

MODULE 2

Introduction to Hadoop

1. Introduction to Big Data and Hadoop

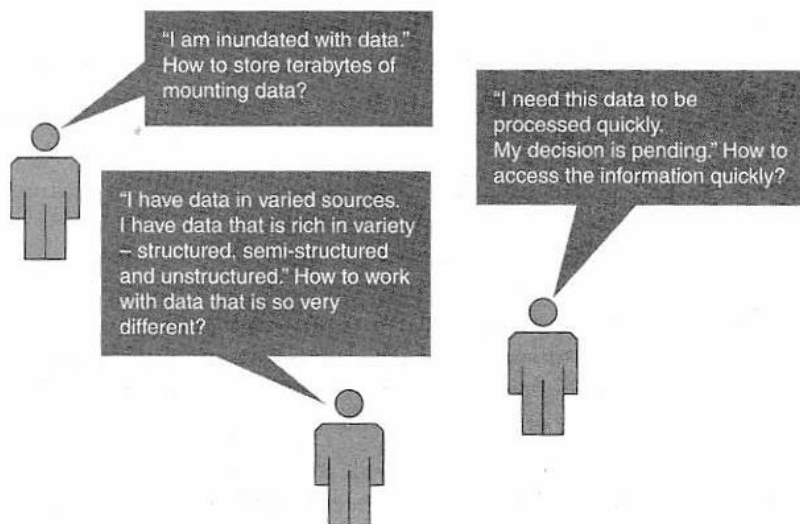


Figure 5.1 Challenges with big volume, variety, and velocity of data.

- **Big Data:** A prevalent term referring to the massive volume, variety, and velocity of data generated daily.
- **Statistics:**
- **Daily:** NYSE (1.5 billion shares/trades), Facebook (2.7 billion comments/likes), Google (24 petabytes processed).
- **Minute:** Facebook (2.5 million content shares), Twitter (300,000 tweets), Instagram (220,000 photos), YouTube (72 hours video upload), Apple (50,000 app downloads), Email (200 million messages), Amazon (\$80,000 sales), Google (4 million search queries).
- **Second:** Banking (10,000+ credit card transactions).
- **Data as a Treasure Trove:** Provides business advantages like product recommendations, new product invention, market analysis, early key indicators, and enables precise analysis.
- **Challenges with Big Data:**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Handling immense volume, variety (structured, semi-structured, unstructured), and velocity of data.
- Accessing and processing diverse data types.
- Working with data that is difficult to manage.

2. Why Hadoop?

Hadoop's popularity stems from its ability to handle massive amounts of diverse data quickly. Key considerations include:

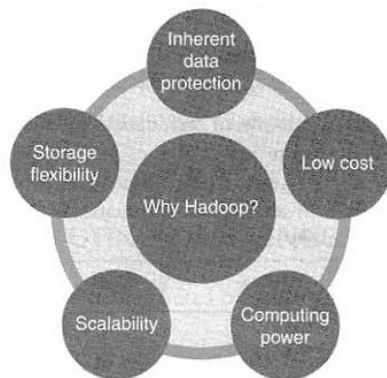
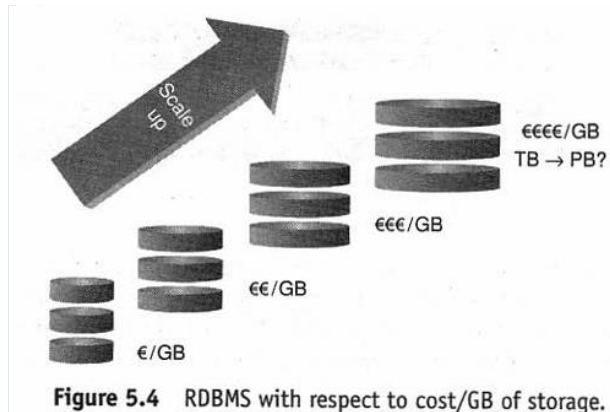


Figure 5.2 Key considerations of Hadoop.

- **Low Cost:** Open-source framework utilizing commodity hardware (inexpensive and readily available).
- **Computing Power:** Based on a distributed computing model, processing large volumes of data in parallel. Processing power increases with more computing nodes.
- **Scalability:** Easily scales by adding more nodes with minimal administration.
- **Storage Flexibility:**
 - Data does not need pre-processing before storage (unlike RDBMS).
 - Store data first, decide on usage later.
 - Supports unstructured data like images and videos.
- **Inherent Data Protection:** Protects against hardware failure by automatically redirecting jobs to functional nodes. Stores multiple data replicas across the cluster.
- **Framework:** Utilizes commodity hardware, a distributed file system, and distributed computing. Data is distributed and duplicated across nodes in a cluster.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

2. RDBMS vs. Hadoop



- **RDBMS Limitations:** Not suitable for large files, images, videos, or advanced analytics like machine learning. High investment required as data volume grows.
- **Hadoop Advantages:**
 - **Data Type:** Suitable for structured, unstructured, and semi-structured data (XML, JSON, text files, etc.).
 - **Focus:** Analytical, Big Data Processing.
 - **Processor:** Uses commodity hardware (processor, network card, hard drives).
 - **Cost:** Significantly lower cost per terabyte of storage compared to RDBMS.

4. Distributed Computing Challenges

- **Hardware Failure:**
 - In a distributed system, hardware failure is common.
- **Hadoop Solution: Replication Factor (RF):** Stores multiple copies (replicas) of data blocks across the network. If a node fails, data is still accessible from other replicas.
- **Analogy:** Project manager sharing critical information with multiple team members to avoid project stalls due to absence.
- **Processing Gigantic Data:**
 - Integrating data spread across multiple machines.
- **Hadoop Solution: MapReduce Programming:** A programming model designed to process data in parallel across distributed nodes.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

5. History of Hadoop

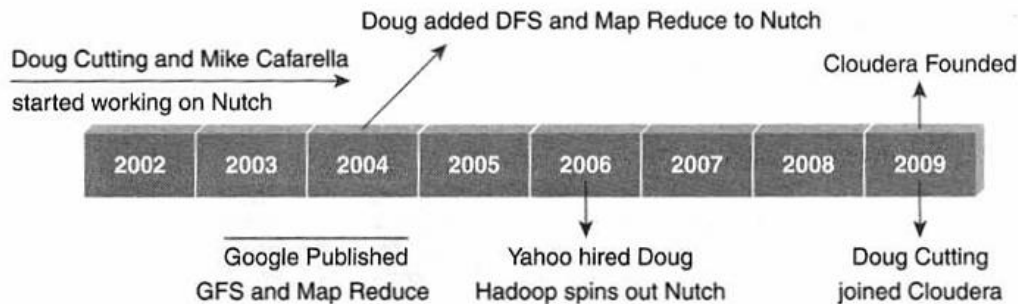


Figure 5.6 Hadoop history.

- Created by Doug Cutting (creator of Apache Lucene).
- Part of the Apache Nutch (Yahoo) project and the Lucene project.
- **Name Origin:** "Hadoop" is a made-up name given by Doug Cutting's child to a stuffed yellow elephant, chosen for being short, easy to spell/pronounce, meaningless, and unused.

6. Hadoop Overview

- **Definition:** An open-source software framework for storing and processing massive amounts of data in a distributed fashion on clusters of commodity hardware.
- **Core Tasks:**
 1. Massive data storage.
 2. Faster data processing.
- **Key Aspects:**
 - **Distributed File System (HDFS):** Stores data across multiple computers, inherently redundant.
 - **MapReduce:** Computational framework that splits tasks across multiple nodes for parallel processing.
- **Hadoop Ecosystem:** Support projects enhancing Hadoop's functionality (e.g., HIVE, PIG, SQOOP, HBASE, FLUME, OOZIE, MAHOUT).
- **Conceptual Layer:**
 - **Data Storage Layer:** Stores huge volumes of data.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- **Data Layer:** Processes data in parallel for insights.
- **High-Level Architecture:** Master-Slave architecture.
- **Master Node:** NameNode.
- **Slave Nodes:** DataNodes.

7. Hadoop Components

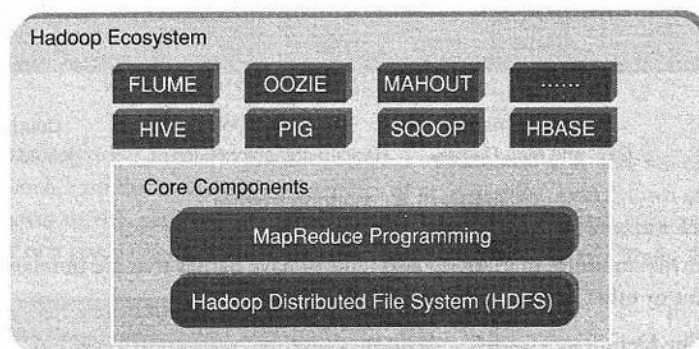


Figure 5.8 Hadoop components.

- **Hadoop Core Components:**
- **HDFS (Hadoop Distributed File System):**
 - Storage component.
 - Distributes data across nodes.
 - Natively redundant.
- **MapReduce:**
 - Computational framework.
 - Splits tasks across nodes.
 - Processes data in parallel.
- **Hadoop Ecosystem:** Supporting projects (HIVE, PIG, SQOOP, HBASE, FLUME, OOZIE, MAHOUT).

8. Hadoop Use Case: ClickStream Data Analysis

- **ClickStream Data:** Tracks customer mouse clicks to understand purchasing behavior.
- **Benefits:**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

1. **Data Integration:** Join ClickStream data with CRM, demographics, sales, and advertising campaign data for deeper customer insights.
2. **Scalability & Cost-Effectiveness:** Store years of data affordably for temporal analysis.
3. **Analysis Tools:** Business analysts can use Apache Pig or Hive for website analysis, organizing, refining, and visualizing ClickStream data.

9. Hadoop Distributors

Companies providing Apache Hadoop products and support (e.g., Cloudera, Hortonworks, MapR, Apache Hadoop).

10. HDFS (Hadoop Distributed File System)

- **Key Points:**
 - Storage component of Hadoop.
 - Distributed File System.
 - Modeled after Google File System.
 - Optimized for high throughput (large block size, computation moves to data).
 - Replication for fault tolerance (software and hardware).
 - Automatically re-replicates data blocks on node failure.
 - Effective for reading/writing large files (gigabytes+).
 - Sits on top of native file systems (e.g., ext3, ext4).
- **Architecture:**
 - Client interacts with NameNode for metadata.
 - Client communicates with DataNodes for reading/writing files.
 - DataNodes communicate for pipeline reads.
- **HDFS Daemons:**
 - **NameNode:**
 - Manages file system namespace.
 - Stores metadata (file properties, block locations) in **FsImage** and **EditLog**.
 - Manages file operations (read, write, create, delete).
 - Single NameNode per cluster.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- **DataNode:**

- Multiple DataNodes per cluster.
- Communicate with each other during pipeline operations.
- Send "heartbeat" messages to NameNode to indicate liveness.
- If a heartbeat is missed, NameNode replicates data to other nodes.

- **Secondary NameNode:**

- Takes periodic snapshots of HDFS metadata.
- Can be configured to bring up the cluster in case of NameNode failure (manual configuration).
- Does not record real-time changes.
- Recommended to run on a separate machine from the NameNode.

- **Anatomy of File Read:**

1. Client calls **open()** on **DistributedFileSystem**.
2. **DistributedFileSystem** contacts NameNode for block locations.
3. NameNode returns DataNode addresses.
4. **DistributedFileSystem** returns **FSDDataInputStream** to the client.
5. Client calls **read()** on the stream, connecting to the closest DataNode for the first block.
6. Client repeatedly calls **read()** to stream data.
7. Upon reaching block end, connection to DataNode closes, and the process repeats for the next block.
8. Client calls **close()** on **FSDDataInputStream** upon completion.

Anatomy of File Write:

0. Client calls **create()** on **DistributedFileSystem**.
1. **DistributedFileSystem** calls NameNode to create the file (checks for existence).
2. NameNode creates the file without initial blocks.
3. **DistributedFileSystem** returns **FSDDataOutputStream** to the client.
4. As client writes data, **DFSOutputStream** splits it into packets, queues them.
5. **DataStreamer** consumes the queue, requests block allocation from NameNode (selecting DataNodes for replicas).
6. A pipeline of DataNodes is formed (e.g., three nodes for default replication).
7. **DataStreamer** sends packets to the first DataNode, which forwards to the next, and so on.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

8. `DFSOutputStream` manages an "Ack queue" for packets acknowledged by all `DataNodes` in the pipeline.
9. Client calls `close()` to flush remaining packets and wait for acknowledgments.
10. `NameNode` is informed of completion.

Replica Placement Strategy (Default):

0. First replica on the same node as the client.
1. Second replica on a different rack.
2. Third replica on the same rack as the second, but a different node.
3. Pipeline is built for reliability.

Working with HDFS Commands:

- `hadoop fs -ls /`: List directories and files at the root.
- `hadoop fs -mkdir /sample`: Create a directory.
- `hadoop fs -put <local_file> <hdfs_path>`: Copy file from local to HDFS.
- `hadoop fs -get <hdfs_file> <local_path>`: Copy file from HDFS to local.
- `hadoop fs -copyFromLocal <local_file> <hdfs_path>`: Similar to `put`.
- `hadoop fs -copyToLocal <hdfs_file> <local_path>`: Similar to `get`.
- `hadoop fs -cat <hdfs_file>`: Display HDFS file content.
- `hadoop fs -cp <source_hdfs_file> <destination_hdfs_path>`: Copy file within HDFS.
- `hadoop fs -rm -r <hdfs_directory>`: Remove a directory.

Special Features of HDFS:

- **Data Replication:** Client doesn't need to track blocks; it's directed to the nearest replica for performance.
- **Data Pipeline:** Efficient writing of blocks to `DataNodes` in a sequential pipeline.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

11. Processing Data with Hadoop: MapReduce

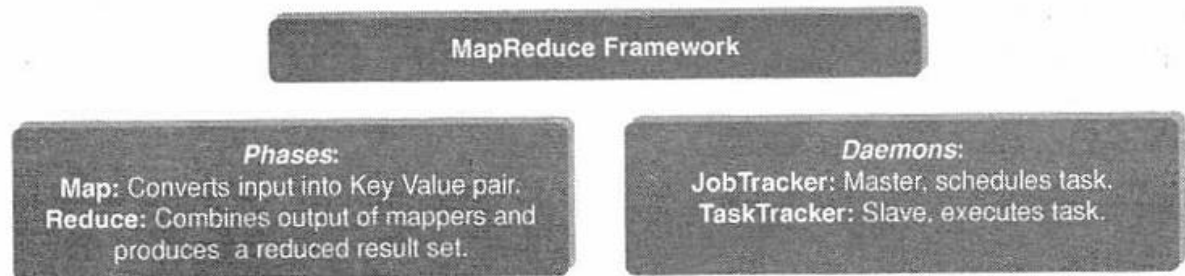


Figure 5.21 MapReduce Programming phases and daemons.

- **MapReduce Programming:** A software framework for processing massive data in parallel.
- **Process:**
 1. Input dataset is split into independent chunks.
 2. **Map Tasks:** Process each chunk in parallel, producing intermediate key-value pairs stored locally.
 3. **Shuffle and Sort:** Framework automatically shuffles and sorts mapper outputs by keys.
 4. **Reduce Tasks:** Process the sorted intermediate data to produce the final reduced output.
- **Data Locality:** MapReduce tasks are assigned to DataNodes where the data resides, minimizing network transfer.
- **Daemons:**
 - **JobTracker (Master):** Schedules tasks on TaskTrackers, monitors tasks, and reschedules failed tasks. Single JobTracker per cluster.
 - **TaskTracker (Slave):** Executes tasks assigned by the JobTracker. One TaskTracker per slave node, can run multiple JVMs for parallel tasks.

MapReduce Workflow:

0. Input dataset split into subsets.
1. Master (JobTracker) and workers (TaskTrackers) processes created.
2. Map tasks read assigned data, extract relevant information, and generate key/value pairs.
3. Partitioner function divides data into regions for specific reducers.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

4. Reduce workers contact map workers for their partition's data, which is shuffled and sorted by keys.
5. Reduce function is called for each unique key, producing output.
6. Master transfers control to the user program upon completion.

MapReduce Example: Word Count:

- **Components:** Driver Class (Job Configuration), Mapper Class (Map Function), Reducer Class (Reduce Function).

SQL vs. MapReduce:

- **Access:** Interactive & Batch (SQL) vs. Batch (MapReduce).
- **Structure:** Static (SQL) vs. Dynamic (MapReduce).
- **Updates:** Read/write many times (SQL) vs. Write once, read many times (MapReduce).
- **Integrity:** High (SQL) vs. Low (MapReduce).
- **Scalability:** Nonlinear (SQL) vs. Linear (MapReduce).

12. Managing Resources and Applications with Hadoop YARN

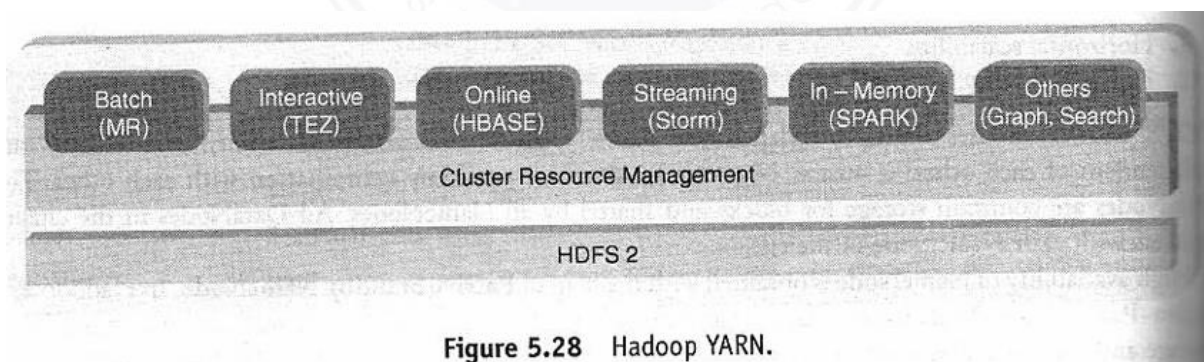


Figure 5.28 Hadoop YARN.

- **Hadoop YARN (Yet Another Resource Negotiator):** A sub-project of Hadoop 2.x, a general-purpose processing platform.
- **Hadoop 1.0 Limitations:**
 - HDFS and MapReduce are tightly coupled.
 - Single NameNode bottleneck.
 - Restricted processing model (batch-oriented MapReduce).

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Not suitable for interactive analysis, machine learning, or memory-intensive algorithms.
- Resource utilization issues (map slots full, reduce slots empty).
- **HDFS Limitations (Hadoop 1.0):** Single NameNode memory limit.
- **Hadoop 2: HDFS Federation:**
- Multiple independent NameNodes for horizontal scalability.
- DataNodes are shared storage.
- **High Availability:** Active-Passive NameNode with automatic failover.
- **Hadoop 2 YARN: Beyond Batch Processing:**
- Stores all data in one place, enabling predictable performance for various processing types (Batch, Interactive, Online, Streaming, Graph).
- **Fundamental Idea:** Separates resource management from job scheduling/monitoring.
- **YARN Daemons:**
- **Global ResourceManager:** Distributes resources among applications.
- **Scheduler:** Allocates resources (pluggable).
- **ApplicationManager:** Accepts job submissions, negotiates resources (containers), restarts ApplicationMasters on failure.
- **NodeManager:** Per-machine slave daemon. Launches application containers, monitors resource usage (memory, CPU, disk, network), reports to ResourceManager.
- **Per-application ApplicationMaster:** Application-specific entity. Negotiates resources from ResourceManager, works with NodeManager for execution and monitoring.
- **Basic Concepts:**
- **Application:** A job submitted to the framework (e.g., MapReduce Job).
- **Container:** Basic unit of allocation, fine-grained resource allocation (Memory, CPU, disk, network). Replaces fixed map/reduce slots.
- **YARN Architecture:**
- 1. Client submits application to ResourceManager.
- 2. ResourceManager launches ApplicationMaster in a container.
- 3. ApplicationMaster registers with ResourceManager.
- 4. ApplicationMaster negotiates resource containers via request protocol.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

5. ResourceManager allocates containers.
6. ApplicationMaster launches containers via NodeManager.
7. NodeManager executes application code and reports status to ApplicationMaster.
8. Client communicates directly with ApplicationMaster for status updates.
9. ApplicationMaster deregisters and shuts down upon completion.

13. MapReduce Programming Deep Dive

- **Map Task Phases:**
 1. **RecordReader:** Converts byte-oriented input into record-oriented key-value pairs.
 2. **Map:** Transforms input key-value pairs into intermediate key-value pairs.
 3. **Combiner (Optional):** Local reducer for intermediate data optimization (reduces network bandwidth and disk space).
 4. **Partitioner:** Splits intermediate data into shards and sends them to specific reducers based on keys.
- **Reduce Task Phases:**
 1. **Shuffle and Sort:** Downloads partitioned data, sorts it by keys, and groups values for each key.
 2. **Reduce:** Applies the reduce function to grouped data to produce output.
 3. **Output Format:** Formats key-value pairs and writes them to a file.
- **Mapper:** Maps input key-value pairs to intermediate key-value pairs.
- **Combiner:** Optimization technique, often the same class as the Reducer, processes intermediate data before it reaches the reducer.
- **Partitioner:** Determines which reducer receives the output of a mapper. Default is hash partitioner. Custom partitioners can be implemented (e.g., based on alphabet for word count).
- **Searching:** MapReduce can be used to search for specific keywords within files.
- **Sorting:** MapReduce can sort data based on specified keys (e.g., student names).
- **Compression:** MapReduce output files can be compressed (e.g., using GzipCodec) to reduce storage space and speed up data transfer.