Big Data Analytics
Lecture 5: Storage and Introduction to Hadoop

Based on Slides by Dr. Tariq Mahmood

Fall 2025

Abstract

These notes cover the foundational concepts of storage technologies crucial for Big Data Analytics and introduce the Hadoop ecosystem. We will delve into the characteristics of HDDs, SSDs, and advanced interfaces like NVMe. The core of the notes focuses on the Hadoop Distributed File System (HDFS), its architecture, advantages, and disadvantages. We will also explore the MapReduce paradigm, the evolution of the Hadoop ecosystem, and its comparison with other data processing frameworks like High-Performance Computing (HPC).

Contents

1 The Foundation: Storage Technologies for Big Data

The term "Big Data" implies not just volume, but also the velocity at which it is generated and the variety of its forms. The ability to store and access this data efficiently is the bedrock upon which all analytics are built. The evolution of storage technology has been a key enabler of the big data revolution.

1.1 Traditional Storage: Hard Disk Drives (HDD)

HDDs have been the workhorse of data storage for decades.

- **Technology:** An HDD consists of a series of spinning magnetic disks, or platters, coated with a ferromagnetic material. Data is written and read by a fast-moving read/write head that hovers over the platters. The direction of magnetization on the platter represents individual bits (0s or 1s).
- **Performance:** A typical consumer HDD spins at 7200 RPMs. The performance is constrained by mechanical movements: **seek time** (moving the head to the correct track) and **rotational latency** (waiting for the correct sector to spin under the head). This results in sequential read/write speeds of approximately 80-160 MB/s.
- **Use Case in BDA:** Due to their low cost per gigabyte, HDDs are extremely cost-efficient for long-term, offline storage. They form the backbone of data lakes and archival systems where massive capacity is needed and access speed is not the primary concern.

1.2 Modern Storage: Solid State Drives (SSD)

SSDs represent a paradigm shift from mechanical to solid-state storage.

- **Technology:** SSDs use non-volatile NAND flash memory to store data. With no moving parts, they are inherently faster, more durable, and more energy-efficient than HDDs. Data is retained even when the device is powered off.
- **Performance:** The absence of mechanical parts eliminates seek time and rotational latency. This allows for significantly lower latency (<0.1 ms) and much higher Input/Output Operations Per Second (IOPS). This makes them ideal for tasks that require fast, random access to data.
- **Faster Processes:** The performance benefits of SSDs are evident in many day-to-day and enterprise tasks, including: booting up operating systems, starting programs, loading large files (e.g., video games, large datasets), and importing/exporting large media files.

1.3 Connecting Storage: Interfaces (SATA vs. NVMe)

The interface connecting the storage drive to the motherboard is a critical determinant of overall performance.

- **SATA (Serial Advanced Technology Attachment):** This interface was originally designed for HDDs. While modern SSDs can use it, the protocol itself becomes a bottleneck. SATA III, the current standard, supports speeds up to 6 Gbps (approximately 600 MB/s), which is easily saturated by modern SSDs.
- **NVMe (Non-Volatile Memory Express):** NVMe is a high-performance interface designed specifically for SSDs. It leverages the PCIe (Peripheral Component Interconnect Express) bus, the same high-speed interface used by graphics cards, providing a much more direct and faster connection to the CPU. Modern NVMe SSDs can achieve speeds of up to 7,000 MB/s and beyond, an order of magnitude faster than SATA SSDs.

1.4 Comparative Analysis: HDD vs. SSD for Big Data Analytics

1.5 The Storage Blues: Capacity vs. Speed

A critical challenge in data storage, termed the "Storage Blues," is that **storage capacity has increased much more rapidly than access speed**.

Table 1: HDD vs. SSD Impact on Big Data Analytics (BDA)

Aspect	HDD (Hard Disk Drive)	SSD (Solid State Drive)	Impact on BDA
Technology	Magnetic spinning disks with a mechanical read/write head.	Flash-based NAND memory with no moving parts.	SSDs eliminate mechanical latency, leading to faster I/O for analytics workloads.
Read/Write Speed	˜80–160 MB/s (sequential).	˜500 MB/s (SATA) to ¿7,000 MB/s (NVMe).	SSDs dramatically accelerate data ingestion, ETL processes, and real-time queries.
Latency	5–10 ms (due to seek time and rotational delay).	<0.1 ms (no mechanical delays).	Lower latency is essential for faster query response times, especially in streaming analytics.
IOPS	˜100–200 IOPS.	50,000–1,000,000+ IOPS.	SSDs can handle the massive parallel reads/writes required by distributed frameworks like Spark and Hadoop.
Cost per GB	Very low (˜$0.02–$0.03/GB).	Higher (˜$0.08–$0.12/GB), but decreasing.	HDDs are better for cold storage (infrequently accessed data), while SSDs are preferred for hot/active datasets.
Power Consumption	Higher (6–10 W).	Lower (2–5 W).	In large-scale BDA clusters, SSDs can lead to significant cost savings in energy and cooling.
Best Use Cases	Data lakes, archival storage, batch analytics (e.g., raw HDFS layers).	Real-time analytics, machine learning pipelines, high-performance query engines (e.g., compute nodes in a hybrid strategy).	

- In 1990, a typical HDD could store 1,370 MB and read that data in about five minutes.
- Over 20 years later, 1-terabyte drives became common, but with a transfer speed of around 100 MB/s, reading the entire drive would take over two and a half hours.

This growing gap means that having a massive amount of data on a single drive is not useful if you cannot process it in a timely manner. This physical limitation is a primary motivation for distributed computing. If one drive is too slow, the solution is to use many drives working in parallel.

2 The Rise of Distributed Systems: Hadoop

To overcome the "Storage Blues" and the physical limitations of single machines, a new paradigm was needed. This led to the development of distributed systems, with Apache Hadoop at the forefront.

2.1 The Problem of Hardware Failure

When you build a system from many commodity components, the probability of a single component failing becomes very high.

- **The Challenge:** As soon as you start using many pieces of hardware, the chance that one will fail is high. In a cluster of thousands of nodes, it is a certainty that nodes will fail every day.

page21_image2.png

Figure 1: RAID 6 provides fault tolerance through parity blocks (e.g., Ap, Aq).

- **The Solution: Replication.** The fundamental solution to hardware failure is to make redundant copies of data. If one copy is lost due to a hardware failure, another copy can be used.

This concept is implemented at different levels. **RAID (Redundant Array of Independent Disks)** implements replication at the hardware level within a single server. Hadoop, however, implements replication at the software level across a cluster of servers.

2.2 Introducing HDFS: The Hadoop Distributed File System

HDFS is the primary storage system of Hadoop. It is a distributed file system designed to run on large clusters of commodity hardware. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. [cite: Apache Hadoop Project]

2.2.1 HDFS Architecture

HDFS has a master/slave architecture.

- **NameNode (Master):** The NameNode is the centerpiece of an HDFS file system. It manages the file system namespace (the directory tree and file metadata) and regulates access to files by clients. It stores the metadata in memory for fast access, including the mapping of file blocks to DataNodes. It does not store the actual data itself.
- **DataNodes (Slaves):** DataNodes are responsible for storing the actual data in HDFS. A file is split into one or more blocks (typically 128 MB in size), and these blocks are stored in a set of DataNodes. DataNodes serve read and write requests from the file system's clients. They also perform block creation, deletion, and replication upon instruction from the NameNode.

2.2.2 How HDFS Works: An Example

Let's consider writing a 300 MB file to HDFS with a block size of 128 MB and a replication factor of 3 (the default).

1. The client contacts the NameNode to initiate the file write. The NameNode responds with the locations of DataNodes suitable for storing the file's blocks.

2. The 300 MB file is split into three blocks: Block A (128 MB), Block B (128 MB), and Block C (44 MB).

3. **Replication:** For Block A, the client writes it to the first DataNode. This DataNode then pipelines the block to a second DataNode, which in turn pipelines it to a third. This creates three copies of Block A on three different machines. The same process is repeated for Block B and Block C.

4. The NameNode is aware of the locations of all replicas. HDFS is "rack-aware," meaning it tries to place replicas on different racks to protect against a full rack failure (e.g., due to a switch failure or power outage). Typically, one replica is placed on a node in the same rack as the client, a second on a different node in the same rack, and the third on a node in a different rack.

2.2.3 Advantages of HDFS

- **Fault Tolerance:** With a default replication factor of 3, HDFS can tolerate the failure of 2 DataNodes holding replicas of a block. The NameNode will automatically detect the failure and create new replicas on other healthy nodes.
- **Scalability:** HDFS can be scaled out horizontally by simply adding more DataNodes to the cluster. This allows it to store petabytes of data.
- **High Throughput:** HDFS is optimized for streaming reads of large files, which is a common access pattern in big data processing. It provides high aggregate read/write bandwidth.
- **Commodity Hardware:** It is designed to run on inexpensive, off-the-shelf hardware, making it a cost-effective solution for storing massive datasets.
- **Data Locality:** HDFS is the foundation for moving computation to the data, not the other way around. This principle is key to Hadoop's performance.

2.2.4 Disadvantages of HDFS

- **High Latency for Random Access:** HDFS is optimized for high throughput via streaming reads, not for low-latency random seeks. It is not suitable for applications that require fast access to small amounts of data (like an OLTP database).
- **Small Files Problem:** The NameNode stores all metadata in memory. A large number of small files can consume the NameNode's memory, creating a bottleneck for the entire cluster. Each small file, even if only a few KB, occupies a block's worth of metadata.
- **Single Point of Failure (SPOF):** In older versions of Hadoop, the NameNode was a single point of failure. If it failed, the entire cluster would be inaccessible. Modern Hadoop has a High Availability feature with a standby NameNode to mitigate this.
- **Write-Once, Read-Many Model:** Files in HDFS are typically written once and then read many times. Appending to files is supported, but arbitrary modifications to files are not.

2.2.5 RAID vs. HDFS Replication

3 Processing Distributed Data: MapReduce

Storing data in a distributed manner is only half the battle. The next problem is how to process it.

- **The Challenge:** Most big data tasks need to combine data. Data from one disk may need to be combined with data from any of the other 99 disks. Coordinating this is very difficult.
- **The Solution: MapReduce.** MapReduce is a programming model and processing engine that abstracts the problem of distributed data processing. It allows developers to write code that can process vast amounts of data in parallel on a large cluster, without having to worry about the complexities of parallelism, fault tolerance, and data distribution. [cite: Google Research]

MapReduce works in two main phases:

Table 2: RAID vs. HDFS Replication

Aspect	RAID Replication	HDFS Replication
Scope	Within a single server.	Across a cluster of servers/nodes.
Granularity	Replicates data at the disk block level.	Replicates HDFS blocks (default 128 MB) across nodes.
Location	All copies are stored inside one machine.	Copies are distributed across different machines and racks.
Failure Recovery	Automatic switch to a mirror disk. Can't survive a full server failure.	Automatic recovery by replicating a missing block to a new healthy node. Can survive entire server or rack failures.
Use Case in BDA	Useful for protecting metadata on critical servers like the NameNode.	The core mechanism for data durability and fault tolerance in a large-scale cluster.

1. **Map Phase:** The master node takes the input, partitions it up into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

2. **Reduce Phase:** 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.

This model, combined with HDFS, creates a powerful, reliable, and scalable platform for batch processing of large datasets.

3.1 Data Locality: The Heart of Hadoop's Performance

The most significant departure of the Hadoop/MapReduce paradigm from traditional High-Performance Computing (HPC) is the principle of **data locality**.

- **HPC Approach:** In a traditional HPC cluster, computation is separated from storage. Compute nodes access data from a shared filesystem over a high-speed network (like a Storage Area Network, or SAN). For data-intensive jobs, this network can become a bottleneck.

- **Hadoop Approach:** Hadoop co-locates data with computation. Because HDFS is running on the same nodes that will run MapReduce tasks, Hadoop's scheduler can run the "Map" task on the same node where the data block resides. This is called **Data-Local** access and is extremely fast as it avoids network traffic. If a local node is busy, the scheduler will try to run the task on a node in the same rack (**Rack-Local**) before going to a different rack (**Different Rack**).

By treating network bandwidth as the most precious resource, Hadoop achieves high performance and scalability for data-heavy workloads.

4 The Broader Hadoop Ecosystem

Over time, "Hadoop" has evolved from just HDFS and MapReduce to a rich ecosystem of tools and projects, often managed by the Apache Software Foundation.

- **YARN (Yet Another Resource Negotiator):** Introduced in Hadoop 2, YARN is a cluster resource management system. It decouples resource management from processing, allowing various distributed applications (not just MapReduce) to run on a Hadoop cluster.

- **HBase:** A non-relational, distributed database that runs on top of HDFS. It provides low-latency random read/write access to individual rows, which HDFS cannot do, making it suitable for building applications.

- **Hive & Pig:** High-level data flow languages and execution frameworks. Hive provides a SQL-like interface (HiveQL) to query data stored in HDFS, while Pig uses a scripting language called Pig Latin. Both compile their queries into MapReduce (or other engine) jobs.

- **Spark:** A fast and general-purpose cluster computing system. Spark can be much faster than MapReduce for many workloads because it performs processing in-memory. It can run on YARN and read/write data from HDFS.

4.1 History of Hadoop

The creation of Hadoop was driven by the needs of web-scale search engines and inspired by seminal papers from Google.

- **Creator:** Doug Cutting, creator of the text search library Apache Lucene.
- **Origins (2002-2004):** Started as part of Apache Nutch, an open-source web crawler. The Nutch team realized they needed a scalable storage system for the billions of pages on the web.
- **Google's Influence:** In 2003, Google published a paper on its distributed filesystem, GFS. This inspired the Nutch team to create the Nutch Distributed Filesystem (NDFS) in 2004. In 2004, Google published another paper on MapReduce, which the team implemented in Nutch.
- **Birth of Hadoop (2006):** The distributed computing and storage parts of Nutch were spun out to create an independent project called Hadoop. The name came from Cutting's son's yellow stuffed elephant.
- **Yahoo!'s Role:** Cutting joined Yahoo!, which provided a dedicated team and resources to turn Hadoop into a web-scale system. By February 2008, Yahoo! was running a 10,000-core Hadoop cluster for its search index.
- **Growth and Adoption (2008-Present):** Hadoop became a top-level Apache project in 2008. It gained fame for breaking the world record for sorting a terabyte of data and was adopted by major tech companies. Today, it is a mature technology used widely in mainstream enterprises and supported by major cloud providers and commercial vendors.

5 Conclusion: The Evolving Landscape

While Hadoop was revolutionary, the big data landscape continues to evolve.

- **The Rise of Spark and Flink:** For many use cases, especially those requiring iterative processing (like machine learning) or real-time stream processing, frameworks like Apache Spark and Apache Flink offer better performance than MapReduce.
- **Cloud Dominance:** A significant portion of new big data deployments are cloud-based. Managed services like Google BigQuery, Amazon Redshift, and Databricks offer serverless, auto-scaling platforms that abstract away the complexity of managing a Hadoop cluster.
- **Convergence:** The lines are blurring. Traditional RDBMS are incorporating ideas from distributed systems, while Hadoop/Hive are adding features like indexes and transactions to become more interactive and database-like.

Understanding the foundational principles of HDFS and MapReduce remains crucial, as they introduced the core concepts of distributed storage, fault tolerance, and data locality that underpin the entire modern big data ecosystem.