework for the processing of unstructured data. This work detailed the distributed file system for Hadoop file system.*

Keywords: Big Data, Hadoop, MapReduce, Hadoop Distributed Filesystem, Java Virtual Machine

Received: 19 May 2020, Revised 9 July 2020, Accepted 3 August 2020

DOI:10.6025/jdp/2020/10/4/105-111

1. Introduction

With the development of computer technology, it is now possible to manage huge amounts of data that were previously impossible to process and that could be used only with the help of supercomputers and at great expense. System prices have fallen as a result of new techniques and distributed processing, which are currently in the focus. The real breakthrough in Big data technology occurred when companies such as Yahoo, Google, and Facebook came to the conclusion that they could process large amounts of data which their products generate. These companies have been tasked to find new technologies that will enable them to save, access, manage and analyze vast amounts of data in real time, in such a way that they can process and fairly and properly utilize the amount of data they have and include them in their networks. Their solutions that have emerged have led to changes in the market data management. In particular, the innovations that MapReduce, Hadoop and Big Table brought forth, proved to be a spark that led to a new generation of data management. In support of this technology, the Apache project is developing a number of projects related to open source Hadoop with: Ambari, Avro, HBase, Cassandra, Chukwu, Mahout, Hive, Pig, Spark. Cassandra's Hadoop is distributed "NoSQL" database open source, which is recognized as the basis for the following generation of business intelligence systems, used by Ebay, Twitter and many other companies whose charac-

teristic is that they have large amounts of active data. The largest known Cassandra cluster has over 300 TB of data on more than 400 servers.

These technologies emphasize one of the most fundamental problems, which is the ability to process large amounts of data in an efficient and timely manner, in a manner that is cost effective and does not require large expenditures.

The scope of what is now considered as a Big Data is wide, and the definitions are unclear, even contradictory. The most widely accepted definition of the term Big Data is derived from the Meta Group analysis (now Gartner) which was conducted in 2001. According to this definition, the term Big Data stands for information resource large quantities, high speed and high diversity of data that require new and innovative methods of processing and optimization of information, improving access to the contents of the data and decision-making. This is the definition of so-called ‘’3V Dimensions’’’: Volume, Variety, Velocity where the amount of data goes up to the peta byte, and the variety of data to be processed in real time is tight (structured, unstructured, text, blogs, multimedia, etc.), Figure 1. According to some estimates, about 80 percent of data is not of a numeric type, but they still need to be involved in the procedure of analysis and decision making.

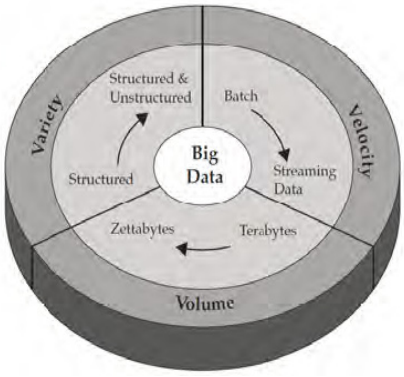

Figure 1. 3D Big Data outlook

Also, when talking about the characteristics, it is important to note two new dimensions:

Versatility: How the data is subject to change? In addition to large quantities and speeds of data processing, data streams can become quite spread with time. This can be explained by a phenomenon in popular media, where one and the same information is repeated many times. Such exceptions are very difficult to handle, especially when you take into account the recent rise in popularity of social networks.

Complexity: How difficult is data processing? When dealing with large amounts of data, they typically come from different sources. In many cases it is a fatal pair: filtering and transforming a piece of information in any way. However, it is necessary to connect the relationships between data and of control.

10.000 Credit card transactions are done every second in the world. 340 million tweets are sent daily, which is some 4.000 tweets per second. Facebook has more than 901 million active users who generate data daily, based on their mutual interaction. More than 5 billion people are calling, sending SMS, MMS, tweeting or surfing the Internet on mobile devices.

2. Map Reduce Programming

In contrast with traditional relational database–oriented information which organizes data into fairly rigid rows and columns that are stored in tables - MapReduce uses key/value pairs. MapReduce is a programming model for processinglarge data sets using parallel distributed algorithms in a cluster. MapReduce program includes Map () procedure that performs filtering and sorting (such as sorting students by name in rows, one row for each name) and Reduce () procedure that performs the operation of aggregation (for example, the number of students in each row). MapReduce system manages the distributed servers, and generally the whole process. The system performs different tasks simultaneously, manages all communications as well as

the transfer of data between different parts of the system, at the same time ensuring a system against redundancy and errors. MapReduce libraries are written in different programming languages. In-Memory machine provide high-performance memory analytics processing. Free implementation is the popular Apache Hadoop organizations.

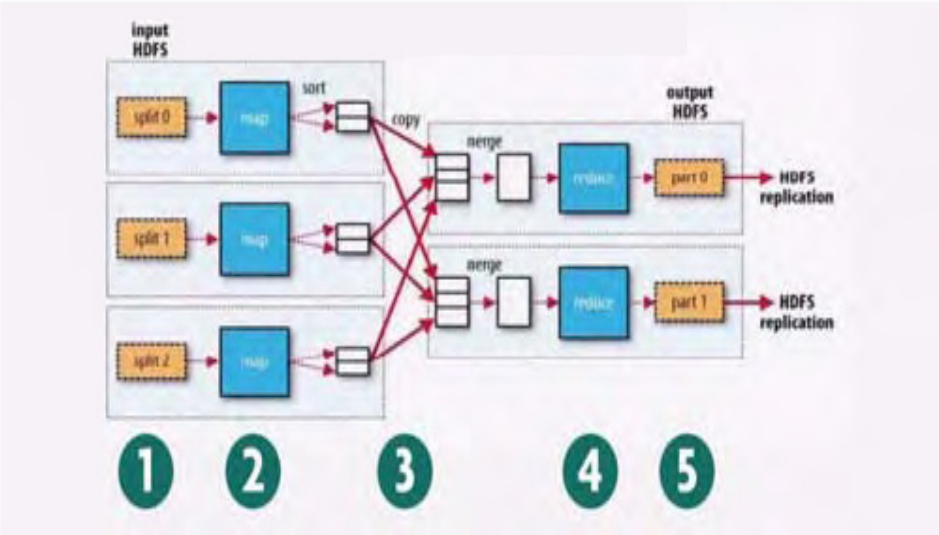

Figure 2. MapReduce workflow

The MapReduce workflow for such a word count function would follow the steps as shown in the diagram below, Figure 2:

1. The system takes input from a file system and splits it up across separate Map nodes

2. The Map function or code is run and generates an output for each Map node-in the word count function, every word is listed and grouped by word per node

3. This output represents a set of intermediate key-value pairs that are moved to Reduce nodes as input

4. The Reduce function or code is run and generates an output for each Reduce node-in the word count example, the reduce function sums the number of times a group of words or key occurs 5. The system takes the outputs from each node to aggregate a final view.

The reduction starts when the data is copied from the mapping phase as soon as available. Reduction phase can start only after the mapping phase is completed and the results are collected.

Reduction consists of three main phases:

1.Shuffle: Reduction is used for grouping the outputs from mapping phase. At this stage the system, for each node that runs the reduction, finds all relevant parts of the outputs that nodes are produced in the mapping phase. Finding is performed using HTTP.

2.Sort: The system groups the inputs to be reduced by using their key. This needs to be done because the different nodes that could do mapping, might produce the keys for the same node for reduction process. The steps of mixing and sorting are carried out simultaneously, that is, when acquiring the output they are joined by together.

3. Reduce: At this stage, the method reducer (Object, Iterable, Context) is performed, which calls for each clustered pair <key, value collections>. The output of this phase is usually entered in RecordWriter using TaskInputOutputContext.write (Object, Object).

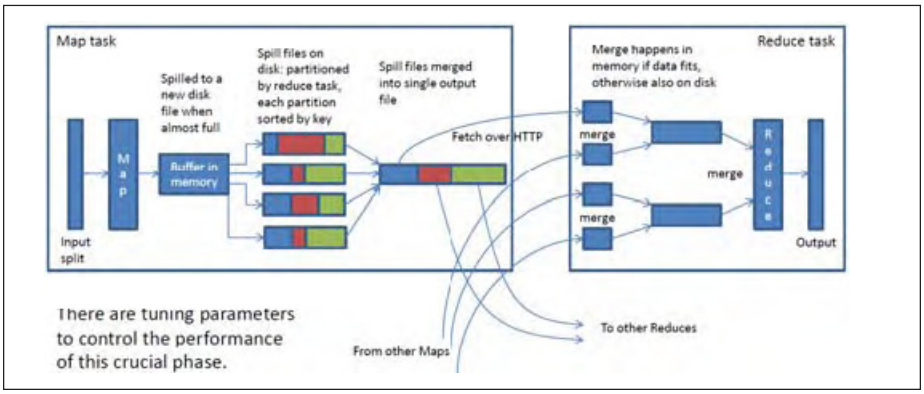

Figure 3. MapReduce tasks

A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks, Figure 3.

There are two types of nodes that control the job execution process: a job-tracker and a number of task-trackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on task-trackers. Task-trackers run tasks and send progress reports to the job-tracker, which keeps a record of the overall progress of each job. If a task fails, the job-tracker can reschedule it on a different task-tracker.

Hadoop divides the input to a MapReduce job into fixedsize pieces called input splits. Hadoop creates one map task for each split, which runs the user defined map function for each record in the split. Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load-balanced if the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine.

Example of the MapReduce process can be seen in Figure 4.

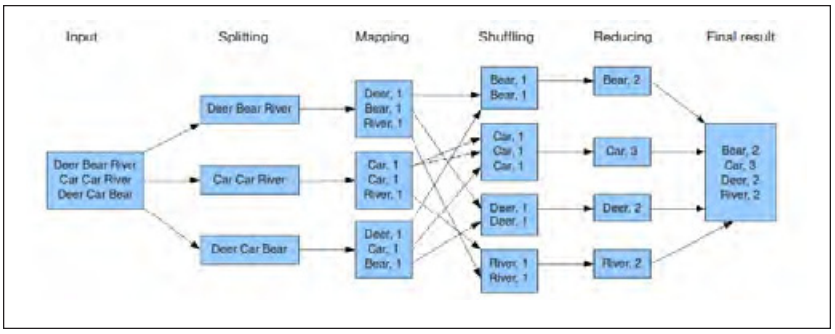

Figure 4. MapReduce example

3. Hadoop Distributed File System

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Even though the Hadoop framework is written in Java, programs for Hadoop need not to be coded in Java but can also be developed in other languages like Python, multi-threaded PHP or C++ .

VM Hadoop consists of the Hadoop Common package, which provides filesystem and OS level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2) and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop.

A typical Hadoop environment consists of several specialized software components: MasterNode, NameNode and Worker Node, Figure 5.

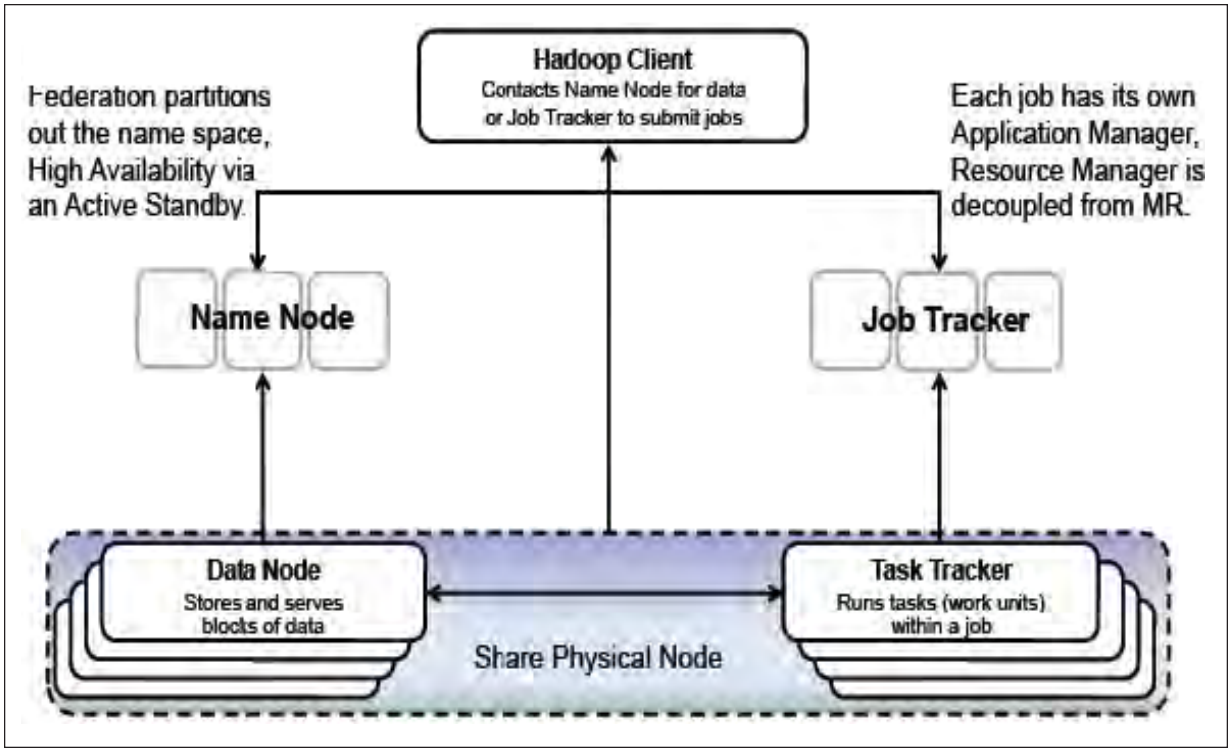

Figure 5. Hadoop environment for better Availability/Scalability

Master node: The majority of Hadoop deployments consist of several master node instances. Having more than one master node helps eliminate the risk of a single point of failure. Here are major elements present in the master node:

JobTracker: This process is assigned to interact with client applications. It is also responsible for distributing MapReduce tasks to particular nodes within a cluster.

TaskTracker: This is a process in the cluster that is capable of receiving tasks (including Map, Reduce, and Shuffle) from a JobTracker.

NameNode: These processes are charged with storing a directory tree of all files in the Hadoop Distributed File System (HDFS). They also keep track of where the file data is kept within the cluster. Client applications contact NameNodes when they need to locate a file, or add, copy, or delete a file.

DataNodes: The DataNode stores data in the HDFS, and is responsible for replicating data across clusters. DataNodes interact with client applications when the NameNode has supplied the DataNode's address.

WorkerNodes: Unlike the master node, whose numbers you can usually count on one hand, a representative Hadoop deployment consists of dozens or even hundreds of worker nodes, which provide enough processing power to analyze a few hundred terabytes all the way up to one petabyte. Each worker node includes a DataNode as well as a TaskTracker.

A Hadoop cluster has nominally a single name-node plus a cluster of data-nodes, although redundancy options are available for the name-node due to its criticality. Each data node serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses TCP/IP sockets for communication. Clients use remote procedure call RPC) to communicate between each other. Figure 6 shows how Hadoop runs a MapReduce job.

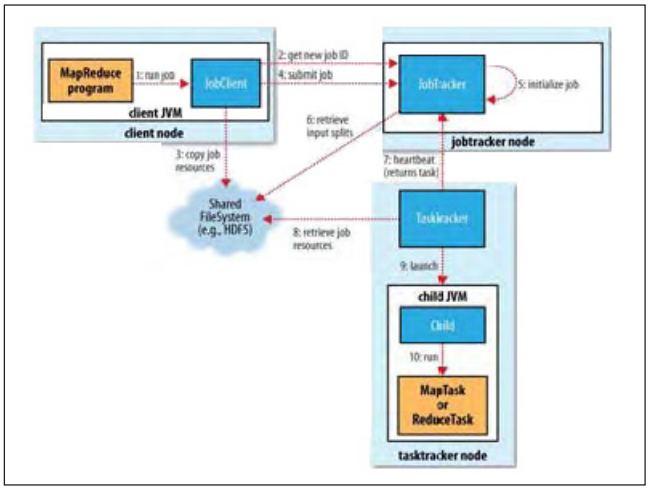

Figure 6. How Hadoop runs a MapReduce job

The job submission process implemented by JobClient's submitJob() method does the following:

MapReduce job is sent to the execution by using the Hadoop client application JobClient. The client application requires major node (JobTracker), a new unique identifier of the transaction, and calculates the partition of the input data. Once the input data is divided into partitions and all the parameters of the transaction have been checked, copied JobClient business components in distributed file system in a directory - the same transaction identifier is generated in the first step. Components of the transaction include the JAR archive with the program itself, with configuration files and partitions of the input data.

After the components of the job became available in a distributed file system, the job is saved in the internal queue Job queue. Job Scheduler then retrieves the job and initializes the tasks required for its execution. Initialized tasks include the Map function (a Map function for each partition of the input data) and Reduce tool (Reduce the number of functions is defined in the configuration file).

Tasks performing is fully orchestrated with main nodes. Before the execution of certain tasks, JobTracker must choose which tasks belong to the work to be performed. Assumed job scheduler chooses what first arrived in the queue. Once you choose a job, JobTracker assigned tasks that make up the selected free job creators of jobs. Tasktrackers run a simple loop that periodically sends heartbeat method calls to the jobtracker. Heartbeats tell the jobtracker that a tasktracker is alive, but they also double as a channel for messages. TaskTracker periodically reports its status main node. The state includes information on free slots for Map and Reduce tasks. Important optimization occurs in assignment to the Map tasks. Map tasks are trying to allocate TaskTracker nodes on which there are data processed by the administration assigned task, thus avoiding the costly network communication, since the data are in the local Map task. In order to optimize the overlap of reading and data processing, framework Hadoop runs more Map and Reduce tasks competitively at the nodes workers.

After the TaskTracker node assigned to the task, it retrieves the JAR archive with the program and launches a separate instance of the Virtual Java Machine (JVM) for performing the assigned task. MapReduce programs can take hours, so workers nodes periodically provide information on the progress of the execution. The job is finished when a TaskTracker which executes the last task in the business told the head node to end the execution of the assigned task.

Performing tasks is subject to hardware or software errors that can manifest in various ways. One advantage of Hadoop framework is to monitor the status of jobs and tasks, handling errors and delays in the work and enabling execution of the transaction in a precarious computer online environment. A mistake by the worker node can cause an exception during the execution of Map / Reduce job or some other error in the JVM system. A common case is the phenomenon of slow workers or workers at a standstill. When the main node receives periodic status message alert workers with errors, it restarts the failed task (avoiding the task start on the same node worker where the error occurred). The master node receives periodic

general messages about the status of work (tasktrackers pool). In addition, the availability of funds in the system is very high using a distributed file system replication blocks (eg. level of replication funds jobs is 10). In this way, apart from the obvious benefits of reducing network communication during the execution of tasks, system reliability and data redundance is increased. Hadoop is not currently considered a mistake in the head node that represents a single point of system failure (single point of failure). One possibility is to introduce redundancy protection using the system to manage the ZooKeeper Internet nodes and determine the primary (main) node.

HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high. HDFS is not fully POSIXcompliant, because the requirements for a POSIX file-system differ from the target goals for a Hadoop application.

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

Hadoop does its best to run the map task on a node where the input data resides in HDFS. This is called the data locality optimization. It should now be clear why the optimal split size is the same as the block size: it is the largest size of input that can be guaranteed to be stored on a single node. If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks, so some of the split would have to be transferred across the network to the node running the map task, which is clearly less efficient than running the whole map task using local data.

4. Conclusion

Unlike Relational database management system - RDBMS, Hadoop MapReduce model exhibits a proportional increase because the Map and Reduce functions do not depend on the size of the input data set, nor the size of the Internet PC network to which the system is running. MapReduce model processes the entire set of data during the execution of queries, while RDBMS systems typically maintains additional data structures (B tree), that speeds up executing queries or updating small amounts of records, but significantly slows down the update of most records in the database. In addition, the MapReduce programming model is designed to handle unstructured (or semi-structured) data such as text, multimedia or binary data.

References

[1] CS246: Mining Massive Datasets, Hadoop Tutorial, January 12, 2016, New York.

[2] A. Hammad, A. García, Hadoop tutorial, Karlsruhe Institut of Technology, SSC, | September, 2014, Germany.

[3] Dhruba Borthakur, HDFS Architecture Guide, Copyright © 2008 The Apache Software Foundation.

[4] Amr Awadallah, Introducing Apache Hadoop: The Modern Data Operating System, Stanford EE380 Computer Systems, 2011, California, USA.

[5] Diana MacLean for CS448G, 2011, A Very Brief Introduction to MapReduce, http://labs.google.com/papers/mapreduce.html

[6] MapReduce by examples, https://github.com/andreaiacono/MapReduce

[7] Ivan Validzi, Primena paralelne obrade u analizi društvenih mre•a, Fakultet elektronike i raunarstva, Zagreb, Croatia, 2015.

[8] Borivoje Miloševi, Danica Miloševi, In Memory baze podataka, Zbornik Radova, Visoka Tehnika 'kola Strukovnih Studija - NI', 2015.