Introduction To Hadoop
What is Big data?
What is Big data?

- According to Wikipedia: Big data is a collection of data sets that are so large and complex to be dealt with by traditional data-processing application software.
Different versions of Big data?

- **Volume**: Data is being generated at an accelerated speed
- **Variety**: Different kinds of data is being generated from various sources
- **Velocity**: Data is being generated at a high speed
- **Veracity**: Uncertainty and inconsistencies in the data
Problems with Big data?

- Storing huge and exponentially growing datasets
- Processing different types of data are having complex structure (structured, semi-structured, un-structured)
- Processing huge amount of data to the computation unit becomes a bottleneck.
What is Hadoop?

- Hadoop is a framework that allows us to store and process large data sets in parallel and distributed fashion.

- Designed to answer the question: "How to process big data with reasonable cost and time?"
  - The internet introduced a new challenge with web logs
    - Large scale (petascale)
    - Unique characteristic: "Write Once, Read Many" (WORM)
  - Google exploited this characteristic in its Google File System (GFS)
  - Hadoop is the open source version of GFS
    - Distributed file system
    - Started as part of project Nutch and Lucene (focused on search)
    - Found to have wide application outside of search
    - Now its own Apache project
What is Hadoop?

- HDFS (Storage)
  - Allows to store any kind of data

- MapReduce (Processing)
  - Allows parallel processing of the data stored in HDFS
How Hadoop overcome the problems

- Storing Exponentially growing huge datasets
  HDFS, storage unit of Hadoop is a distributed file system

- Storing different types of data
  HDFS allows to store any kind of data

- Processing Data Faster
  HDFS allows to store any kind of data
Hadoop Features

- Highly fault-tolerant
- High throughput
- Handles large data sets effectively
- Streaming access to file system data
- Can be built from commodity hardware
- Files browsable with any HTTP browser
- Initially provided a Java API — but now supports most programming languages
Key Aspects

- The key aspects of Hadoop we will be discussed are:
  - Architecture
  - Protocol - rules for operation
  - Data Organization
  - Robustness
  - API to access services
  - Software (MapReduce)
Architecture
**Architecture: HDFS**

**HDFS**
- Storage unit of Hadoop
- Distributed file System
- Divide files into smaller chunks and stores across the cluster
- Store any kind of data
- No schema validation is required while inserting data
Architecture: NameNode and DataNodes

- HDFS clusters consist of a single **NameNode** and multiple **DataNodes**

**NameNode**

A master server that manages the filesystem namespace, tracks metadata, and regulates client access to files.

**DataNodes**

Usually one per node in a cluster.

Manages storage attached to their node.

Serves read/write requests, file creation/deletion, and replication.
Architecture: Files

- HDFS exposes a filesystem namespace, and allows user data to be stored in files.
- A file is split into one or more blocks, and sets of blocks are stored in the DataNodes
  - All blocks in a file are the same size (except the last)
  - These blocks may be replicated and/or migrated
HDFS exposes a filesystem namespace, and allows user data to be stored in files.

A file is split into one or more blocks, and sets of blocks are stored in the DataNodes

- All blocks in a file are the same size (except the last)
- These blocks may be replicated and/or migrated

Example:
Architecture

Client → NameNode
  MetaData ops

NameNode → DataNodes
  Block ops

DataNodes → Client
  MetaData (name, replicas, location, etc)

DataNodes → Blocks
  Replication

Blocks → DataNodes
  Write

Rack 2

Read

Write
File System

- Hierarchical file system with directories and files
  - Create, remove, move, rename, etc
- NameNode maintains the file system
- Any metadata changes to the file system recorded by the NameNode
- The replication factor for each file is also stored by the NameNode
Data Replication

- HDFS is designed to store very large files across machines in a cluster.
- Each file is a sequence of blocks.
  - All blocks in a file are the same size (except the last block).
  - Blocks are replicated for fault tolerance.
  - Block size and replica are configurable per file.
- The NameNode receives a heartbeat and BlockReport from each DataNode.
  - This report contains information about all the blocks on the DataNode.
Replica Placement

- Replica placement is critical to reliability and performance
- Optimizing replica placement is what distinguishes HDFS from other distributed file systems
- First, what does a typical cluster look like...
Replica Placement

Rack 1

Rack 2

Rack n
A rack consists of multiple DataNodes.
A rack consists of multiple DataNodes. A cluster contains many racks.
Replica Placement

A rack consists of multiple DataNodes

Communication between racks are through network switches

A cluster contains many racks
A rack consists of multiple DataNodes. A cluster contains many racks. Communication between racks are through network switches, which is slower than within rack communication.
From Brad Hedlund: a very nice picture

Hadoop Cluster

World

switch

switch

switch

switch

switch

switch

switch

switch

Name Node
DN + TT
DN + TT
DN + TT
DN + TT
Rack 1

Job Tracker
DN + TT
DN + TT
DN + TT
DN + TT
Rack 2

Secondary NN
DN + TT
DN + TT
DN + TT
DN + TT
Rack 3

Client
DN + TT
DN + TT
DN + TT
DN + TT
Rack 4

DN + TT
DN + TT
DN + TT
DN + TT
Rack N
Replica Placement

Rack Aware Placement

- Goal: Improve reliability, availability, and network bandwidth utilization
- NameNode determines the rack id for each DataNode
- Replicas are typically placed on unique racks
  - Simple scheme but non-optimal
  - Writes are expensive
  - Typical replication factor is 3
Rack Aware Placement

- **Goal:** Improve reliability, availability, and network bandwidth utilization
- **NameNode** determines the rack id for each **DataNode**
- Replicas are typically placed on unique racks
  - Simple scheme but non-optimal
  - Writes are expensive
  - Typical replication factor is 3
- **Improvement:** Place one on a node in the local rack, one on a node in a remote rack, and another on a different node in that same remote rack.
- For a file this means $\frac{1}{3}$ of the replicas on one node, $\frac{2}{3}$ on one rack, and the other $\frac{1}{3}$ distributed across all other racks.
Replica Selection

- When a client attempts to perform a read, HDFS tries to minimize bandwidth consumption and latency.
  - If there is a replica on the reader's node, then that is preferred
  - Otherwise try for the same rack
  - An HDFS cluster may span multiple data centers, a replica in the local data center is preferred over a remote data center
Startup (Safemode)

- On initial startup of a NameNode, the system enters a safe mode
  - During safe mode no replication occurs
- Each DataNode checks in with a heartbeat and a BlockReport
- Each Block in the BlockReport has a minimum number of replicas to be considered "safely replicated"
- The NameNode waits for a certain percentage of blocks to reach the threshold for safely replicated before leaving safe mode and replicating any blocks which are lacking
File System MetaData

- The HDFS namespace is stored by NameNode.
- There are two files associated with the metadata:
  - **EditLog**: It contains all the recent modifications made to the file system metadata.
    - Creating a new file
    - Changing the replication factor
    - Deletion
  - **FsImage**: It contains the complete state of the file system namespace since the start of the namespace. Entire namespace including block mapping is stored in FsImage file in the NameNodes local filesystem.
A DataNode stores data in files in its local file system. DataNode has no knowledge about the HDFS file system. It stores each block in a separate file. It does not create all files in the same directory. It uses heuristics to determine the optimal number of files per directory. Upon starting, it generates a list of all blocks and sends the report to the NameNode.
Robustness
Possible Failures

● The primary objective of HDFS is to store data reliably in the presence of failures

● Three common failures that it must handle are:
  ○ DataNode failure
  ○ Network partition
  ○ NameNode failure
DataNode Failure and Heartbeat

- A crashed DataNode or a network partition can cause a subset of DataNodes to lose connectivity with the NameNode.
- NameNode detects this by the absence of a heartbeat:
  - NameNode marks these DataNodes, and does not send requests to them.
  - Data registered to the failed DataNode is not available to the HDFS.
  - Death of a DataNode may cause some blocks to require more replication.
Re-Replication

● Sometimes Blocks in the system may fall below the required replication factor
● This can occur for a number of reasons
  ○ A DataNode has become unavailable
  ○ A replica may become corrupted
  ○ A hard disk on a DataNode may fail
  ○ The replication factor may have been increased
Data Integrity

- What if a block of data fetched from a DataNode arrives corrupted
  - Fault in a storage device
  - Network faults
  - Buggy software
- An HDFS client creates a checksum for every block of its file and stores it in the HDFS namespace
- When the client retrieves the contents of a file it verifies that they match...if not it must retrieve the block from another replica
MetaData Disk Failure

- **FsImage** and **EditLog** are central data structures of HDFS
  - Corruption of these files can cause an entire HDFS instance to become non-functional.
- A NameNode can be configured to maintain multiple copies of these files
  - These copies are updated synchronously
  - MetaData is not data intensive
- The NameNode is a potential single point of failure
  - This currently requires manual intervention
Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes
- A scheme may move data from one DataNode to another if the free space on a DataNode is falling below a certain threshold
- A scheme may dynamically create and place additional replicas and rebalance other data if there is sudden high demand for a particular file
- These types of rebalancing are not yet implemented
Data Organization
Data Block

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.
Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.
Staging (Cont.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion; also it has precedence is AFS (Andrew file system).
API
FS Shell, Admin, and Browser Interface

- HDFS organizes its data in files and directories
- Command line interface called the FS shell
  - Syntax similar to bash and csh
    - ie: /bin/hadoop dfs -mkdir /foo
- There is also a DFSAdmin interface
- A browser can be used to view the namespace
Space Reclamation

- When a file is deleted, HDFS moves it to a trash directory for a configurable amount of time.
- A client can request for the file to be recovered during this time.
- After the specified time the file is deleted along with replicas, and all space is reclaimed.
- This will also occur automatically if the replication factor is reduced.