

Big Data

Measuring data

Bit	1 bit	1/8
Nibble	4 bits	1/2 (rare)
Byte	8 bits	1
Kilobyte	1,024 bytes	1,024
Megabyte	1,024 kilobytes	1,048,576
Gigabyte	1,024 megabytes	1,073,741,824
Terabyte	1,024 gigabytes	1,099,511,627,776
Petabyte	1,024 terabytes	1,125,899,906,842,624
Exabyte	1,024 petabytes	1,152,921,504,606,846,976
Zettabyte	1,024 exabytes	1,180,591,620,717,411,303,424
Yottabyte	1,024 zettabytes	1,208,925,819,614,629,174,706,176

Byte	A single letter, like "A."
Kilobyte	A 14-line e-mail. A pretty lengthy paragraph of text.
Megabyte	A good sized novel. Shelley's "Frankenstein" is only about four-fifths of a megabyte.
Gigabyte	About 300 MP3s. About 40 minutes of video at DVD quality (this varies, depending on maker). A CD holds about three-fourths of a gigabyte.
Terabyte	About thirty and a half weeks worth of high-quality audio. Statistically, the average person has spoken about this much by age 25.
Petabyte	The amount of data available on the web in the year 2000 is thought to occupy 8 petabytes.
Exabyte	In a world with a population of 3 billion, all information generated annually in any form would occupy a single exabyte. Supposedly, everything ever said by everyone who is or has lived on the planet Earth would take up 5 exabytes.
Zettabyte	Three hundred trillion MP3s; Two hundred billion DVDs. If every person living in the year 2000 had a 180 gigabyte hard drive filled completely with data, all the data on all those drives would occupy 1 zettabyte.

Sources of Big Data

Google 100 TB of data a day in 2004 to 20 PB a day in 2008

eBay two data warehouses 2PB and 6.5PB In April 2009 and growing by 150 billion new records per day.

Facebook 2.5 PB of user data, growing 15 TB per day.

Sources of Big Data

- Facebook
- One Airline flight recorder data
- Electrical Power grid
- NYSE

PB datasets are rapidly becoming the norm, and the trends are clear: our ability to store and process data is fast overwhelming

Fundamental Data characteristic

- Unstructured Data (vs. Structured Data)
 - Volume (Huge amount of data)
 - Data is in digital format
-
- Challenge is to make sense out of it. That is termed as Big Data Analytics

Fundamental Data characteristic

- Velocity
- Volume
- Variety

Also known as 3 V's

BIG IDEAS

Tackling large-data problems requires a distinct approach that sometimes runs counter to traditional models of computing.

All of these ideas have been discussed in the computer science literature for some time (some for decades), and MapReduce is certainly not the 1st to adopt these ideas.

Google deserve tremendous credit for pulling these various threads together and demonstrating the power of these ideas on a scale previously unheard of.

1. Scale out, not up

For data-intensive workloads, a large number of commodity low-end servers (i.e., the scaling out approach) is preferred over a small number of high-end servers (i.e., the scaling up approach).

The latter approach of purchasing symmetric multi-processing (SMP) machines with a large number of processor sockets (dozens, even hundreds) and a large amount of shared memory (hundreds or even thousands of gigabytes) is not cost effective, since the costs of such machines do not scale linearly (i.e., a machine with twice as many processors is often significantly more than twice as expensive).

2. Assume failures are common

At warehouse scale, failures are not only inevitable, but commonplace.

Problem 1

Assume that a 10000 server cluster is built from reliable machines with a mean-time between failures (MTBF) of 1000 days

(a) What is the failure rate?

(b) If MTBF of a machine is 10000 days, what is the failure rate?

Assume failures are common

Let us suppose that a cluster is built from reliable machines with a mean-time between failures (MTBF) of 1000 days (about three years). Even with these reliable servers, a 10,000-server cluster would still experience roughly **10 failures a day**.

For the sake of argument, let us suppose that a MTBF of 10,000 days (about thirty years) were achievable at realistic costs (which is unlikely). Even then, a 10,000-server cluster would still experience **one failure daily**.

3. Move processing to the data

In traditional high-performance computing (HPC) applications (e.g., for climate or nuclear simulations), it is commonplace for a supercomputer to have processing nodes and storage nodes linked together by a high-capacity interconnect.

Many Big Data workloads are not very processor-demanding, which means that the separation of compute and storage creates a bottleneck in the network.

Move processing to the data

As an alternative to moving data around, it is more efficient to move the processing around.

That is, MapReduce assumes an architecture where processors and storage (disk) are co-located.

In such a setup, we can take advantage of data locality by running code on the processor directly attached to the block of data we need.

The distributed file system (DFS) is responsible for managing the data over which MapReduce operates.

4. Process data sequentially and avoid random access

Data-intensive processing by definition means that the relevant datasets are too large to fit in memory and must be held on disk.

Seek times for random disk access are fundamentally limited by the mechanical nature of the devices: read heads can only move so fast and platters can only spin so rapidly. As a result, it is desirable to avoid random data access, and instead organize computations so that data is processed sequentially.

5. Hide system-level details from the application developer

Writing code is difficult because the programmer must simultaneously keep track of many details in short term memory.

Short-term memory (or "primary" or "active memory") is the capacity for holding a small amount of information in mind in an active, readily available state for a short period of time. The duration of short-term memory (when rehearsal or active maintenance is prevented) is believed to be in the order of seconds. A commonly cited capacity is 7 ± 2 elements. In contrast, long-term memory can hold an indefinite amount of information.

6. Seamless scalability

For data-intensive processing, it goes without saying that scalable algorithms are highly desirable.

Why DFS

Problem 2

Read 1 TB data

- (a) 1 machine having 4 I/O channels (or 4 hard drives) such that each can read 100 MB/sec.

- (b) 10 machine having each having 4 I/O channels (or 4 hard drives) such that each can read 100 MB/sec.

Why DFS

Read 1 TB data

- 1 machine
 - 4 I/O channels (4 hard drives)
 - Each 100 MB/sec
 - $1000000 \text{ MB} / 400 \text{ MB} = 41.666 \text{ minutes}$
- 10 machine
 - 4 I/O channels
 - Each 100 MB/sec
 - $1000000 \text{ MB} / 4000 \text{ MB} = 4.1666 \text{ minutes}$
- I/O speed is the challenge; not the storage capacity. So we need to distribute data among many nodes (machines)

What is DFS

- Machines are physically located at different places
- Logically, there is only one file system
- So we can read data in parallel into multiple machines

What is Hadoop

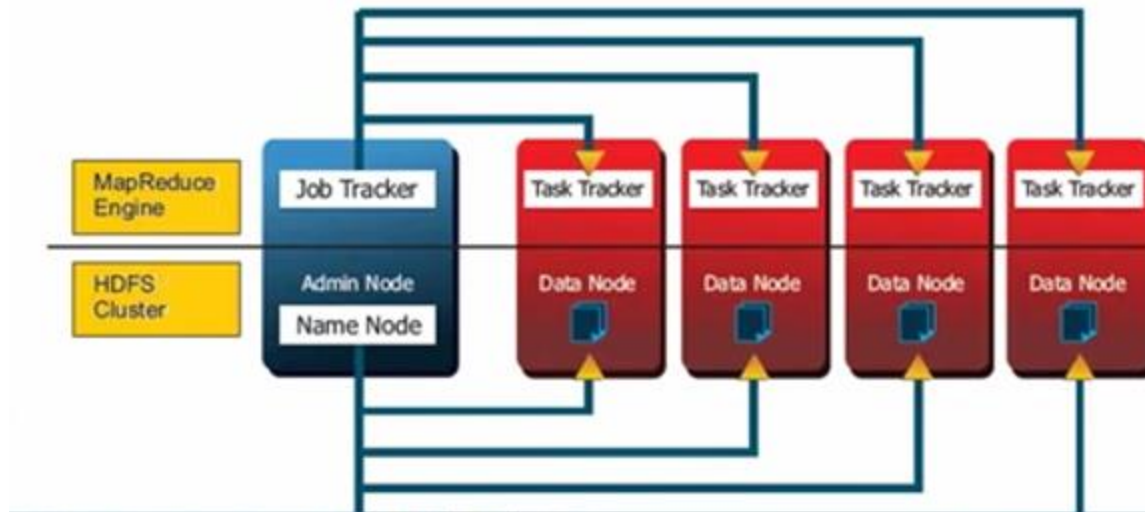
Definition

Hadoop is a framework that allows for distributed processing of large data sets across clusters of commodity computers using a simple computing model (called MapReduce to retrieve and analyze data).

Google, Yahoo, IBM, LinkedIn, Facebook, eBay, Amazon, ...

Hadoop

- HDFS (Hadoop Distributed File System) for storage
- MapReduce for processing



HDFS

- Highly fault-tolerant (replication of data in multiple nodes)
- High throughput (yahoo uses 5000 nodes)
- Suitable for applications with large data sets
- Streaming access to file system data
 - Write once and read many times.
 - Where getting the entire data faster is more important than getting a specific record.
- Can be built out of commodity hardware

HDFS

Definition

HDFS is a file system designed for storing very large files (in terabytes) with streaming data access patterns, running clusters on commodity hardware.

HDFS is not suitable for

- Low-latency data access
(latency: time interval between data request and data availability)
- Lots of small files
- Multiple writers, arbitrary modifications

HDFS components

- NameNode --- Job Tracker is a daemon
- DataNode --- Task Tracker is a daemon

Daemon is a service or process that runs in the background in unix environment

NameNode

- Master of the system
- Single point of failure
- Maintains and manages the blocks that are present on DataNodes
- Very expensive hardware with double/triple redundancy (RAID)

RAID

- **RAID** is a storage technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement.
- The term "RAID" was first used by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987, standing for **redundant array of inexpensive disks**.
- Industry RAID manufacturers later tended to interpret the acronym as standing for **redundant array of independent disks**.
- RAID is now used as an umbrella term for computer data storage schemes that can divide and replicate data among multiple physical drives.

DataNode

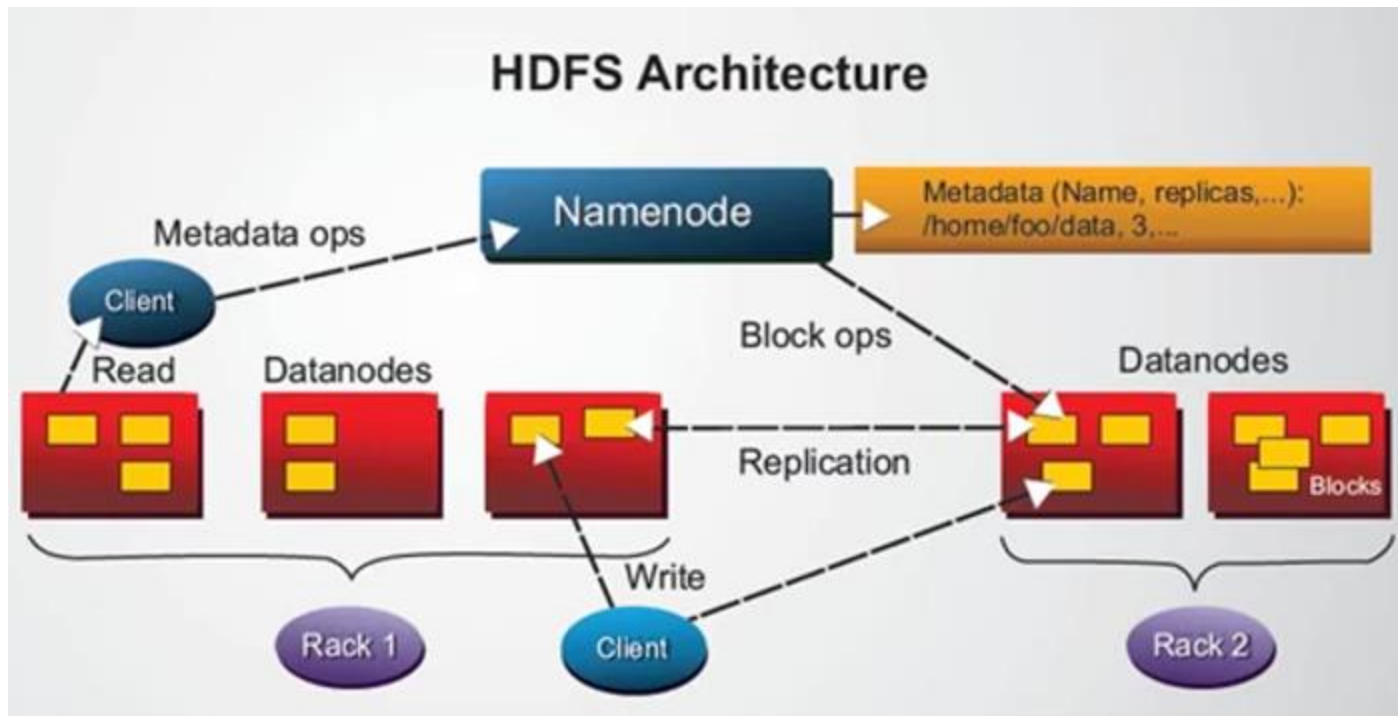
- Slaves which are deployed on each machine and provide the actual storage
- Responsible for serving the read and write requests from the clients

Job Tracker and Task Tracker



HDFS Architecture

Minimum Block size 64MB (default in linux is 8KB)



Secondary NameNode

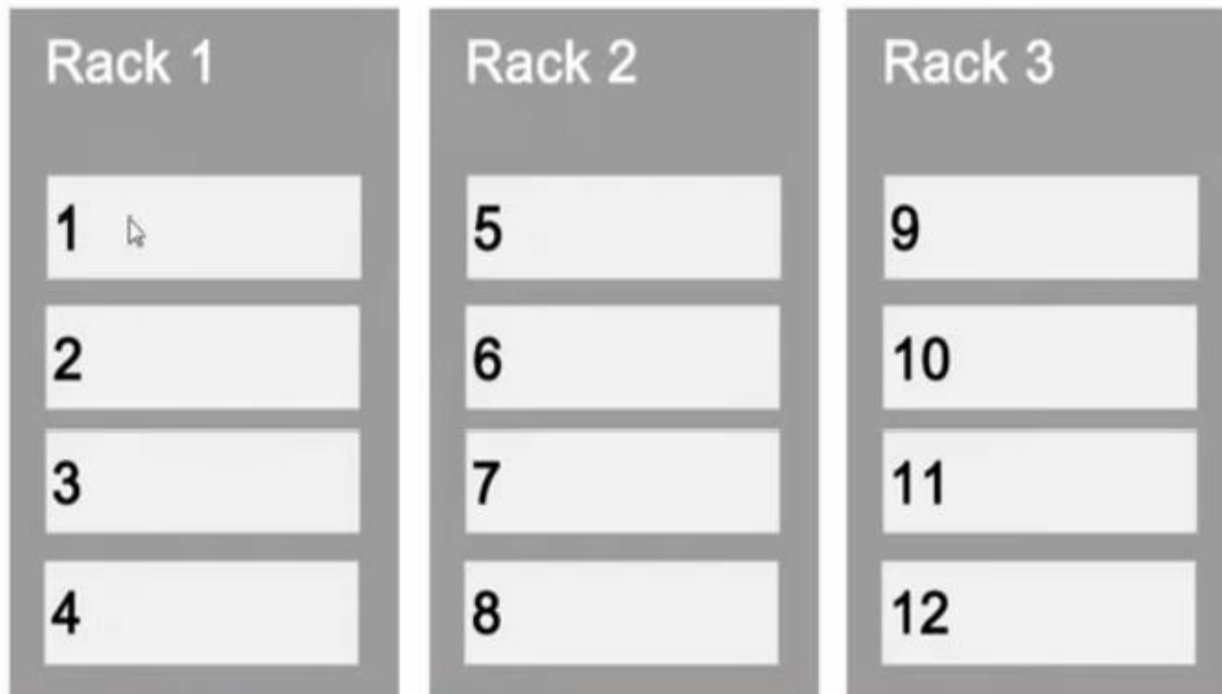
- NameNode keeps all the data in the RAM. Secondary NameNode Reads data from the RAM of the NameNode and writes into hard drive.
- Secondary NameNode is not a substitute for the NameNode. If the NameNode fails, the Secondary NameNode will not become the NameNode.
- In Gentoo Hadoop there are active and passive NameNodes. If active one fails, passive one takes over. There is a secondary NameNode in Gentoo Hadoop as well.

Three data blocks (A, B, C), three racks and each rack has 4 name nodes

Block A : 

Block B : 

Block C : 






After placing Block A

Block A : 

Block B : 

Block C : 


Rack 1		Rack 2		Rack 3	
1		5		9	
2		6		10	
3		7		11	
4		8		12	




After placing Block B



Block A : 

Block B : 

Block C : 

Rack 1	
1	
2	
3	
4	

Rack 2	
5	
6	 
7	
8	

Rack 3	
9	
10	
11	
12	

After placing Blocks A, B and C

Block A : 

Block B : 

Block C : 

