Unit-1
Apache Hadoop

Prepared By

Prof. Rajkumar Chalse

What is Hadoop ?

Open source **software framework** designed for **storage** and **processing** of **large scale data** on clusters of commodity hardware

What is Hadoop?

- A free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment
- Based on Google File System (GFS)

Why Hadoop?

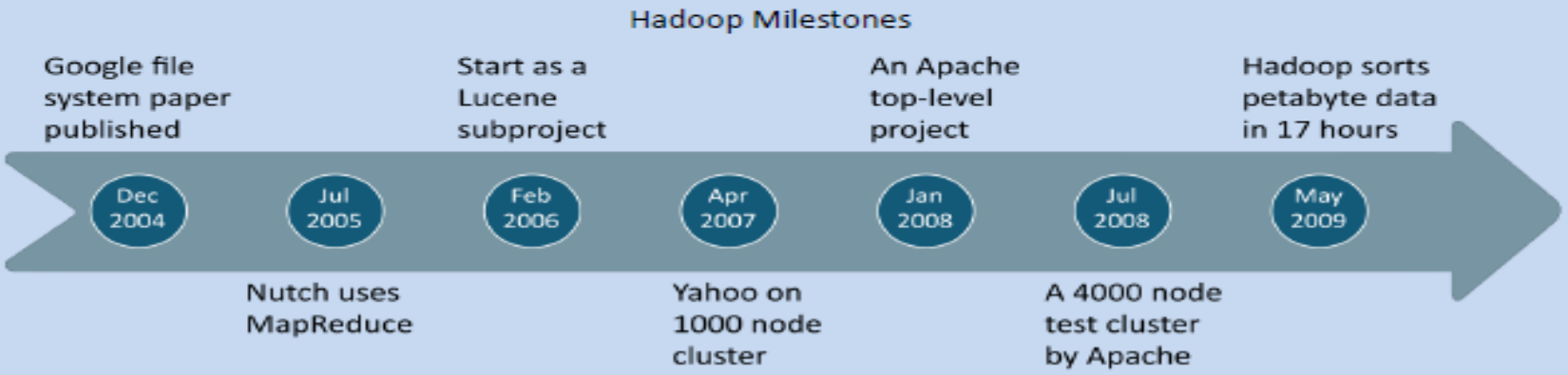
- Runs a number of applications on distributed systems with thousands of nodes involving petabytes of data
- Has a distributed file system, called Hadoop Distributed File System or HDFS, which enables fast data transfer among the nodes

History of Hadoop

- Hadoop came from the **Google File System** paper published in **October 2003**.
- Hadoop was created by **Doug Cutting** and was first released in **April 2006**.
- Cutting named it after his **son's toy elephant**.
- **Elephant** symbolically **represents large data** that can be stored and processed by Hadoop.

History and Milestones of Hadoop

Hadoop originated from the Nutch open source project on search engines and works over distributed network nodes.



Biggest Users Hadoop

Why not use RDBMS ?

- Scalability is an issue especially in RDBMS when data is in TB.
- Databases are designed for structured data only whereas >85% of Big Data is unstructured.
- As data get bigger indexes need to be changed, queries need to be optimized etc to achieve minor improvements .
- We can't add more computing power to increase performance, i.e horizontal scalability is not possible.
- Hardware cost and Database License cost are quite expensive.

What our solution must have

- Handle high volume of data.
- Data loss should be avoided.
- Horizontally scalable.
- Cost effective.
- Easy to work on(even for non programmers or new programmers).

Hadoop meets all of there requirements

- As data grows more nodes can be added seamlessly.
- Works faster than RDBMS.
- Data replication prevents data loss.
- Hadoop is cost effective, Commodity hardware can be used no specialized hardware is needed.
- Hadoop is open source and freely available so no licensing fee.

RDBMS vs Apache Hadoop

- Hadoop is useful when dealing with data in PB while RDBMS is good for data in GB.
- Hadoop supports dynamic schema while RDBMS is best suited for static schema.
- RDBMS is vertically scalable i.e we can improve resource and modify queries that won't speed up the performance that much since you cannot distribute the problem into a number of nodes.
- Hadoop enables horizontal scaling i.e the more nodes we add the more quickly the problem is solved.
- Hadoop allows use of commodity hardware while RDBMS needs specialized hardware.
- Hadoop is more used for batch queries while RDBMS is for batch interactive.
- Hadoop is write once read many times and RDBMS is read and write many times.

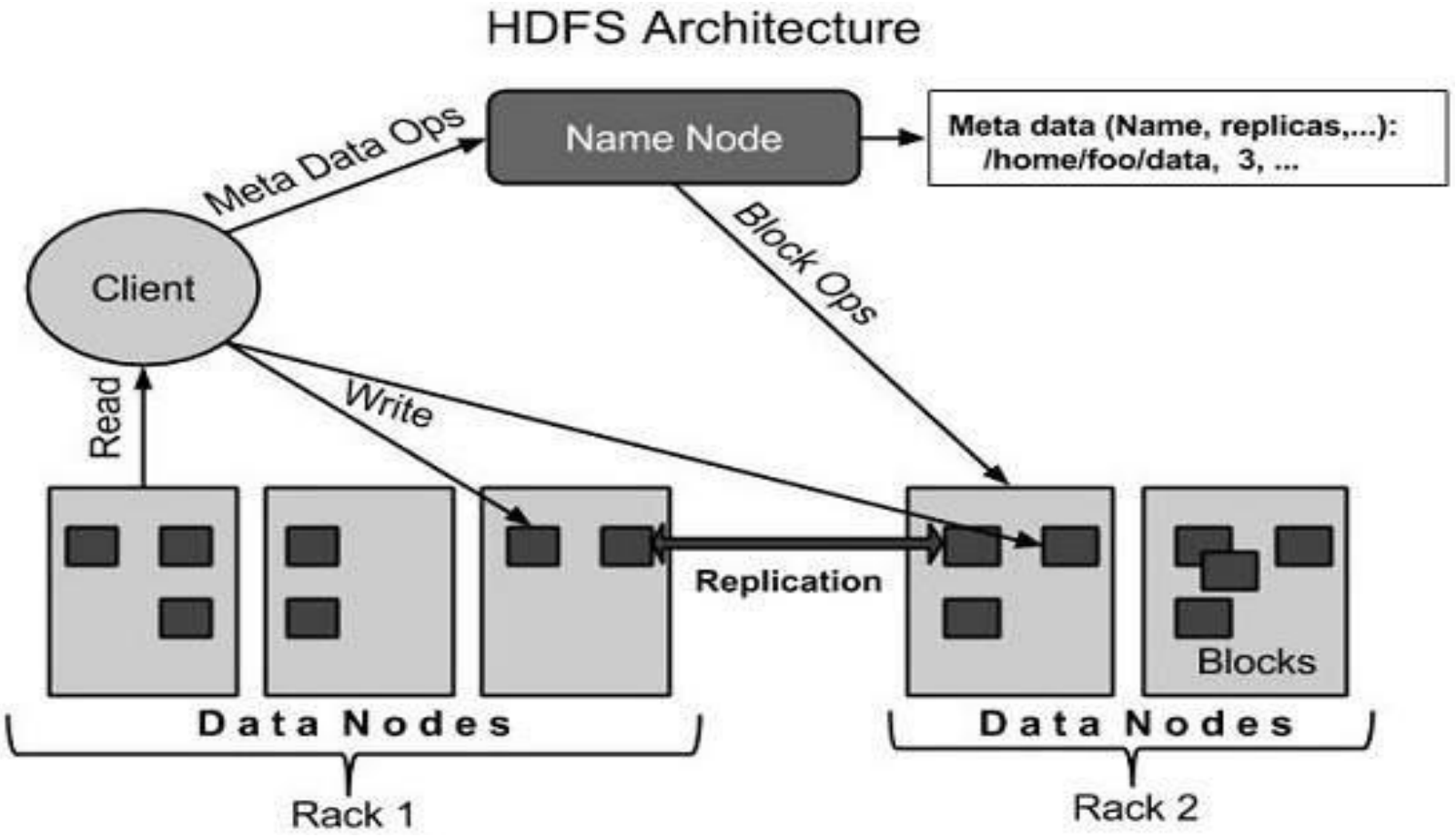
Hadoop Architecture

- The base Apache Hadoop framework is composed of the following modules:
- **Hadoop Distributed File System (HDFS) –** a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- **Hadoop MapReduce –** an implementation of the MapReduce programming model for large scale data processing.
- **Hadoop YARN –** a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and
- **Hadoop Common –** contains libraries and utilities needed by other Hadoop modules;

HDFS

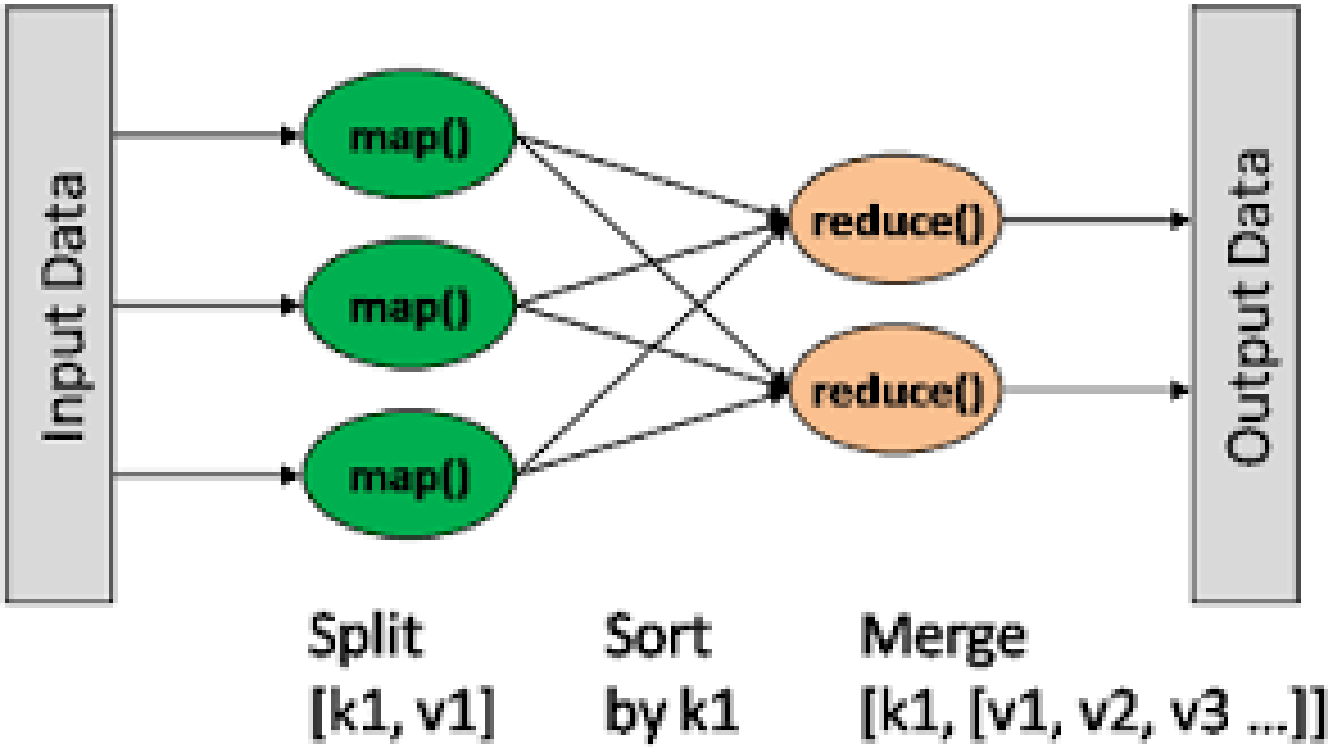
- The Hadoop Distributed File System (HDFS) is the underlying file system of a Hadoop cluster. It provides scalable, fault-tolerant, rack-aware data storage designed to be deployed on commodity hardware. Several attributes set HDFS apart from other distributed file systems. Among them, some of the key differentiators are that HDFS is:
- designed with hardware failure in mind
- built for large datasets, with a default block size of 128 MB
- optimized for sequential operations
- rack-aware
- cross-platform and supports heterogeneous clusters

- Data in a Hadoop cluster is broken down into smaller units (called blocks) and distributed throughout the cluster.
- Each block is duplicated twice (for a total of three copies), with the two replicas stored on two nodes in a rack somewhere else in the cluster. Since the data has a default replication factor of three, it is highly available and fault-tolerant.
- If a copy is lost (because of machine failure, for example), HDFS will automatically re-replicate it elsewhere in the cluster, ensuring that the threefold replication factor is maintained.


HDFS Architecture
Name Node
Meta data (Name, replicas,...):
/home/foo/data, 3, ...
Meta Data Ops
Block Ops
Client
Read
Write
Replication
Blocks
Data Nodes
Data Nodes
Rack 1
Rack 2

Map Reduce

- MapReduce is a framework for processing highly distributable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster. The framework is inspired by the map and reduce functions commonly used in functional programming.
- In the **“Map”** step, the master node takes the input, partitions it up into **smaller sub-problems**, and distributes them to worker nodes. The **worker node processes the smaller problem, and passes the answer back to its master node**. In the **“Reduce”** step, the **master node then collects the answers** to all the **sub-problems and combines them** in some way to form the output – the answer to the problem it was originally trying to solve.


Input Data
map()
map()
map()
reduce()
reduce()
Output Data
Split
[k1, v1]
Sort
by k1
Merge
[k1, [v1, v2, v3 ...]]

Namenode

- NameNode is the centerpiece of HDFS.
- NameNode is also known as the Master
- NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster.
- NameNode does not store the actual data or the dataset. The data itself is actually stored in the DataNodes.
- NameNode knows the list of the blocks and its location for any given file in HDFS. With this information NameNode knows how to construct the file from blocks.
- NameNode is so critical to HDFS and when the NameNode is down, HDFS/Hadoop cluster is inaccessible and considered down.
- NameNode is a single point of failure in Hadoop cluster.
- NameNode is usually configured with a lot of memory (RAM). Because the block locations are help in main memory.

Typical Namenode Configuration

- Processors: 2 Quad Core CPUs running @ 2 GHz
- RAM: 128 GB
- Disk: 6 x 1TB SATA
- Network: 10 Gigabit Ethernet

Datanode

- DataNode is responsible for storing the actual data in HDFS.
- DataNode is also known as the Slave
- NameNode and DataNode are in constant communication.
- When a DataNode starts up it announce itself to the NameNode along with the list of blocks it is responsible for.
- When a DataNode is down, it does not affect the availability of data or the cluster. NameNode will arrange for replication for the blocks managed by the DataNode that is not available.
- DataNode is usually configured with a lot of hard disk space. Because the actual data is stored in the DataNode.

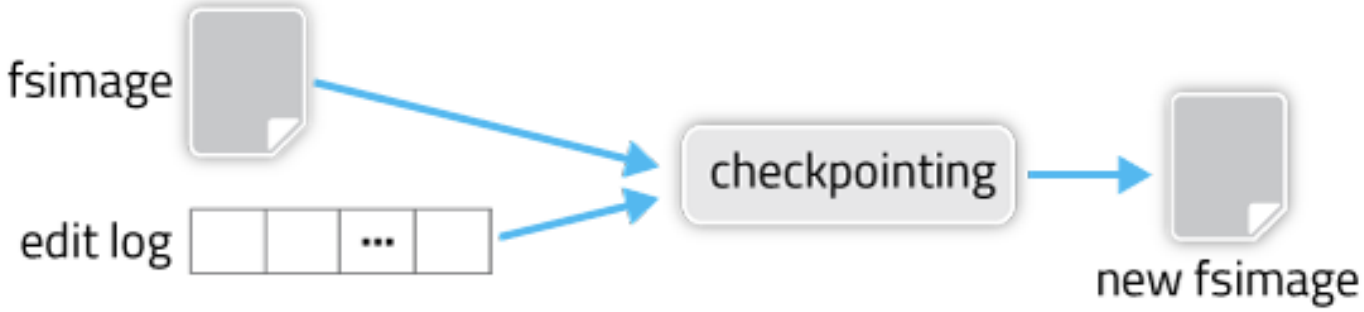
Typical Datanode Configuration

- Processors: 2 Quad Core CPUs running @ 2 GHz
- RAM: 64 GB
- Disk: 12-24 x 1TB SATA
- Network: 10 Gigabit Ethernet

Checkpointing

- HDFS metadata can be thought of consisting of two parts: the base filesystem table (stored in a file called **fsimage**) and the edit log which lists changes made to the base table (stored in a file called **edits**).
- Checkpointing is a process of reconciling fsimage with edits to produce a new version of fsimage.
- There are two benefits arising out of this: a more **recent version of fsimage**, and **a truncated edit log**.

- **fsimage** is like a **snapshot of** the state of the **filesystem as at particular moment** whereas the **edit log is a list of changes** to the **filesystem** state since then.
- When the **namenode starts up** it reads the **fsimage file as its starting point** and then **applies any edits** from the log and then starts listening to data nodes for details of where the blocks of data reside (this information is not stored in fsimage or the edit log but rather is maintained in the memory of the name node).


fsimage
edit log
checkpointing
new fsimage

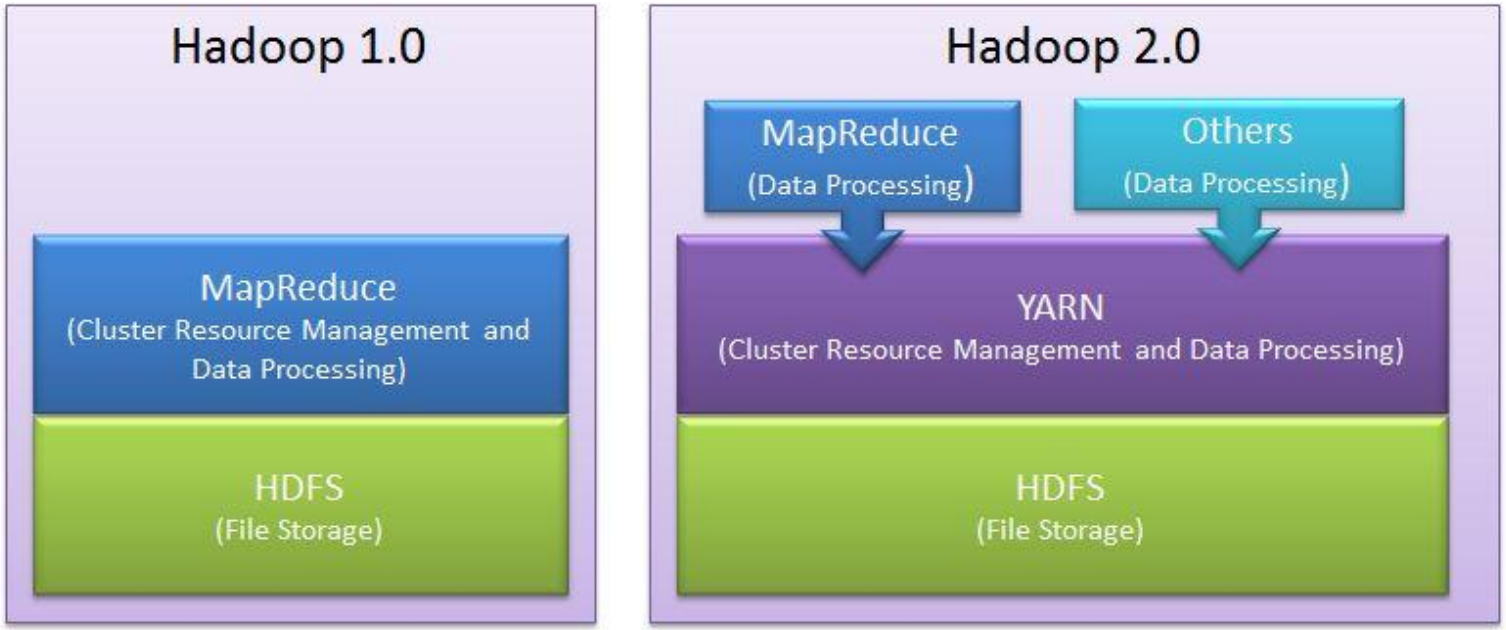
Secondary Namenode

- Secondary Namenode does not serve as a backup namenode but it only periodically reads file system changes log and ap-plies them to fsimage file in order to bring the system upto date. This allows the system to start faster next time.

YARN

- Stands for Yet Another Resource Negotiator and was introduced in Hadoop 2.0.
- In Yarn, the job tracker is split into two different daemons called Resource Manager and Node Manager (node specific). The resource manager only manages the allocation of resources to the different jobs apart from comprising a scheduler which just takes care of the scheduling jobs without worrying about any monitoring or status updates. Different resources such as memory, cpu time, network bandwidth etc. are put into one unit called the Resource Container. There are different AppMasters running on different nodes which talk to a number of these resource containers and accordingly update the Node Manager with the monitoring/status details.

YARN is the most important part of Hadoop 2.0



Problems with Hadoop 1.0

- **It limits scalability:** JobTracker runs on single machine doing several task like
- ->Resource management
- ->Job and task scheduling and
- ->Monitoring
- Although there are so many machines (DataNode) available; they are not getting used. This limits scalability.
- **Availability Issue:** In Hadoop 1.0, JobTracker is single Point of availability. This means if JobTracker fails, all jobs must restart.

- **Problem with Resource Utilization:** In Hadoop 1.0, there is concept of predefined number of map slots and reduce slots for each TaskTrackers. Resource Utilization issues occur because maps slots might be ‘full’ while reduce slots is empty (and vice-versa). Here the compute resources (DataNode) could sit idle which are reserved for Reduce slots even when there is immediate need for those resources to be used as Mapper slots.
- **Limitation in running non-MapReduce Application:** In Hadoop 1.0, Job tracker was tightly integrated with MapReduce and only supporting application that obeys MapReduce programming framework can run on Hadoop.

Advantages of YARN

- **Yarn does efficient utilization of the resource.** There are no more fixed map-reduce slots. YARN provides central resource manager. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource.
- **Yarn can even run application that do not follow MapReduce model.**
- YARN decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. This also streamlines MapReduce to do what is does best - process data.

- **YARN is backward compatible.** This means that existing MapReduce job can run on Hadoop 2.0 without any change.

- **No more JobTracker and TaskTracker needed in Hadoop 2.0**
- **JobTracker and TaskTracker has totally disappeared**. YARN splits the two major functionalities of the JobTracker i.e. resource management and job scheduling/monitoring into 2 separate daemons (components).
- ->Resource Manager
- ->Node Manager(node specific)

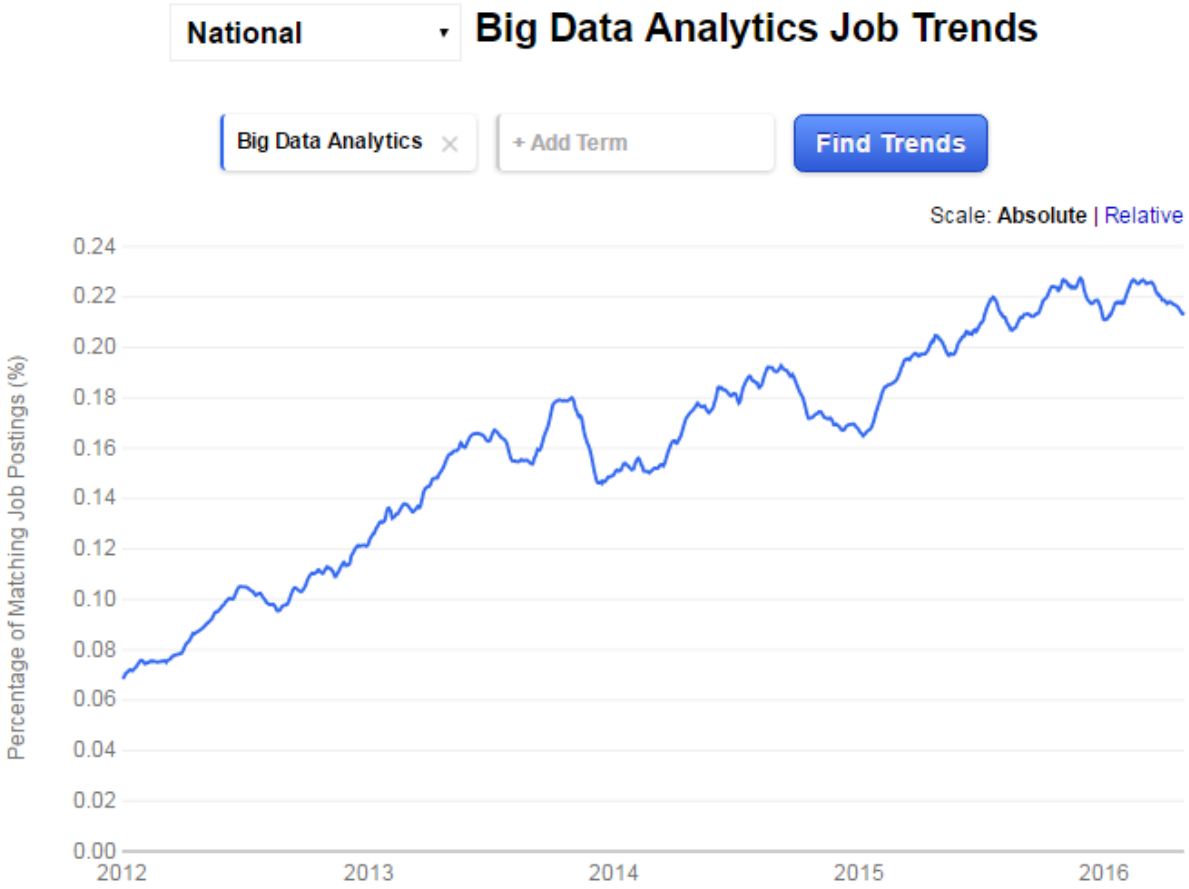
Limitations of Hadoop

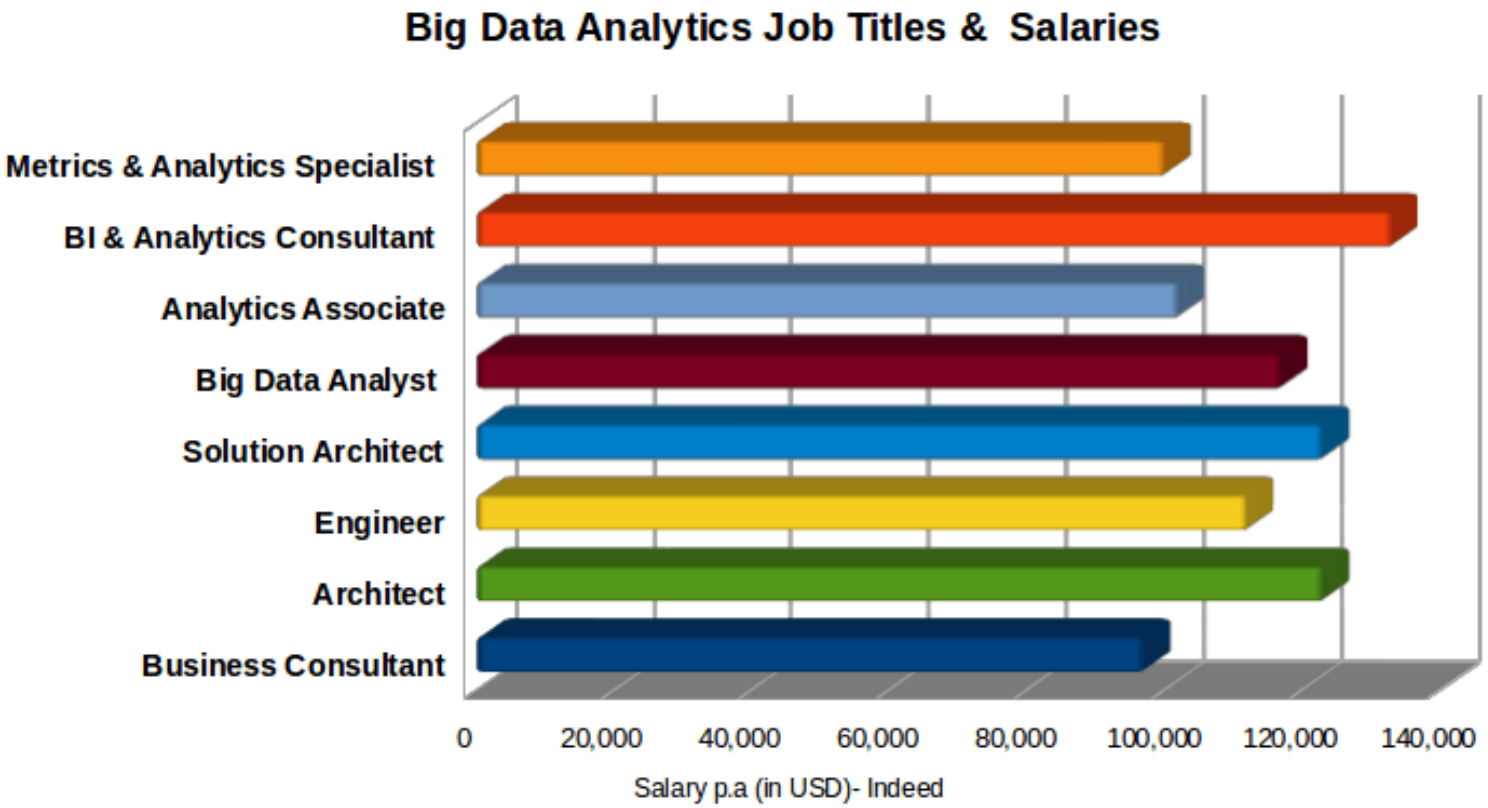
- **Security Concerns.** Just managing a complex application such as Hadoop can be challenging. A classic example can be seen in the Hadoop security model, which is disabled by default due to sheer complexity. If whoever's managing the platform lacks the know how to enable it, your data could be at huge risk. Hadoop is also missing encryption at the storage and network levels, which is a major selling point for government agencies and others that prefer to keep their data under wraps.
- **Vulnerable By Nature.** Speaking of security, the very makeup of Hadoop makes running it a risky proposition. The framework is written almost entirely in Java, one of the most widely used yet controversial programming languages in existence. Java has been heavily exploited by cybercriminals and as a result, implicated in numerous security breaches. For this reason, several experts have suggested dumping it in favor of safer, more efficient alternatives.
- **Not Fit for Small Data.** While big data isn't exclusively made for big businesses, not all big data platforms are suited for small data needs. Unfortunately, Hadoop happens to be one of them. Due to its high capacity design, the Hadoop Distributed File System or HDFS, lacks the ability to efficiently support the random reading of small files. As a result, it is not recommended for organizations with small quantities of data.

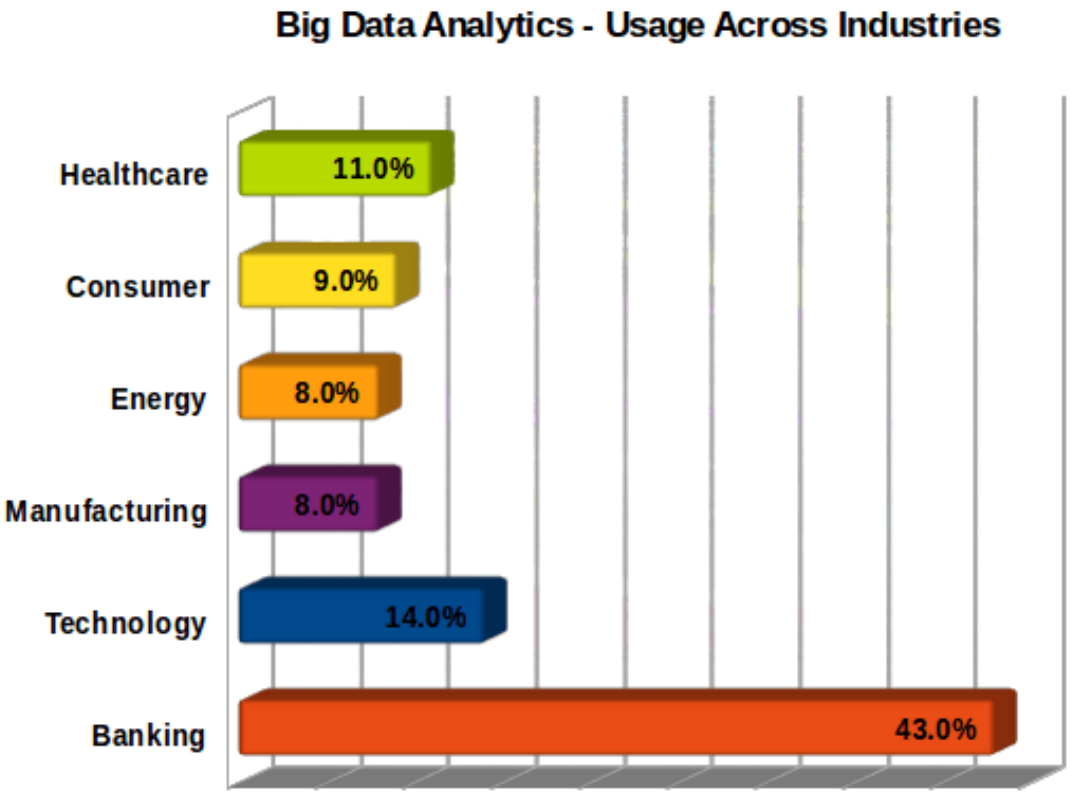
- **Potential Stability Issues.** Hadoop is an open source platform. That essentially means it is created by the contributions of the many developers who continue to work on the project. While improvements are constantly being made,like all open source software, Hadoop has had its fair share of stability issues. To avoid these issues, organizations are strongly recommended to make sure they are running the latest stable version, or run it under a third-party vendor equipped to handle such problems.
- **General Limitations.** When it comes to making the most of big data, Hadoop may not be the only answer. Apache Flume, Mill-wheel, and Google's own Cloud Data-flow as possible solutions. What each of these platforms have in common is the ability to improve the efficiency and reliability of data collection, aggregation, and integration.
- **Multiple copies of already big data.** Because HDFS was built without the notion of efficiency, it results in multiple copies of the data. At a minimum, there are generally three copies of the data. And because of the need for data locality in maintaining performance, we very often see six copies of the data required and that's for data that's already "big" by definition.

- **Very limited SQL support.** There are open source components which attempt to set up Hadoop as a queryable data warehouse, but these offer very limited SQL support. Typically they lack such basic SQL functions such as subqueries, ‘group by’ analytics, etc.
- **Inefficient execution.** HDFS has no notion of a query optimizer, so cannot pick an efficient cost-based plan for execution. Because of this, Hadoop clusters are generally significantly larger than would be required for a similar database.
- **Challenging framework.** The MapReduce framework is notoriously difficult to leverage for more than simple transformational logic. There are open source components which attempt to simplify this, but they also use proprietary languages.

Demand of Big Data Professionals




Big Data Analytics Job Titles & Salaries
Metrics & Analytics Specialist
BI & Analytics Consultant
Analytics Associate
Big Data Analyst
Solution Architect
Engineer
Architect
Business Consultant
0
20,000
40,000
60,000
80,000
100,000
120,000
140,000
Salary p.a (in USD)- Indeed



Source: Peer Research – Big Data Analytics Survey