International Journal of Recent Advances in Multidisciplinary Research
Vol. 02, Issue 08, pp.0656-0661, August, 2015

Research Article

HADOOP VS BIG DATA

***Regha, S. and Dr. Manimekalai, M.**

Department of computer Science, Shrimati Indira Gandhi College, Trichy, India

ARTICLE INFO

Article History:
Received 29th May 2015
Received in revised form
12th June, 2015
Accepted 16th July, 2015
Published online 31st August, 2015

Keywords:
Big Data,
Benefits of Big Data,
Traditional Approach,
MapReduce Algorithm,
Hadoop and Hadoop
Distributed File System.

ABSTRACT

Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. The amount of data produced by us from the beginning of time till 2003 was 5 billion gigabytes. If you pile up the data in the form of disks it may fill an entire football field. The same amount was created in every two days in 2011, and in every ten minutes in 2013. This rate is still growing enormously. Though all this information produced is meaningful and can be useful when processed, it is being neglected.90% of the world's data was generated in the last few years. Big data means really a big data, it is a collection of large datasets that cannot be processed using traditional computing techniques. Big data is not merely a data, rather it has become a complete subject, which involves various tools, technqiues and frameworks.Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

INTRODUCTION

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

- **Black Box Data**: It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
- **Social Media Data**: Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
- **Stock Exchange Data**: The stock exchange data holds information about the 'buy' and 'sell' decisions made on a share of different companies made by the customers.
- **Power Grid Data**: The power grid data holds information consumed by a particular node with respect to a base station.
- **Transport Data**: Transport data includes model, capacity, distance and availability of a vehicle.
- **Search Engine Data**: Search engines retrieve lots of data from different databases.

Corresponding author: Regha, S.,
Department of computer Science, Shrimati Indira Gandhi College, Trichy, India.

Thus Big Data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types.

- **Structured data**: Relational data.
- **Semi Structured data**: XML data.
- **Unstructured data**: Word, PDF, Text, Media Logs.

Benefits of Big Data

Big data is really critical to our life and its emerging as one of the most important technologies in modern world. Follow are just few benefits which are very much known to all of us:

- Using the information kept in the social network like Facebook, the marketing agencies are learning about the response for their campaigns, promotions, and other advertising mediums.
- Using the information in the social media like preferences and product perception of their consumers, product companies and retail organizations are planning their production.
- Using the data regarding the previous medical history of patients, hospitals are providing better and quick service.

Big Data Technologies

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in realtime and can protect data privacy and security.

	Operational	Analytical
Latency	1 ms - 100 ms	1 min - 100 min
Concurrency	1000 - 100,000	1 - 10
Access Pattern	Writes and Reads	Reads
Queries	Selective	Unselective
Data Scope	Operational	Retrospective
End User	Customer	Data Scientist
Technology	NoSQL	MapReduce, MPP Database

There are various technologies in the market from different vendors including Amazon, IBM, Microsoft, etc., to handle big data. While looking into the technologies that handle big data, we examine the following two classes of technology:

Operational Big Data

This include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored. NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement. Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding and without the need for data scientists and additional infrastructure.

Analytical Big Data

This includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data. MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines. These two classes of technology are complementary and frequently deployed together.

Operational vs. Analytical Systems

Big Data Challenges

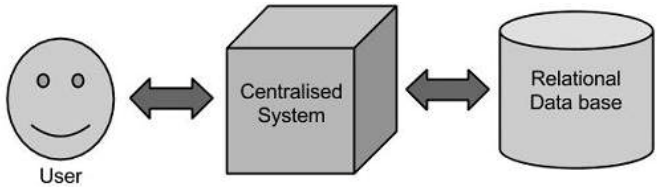
The major challenges associated with big data are as follows:

- Capturing data
- Curation
- Storage
- Searching
- Sharing
- Transfer
- Analysis
- Presentation

To fulfill the above challenges, organizations normally take the help of enterprise servers.

Traditional Approach

In this approach, an enterprise will have a computer to store and process big data. Here data will be stored in an RDBMS like Oracle Database, MS SQL Server or DB2 and sophisticated softwares can be written to interact with the database, process the required data and present it to the users for analysis purpose.

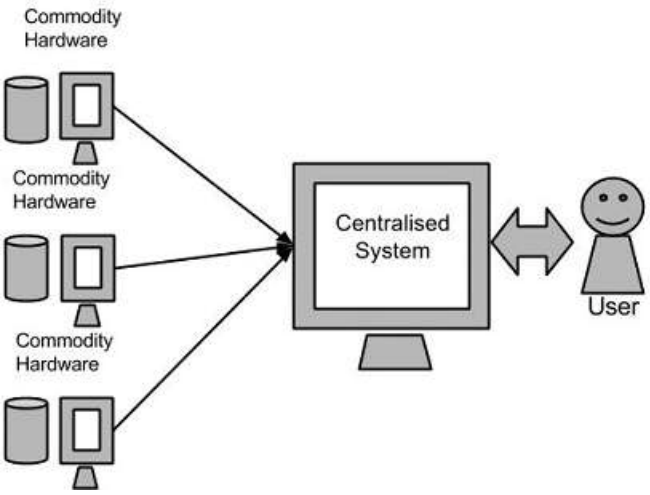


Limitation

This approach works well where we have less volume of data that can be accommodated by standard database servers, or up to the limit of the processor which is processing the data. But when it comes to dealing with huge amounts of data, it is really a tedious task to process such data through a traditional database server.

Google's Solution

Google solved this problem using an algorithm called MapReduce. This algorithm divides the task into small parts and assigns those parts to many computers connected over the network, and collects the results to form the final result dataset. Above diagram shows various commodity hardwares which could be single CPU machines or servers with higher capacity.



MapReduce

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job. The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

  - **Map stage**: The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage**: This stage is the combination of the **Shuffle** stage and the**Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

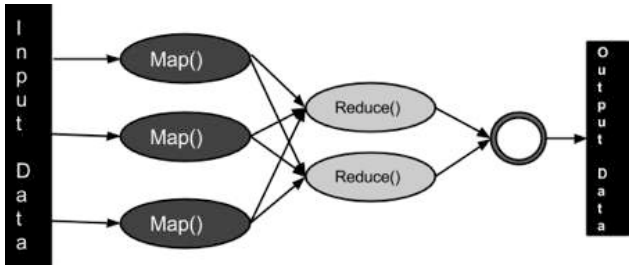
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and Outputs (Java Perspective)

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types. The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface.



Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

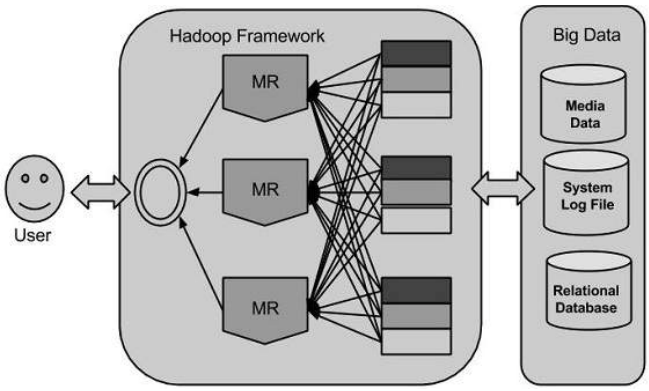
Terminology

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.
- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to JobTracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.

Task Attempt - A particular instance of an attempt to execute a task on a SlaveNode.

Hadoop

Doug Cutting, Mike Cafarella and team took the solution provided by Google and started an Open Source Project called HADOOP in 2005 and Daug named it after his son's toy elephant. Now Apache Hadoop is a registered trademark of the Apache Software Foundation.



Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes.

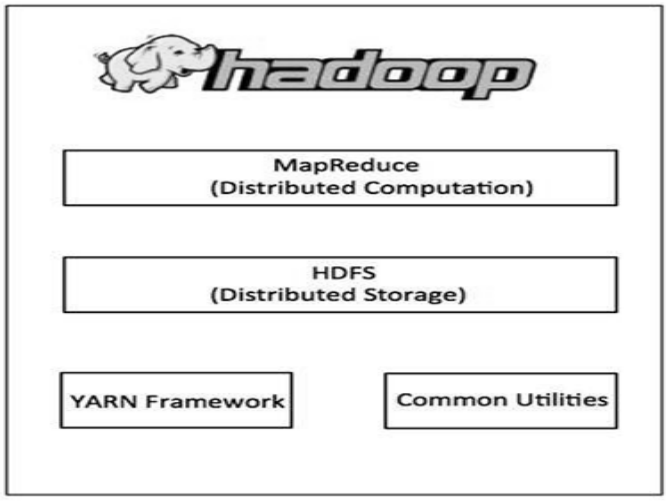
In short, Hadoop framework is capabale enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amounts of data.

Hadoop Architecture

Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
- **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
- **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.

We can use following diagram to depict these four components available in Hadoop framework.



Since 2012, the term "Hadoop" often refers not just to the base modules mentioned above but also to the collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Spark etc.

How Does Hadoop Work?

Stage 1

A user/application can submit a job to the Hadoop (a hadoop job client) for required process by specifying the following items:

- The location of the input and output files in the distributed file system.
- The java classes in the form of jar file containing the implementation of map and reduce functions.
- The job configuration by setting different parameters specific to the job.

Stage 2

The Hadoop job client then submits the job (jar/executable etc) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Stage 3

The Task Trackers on different nodes execute the task as per Map Reduce implementation and output of the reduce function is stored into the output files on the file system. Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS). The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.
A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode. HDFS provides a shell like any other file system and a list of commands are available to interact with the file system. These shell commands will be covered in a separate chapter along with appropriate examples.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

Given below is the architecture of a Hadoop File System.
HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

- **Fault detection and recovery**: Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets**: HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data**: A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

Conclusion

As data engineering and data analytics advances to a next level, Big data testing is inevitable. Big data processing could be Batch, Real-Time, or Interactive 3 stages of Testing Big Data applications are Data staging validation ,"MapReduce" validation,Output validation phase. Architecture Testing is the important phase of Big data testing, as poorly designed system may lead to unprecedented errors and degradation of performance.

Performance testing for Big data includes verifying Data throughput, Data processing and Sub-component performance Big data testing is very different from Traditional data testing in terms of Data, Infrastructure & Validation Tools. Big Data Testing challenges include virtualization, test automation and dealing with large dataset.

Performance testing of Big Data applications is also an issue. Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capabale enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amounts of data. Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores. Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer. Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption. Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Acknowledgments

We would like to thank all students and Professors of the seminars of Big Data for various University, summer term 2012 onwards, for the fruitful discussions. We would like to thank the entire Hadoop++/HAIL team for their feedback and support.

REFERENCES

Abadi, D. *et al*. 2009. Column-Oriented Database Systems. PVDLB, 2(2):1664–1665.

Afrati, F. N. and. Ullman, J. D 2010. Optimizing Joins in a Map-Reduce Environment. In EDBT, pages 99–110.

Babu, S. 2010. Towards automatic optimization of MapReduce programs. In SOCC, pages 137–142.

Blanas *et al*. S. 2010. A Comparison of Join Algorithms for Log Processing in MapReduce. In SIGMOD, pages 975–986.

Dean, J. and Ghemawat, S. 2010. MapReduce: A Flexible Data Processing Tool. CACM, 53(1):72–77.

Dittrich, J., Quiane-Ruiz, J.A., Jindal, A., Kargin, Y., Setty, V. and Schad, J. 2010. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). PVLDB, 3(1):519–529.

Dittrich, J., Quiane-Ruiz, S., Richter, S., Schuh, A., Jindal, and ´ Schad, J. 2012. Only Aggressive Elephants are Fast Elephants. PVLDB, 5.

Floratou, A. *et al*. 2011. Column-Oriented Storage Techniques for MapReduce. PVLDB, 4(7):419–429.

Gates, A. *et al*. 2009. Building a HighLevel Dataflow System on Top of MapReduce: The Pig Experience. PVLDB, 2(2):1414–1425.

Ghemawat, S., Gobioff, H. and Leung, S.T. 2003. The Google file system. In SOSP, pages 29–43.

Hadoop, http://hadoop.apache.org/mapreduce/.

Herodotou, H. and Babu, S. 2011. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. PVLDB, 4(11):1111–1122.

Isard, M. *et al*. 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In EuroSys, pages 59–72.

Jahani, E., Cafarella, M. J. and C. Re. 2011. Automatic Optimization for ´ MapReduce Programs. PVLDB, 4(6):385–396.

Jiang, *et al*. D. 2010. The Performance of MapReduce: An In-depth Study. PVLDB, 3(1-2):472–483.

****