Big Data Journey

Quote: -

In the pioneer days, they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers. —Grace Hopper

UNITS OF DATA STORAGE

Name of Data Size	Equivalent	Approximate Size
1 Bit	1 Bit	Logic 1 or Logic 0
1 Byte	8 Bits	Single Character
1 KB (Kilo Byte)	1024 Byte	Single page data
1 MB (Mega Byte)	1024 KB	1 High Quality Photograph
1 GB (Giga Byte)	1024 MB	Single Movie [2 H 30 M duration]
1 TB (Tera Byte)	1024 GB	100 Films [Each 3 Hour Duration]
1 PB (Peta Byte)	1024 TB	0.7 Million CD's (OR) 500 Books[Each 1 Billion Pages]
1 EB (Exa Byte)	1024 PB	Single day data stored in internet on entire globe in 2012
1 ZB (Zetta Byte)	1024 EB	25 Billion Pen Drives [Each 64 GB]
1 YB (Yotta Byte)	1024 ZB	1 YB Memory Chip design Cost -$ 2500 Million (OR) 11 Trillion Years for Download 1 YB Data
1 BB (Bronto Byte)	1024 YB	1 followed by 27 Zeros
1 Geop Byte	1024 BB	Extremely Tremendous Sophisticated Memory Size

Data!

- ✓ We live in the data age. It's not easy to measure the total volume of data stored electronically, but an IDC estimate put the size of the "digital universe" at 4.4 zettabytes in 2013 and is forecasting a tenfold growth by 2020 to 44 zettabytes.1 A zettabyte is 1021 bytes, or equivalently one thousand exabytes, one million petabytes, or one billion terabytes. That's more than one disk drive for every person in the world. This flood of data is coming from many sources.

- ✓ Consider the following:
  - The New York Stock Exchange generates about 4−5 terabytes of data per day.
  - Facebook hosts more than 240 billion photos, growing at 7 petabytes per month.
  - Ancestry.com, the genealogy site, stores around 10 petabytes of data.
  - The Internet Archive stores around 18.5 petabytes of data.
  - The Large Hadron Collider near Geneva, Switzerland, produces about 30 petabytes of data per year.

- ✓ The trend is for every individual's data footprint to grow, but perhaps more significantly, the amount of data generated by machines as a part of the Internet of Things will be even greater than that generated by people. Machine logs, RFID readers, sensor networks, vehicle GPS traces, retail transactions—all of these contribute to the growing mountain of data. The volume of data being made publicly available increases every year, too.

- ✓ Organizations no longer must merely manage their data; success in the future will be dictated to a large extent by their ability to extract value from other organizations' data.

- ✓ It has been said that "more data usually beats better algorithms," which is to say that for some problems (such as recommending movies or music based on past preferences), however fiendish your algorithms, often they can be beaten simply by having more data (and a less sophisticated algorithm).3 The good news is that big data is here. The bad news is that we are struggling to store and analyze it.

Problem: -Data Storage and Analysis

- ✓ The problem is simple: although the storage capacities of hard drives have increased massively over the years, access speeds—the rate at which data can be read from drives—have not kept up. One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s,4 so you could read all the data from a full drive in around five minutes. Over 20 years later, 1-terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data on the disk.
- ✓ Using only one-hundredth of a disk may seem wasteful. But we can store 100 datasets, each of which is 1 terabyte, and provide shared access to them. We can imagine that the users of such a system would be happy to share access in return for shorter analysis times, and statistically, that their analysis jobs would be likely to be spread over time, so they wouldn't interfere with each other too much.

First Problem: - Storage and Hardware Failure

- ✓ There's more to being able to read and write data in parallel to or from multiple disks, though. The first problem to solve is a hardware failure: as soon as you start using many pieces of hardware, the chance that one will fail is high. A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID works, for instance, although Hadoop's filesystem, the Hadoop Distributed Filesystem (HDFS), takes a slightly different approach, as you shall see later.

Second Problem: - Combine processing of data

- ✓ The second problem is that most analysis tasks need to be able to combine the data in some way, and data read from one disk may need to be combined with data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources but doing this correctly is notoriously challenging. MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values. We look at the details of this model in later chapters, but the important point for the present discussion is that there are two parts to the computation—the map and the reduce—and it's the interface be- tween the two where the "mixing" occurs. Like HDFS, MapReduce has built-in reliability.
- ✓ In a nutshell, this is what Hadoop provides: a reliable, scalable platform for storage and analysis. What's more, because it runs on commodity hardware and is open source, Hadoop is affordable.

Comparison with Other Systems

Hadoop isn't the first distributed system for data storage and analysis, but it has some unique properties that set it apart from other systems that may seem similar. Here we look at some of them.

- ✓ **Hadoop Vs RDBMS: -**

- Why can't we use databases with lots of disks to do the large-scale analysis? Why is Hadoop needed?
- The answer to these questions comes from another trend in disk drives: seek time is improving more slowly than the transfer rate. Seeking is the process of moving the disk's head to a particular place on the disk to read or write data. It characterizes the latency of a disk operation, whereas the transfer rate corresponds to a disk's bandwidth.
- If the data access pattern is dominated by seeks, it will take longer to read or write large portions of the dataset than a stream through it, which operates at the transfer rate. On the other hand, for updating a small proportion of records in a database, a traditional B-Tree (the data structure used in relational databases, which is limited by the rate at which it can perform seeks) works well. For updating the majority of a database, a BTree is less efficient than MapReduce, which uses Sort/Merge to rebuild the database.
- In many ways, MapReduce can be seen as a complement to a Relational Database Management System (RDBMS). (The differences between the two systems are shown in Table 1-1.) MapReduce is a good fit for problems that need to analyze the whole dataset in a batch fashion, particularly for ad hoc analysis. An RDBMS is good for point queries or updates, where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data. MapReduce suits applications where the data is written once and read many times, whereas a relational database is good for continually updated datasets.

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Transactions	ACID	None

	Traditional RDBMS	MapReduce
Structure	Schema-on-write	Schema-on-read
Integrity	High	Low
Scaling	Nonlinear	Linear

Schema on Write (SQL RDBMS)

1. Create schema

```
CREATE TABLE Customers (
        Key int, Name varchar(40), ...
) .
```

2. Add data

```
BULK INSERT Customers
FROM 'c:\temp\custfile.txt'
WITH FIELDTERMINATOR = '","'
```

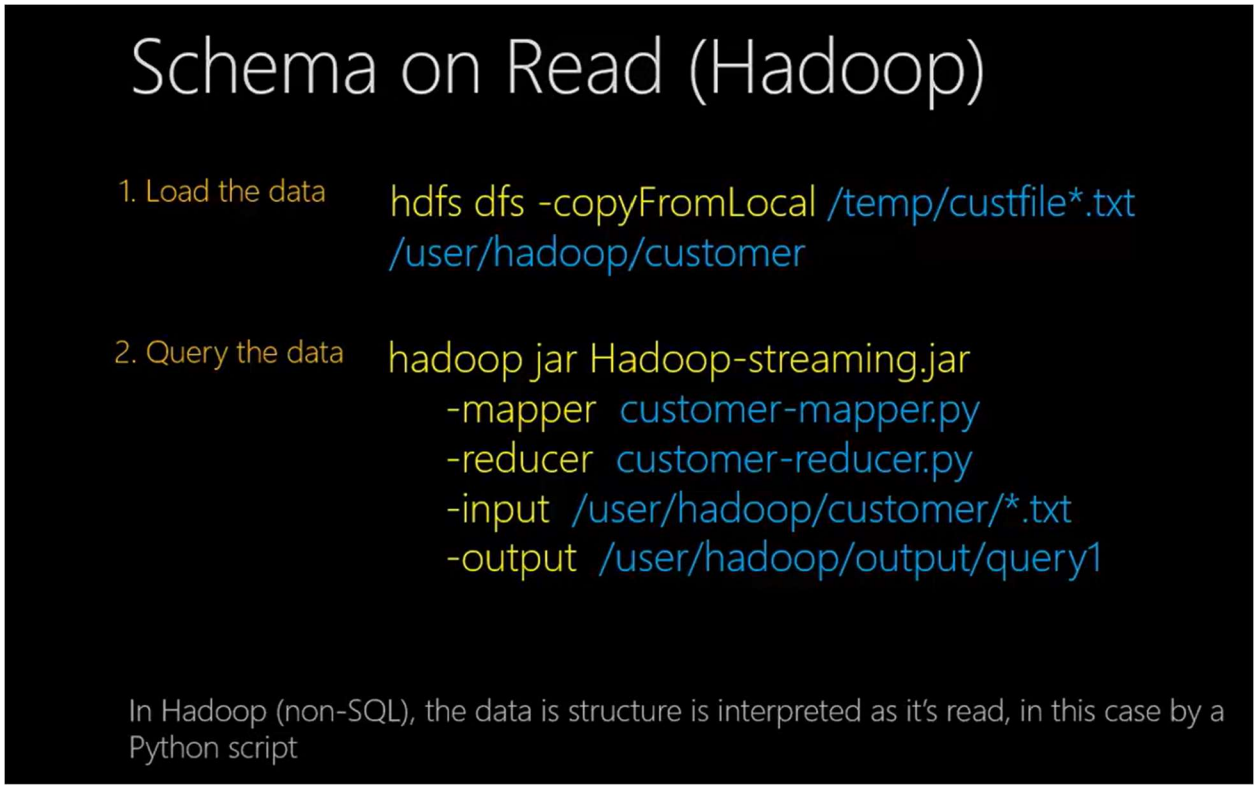
3. Query data

```
SELECT Key , Name FROM Customers
```

In SQL, you can't add data until AFTER the table's schema has been declared

If the schema is to be redefined, the table is dropped and re-loaded

- What are implications when the data volume is 500TB?



However, the differences between relational databases and Hadoop systems are blurring. Relational databases have started incorporating some of the ideas from Hadoop, and from the other direction, Hadoop systems such as Hive are becoming more interactive (by moving away from MapReduce) and adding features like indexes and transactions that make them look more and more like traditional RDBMSs.

Another difference between Hadoop and an RDBMS is the amount of structure in the datasets on which they operate. Structured data is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema. This is the realm of the RDBMS. Semi-structured data, on the other hand, is looser, and though there may be a schema, it is often ignored, so it may be used only as a guide to the structure of the data: for example, a spreadsheet, in which the structure is the grid of cells, although the cells themselves may hold any form of data. Unstructured data does not have any internal structure: for example, plain text or image data. Hadoop works well on unstructured or semi-structured data because it is designed to interpret the data at processing time (so-called schema-on-read). This provides flexibility and avoids the costly data-loading phase of an RDBMS since in Hadoop it is just a file copy.

Relational data is often normalized to retain its integrity and remove redundancy. Normalization poses problems for Hadoop processing because it makes reading a record a nonlocal operation, and one of the central assumptions that Hadoop makes is that it is possible to perform (high-speed) streaming reads and writes.

A web server log is a good example of a set of records that is not normalized (for example, the client hostnames are specified in full each time, even though the same client may appear many times), and this is one reason that logfiles of all kinds are particularly well suited to analysis with Hadoop.

Note that Hadoop can perform joins; it's just that they are not used as much as in the relational world. MapReduce—and the other processing models in Hadoop—scales linearly with the size of the data. Data is partitioned, and the functional primitives (like map and reduce) can work in parallel on separate partitions. This means that if you double the size of the input data, a job will run twice as slowly. But if you also double the size of the cluster, a job will run as fast as the original one. This is not generally true of SQL queries.

Comments: -

Big data is a problem, not a technology. Vs of big data are created terms. So basically, we should remember there are two terms that make big data: - quantity (volume) and processing time. It depends on what your data has for the storage process. Even if one of the things isn't satisfied then it's big data. It depends on the business needs and resources you have.

Evolution Of Hadoop: -

Hadoop History

Hadoop was started by **Doug Cutting and Mike Cafarella** in the year 2002 when they both started to work on the Apache Nutch project. The Apache Nutch project was the process of building a search engine system that can index 1 billion pages. After a lot of research on Nutch, they concluded that such a system will cost around half a million dollars in hardware, and along with a monthly running cost of $30, 000 approximately, which is very expensive. So, they realized that their project architecture will not be capable enough to work around billions of pages on the web. So, they were looking for a feasible solution that can reduce the implementation cost as well as the problem of storing and processing large datasets.

In 2003, they came across a paper that described the architecture of Google's distributed file system, called **GFS (Google File System)** which was published by Google, for storing large data sets. Now they realize that this paper can solve their problem of storing very large files which were being generated because of web crawling and indexing processes. But this paper was just the half solution to their problem.

In 2004, Google published one more paper on the technique **MapReduce**, which was the solution for processing those large datasets. Now this paper was another half solution for Doug Cutting and Mike Cafarella for their Nutch project. These techniques (GFS & MapReduce) were just on white paper at Google. Google didn't implement these two techniques. Doug Cutting knew from his work on Apache Lucene (It is a free and open-source information retrieval software library, originally written in Java by Doug Cutting in 1999) that open-source is a great way to spread the technology to more people. So, together with Mike Cafarella, he started implementing Google's techniques (GFS & MapReduce) as open source in the Apache Nutch project.

In 2005, Cutting found that Nutch is limited to only 20-to-40 node clusters. He soon realized two problems:
(a) Nutch wouldn't achieve its potential until it ran reliably on the larger clusters
(b) And that was looking impossible with just two people (Doug Cutting & Mike Cafarella).

The engineering task in the Nutch project was much bigger than he realized. So he started to find a job with a company that is interested in investing in their efforts. And he found Yahoo! Yahoo had a large team of engineers that were eager to work on this there project.

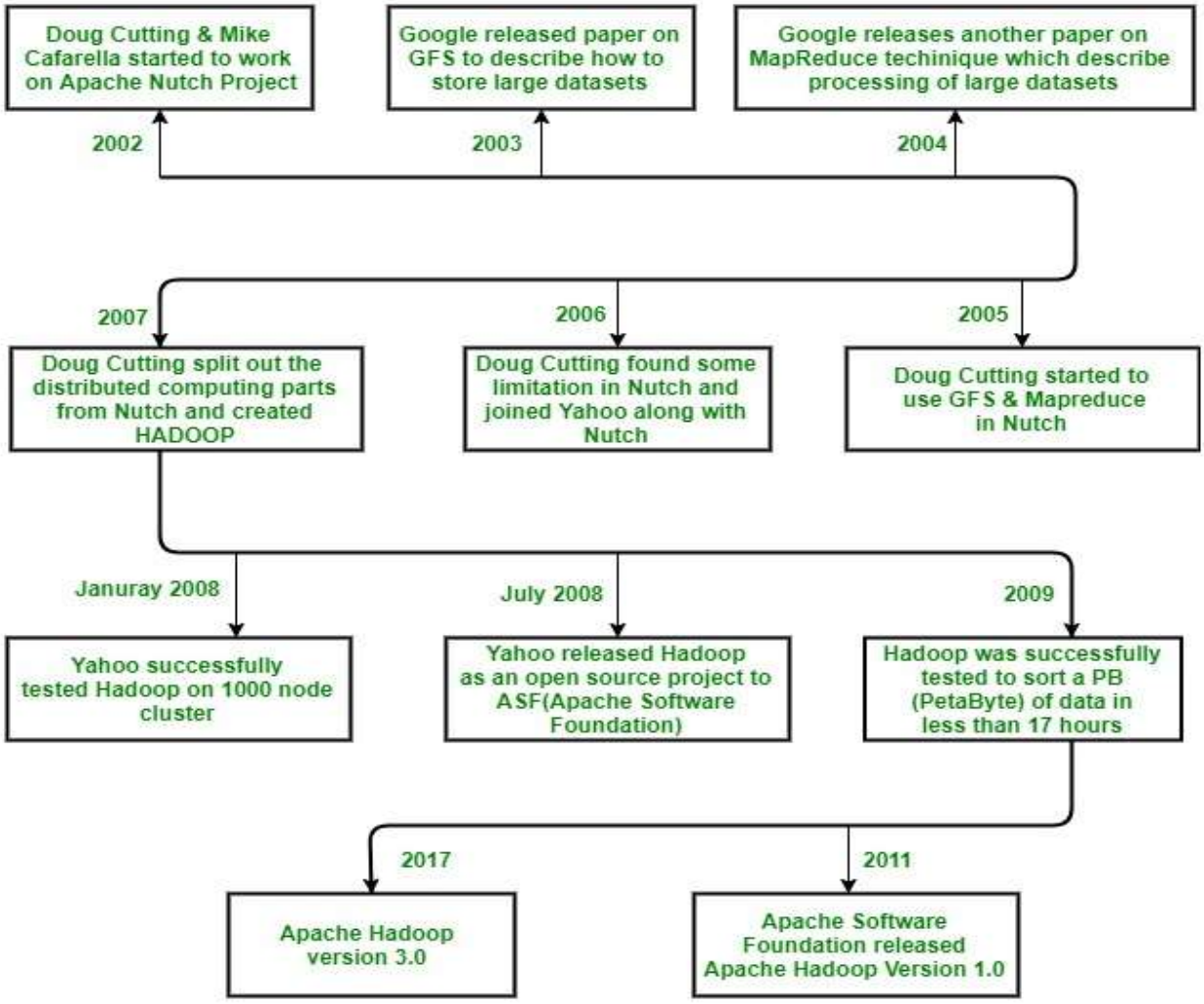
So, **in 2006,** Doug Cutting joined Yahoo along with the Nutch project. He wanted to provide the world with an open-source, reliable, scalable computing framework, with the help of Yahoo. So at Yahoo first, he separates the distributed computing parts from Nutch and **formed a new project Hadoop (He gave the name Hadoop it was the name of a yellow toy elephant that was owned by Doug Cutting's son. and it was easy to pronounce and was the unique word.**) Now he wanted to make Hadoop in such a way that it can work well on thousands of nodes. So with GFS and MapReduce, he started to work on Hadoop.

In 2007, Yahoo successfully tested Hadoop on a 1000-node cluster and start using it.

In January 2008, Yahoo released Hadoop as an open-source project to ASF(Apache Software Foundation). And in **July of 2008, Apache Software Foundation successfully tested a 4000-node cluster with Hadoop.**

In 2009, Hadoop was successfully tested to sort a PB (Petabyte) of data in less than 17 hours for handling billions of searches and indexing millions of web pages. And **Doug Cutting Left Yahoo and joined Cloudera to fulfill the challenge of spreading Hadoop to other industries.**

In **December 2011,** Apache Software Foundation released **Apache Hadoop version 1.0.**
And later in Aug 2013, **Version 2.0.6 was available**.
And currently, we have **Apache Hadoop version 3.0** which was released in **December 2017**.



References

Conceptual Understanding Credit to Suraj G: - Online Learning Center

https://www.youtube.com/watch?v=JfQMHqUq1hM&list=PL2jvR2CemrGopSjBChw8_Hfn10dIGIssX

Theoretical Concept was copied from

White, T. (2015). *Hadoop: The definitive guide*. O'Reilly.

Evolution Of Hadoop: -

https://www.geeksforgeeks.org/hadoop-history-or-evolution/