Real-Time and Big Data Analytics

Course Notes

Spring 2025

Your Name

March 11, 2025

Contents

Chapter 1

Introduction to Real-Time and Big Data Analytics

Chapter 2

Big Data Storage

2.1 Definition of Big Data

Big Data is when **the size of the data itself** becomes part of the problem.

Key Concept

One 10 TB hard disk drive (HDD) is less effective than 100 HDDs distributed across 20 computers due to **CPU limitations** when dealing with big data processing.

2.2 Big Data Storage Concepts

Big Data analytics uses **highly scalable distributed technologies and frameworks**. To store Big Data datasets, **often in multiple copies**, innovative storage strategies and technologies have been created to achieve **cost-effective** and **highly scalable** storage solutions.

Key concepts we'll explore include:

- Clusters
- Distributed file systems
- Relational database management systems (RDBMS)
- NoSQL databases
- Sharding
- Replication

- CAP theorem
- ACID
- BASE

2.3 Clusters

A cluster is a tightly coupled **collection of servers ("nodes")**. These servers usually have similar hardware specifications and are **connected together via a network** to work as a single unit.

Each node in the cluster has its **own dedicated resources**, such as CPU, memory, and storage.

A cluster can **execute a job** by **splitting it into small tasks** and **distributing their execution onto different computers** that belong to the cluster.

Cluster → Racks → Nodes (servers) → CPU, Memory, Storage

2.4 Distributed File Systems

A file is **the most basic unit of storage** to store data.

A file system (FS) is the method of **organizing files** on a storage device.

A distributed file system (DFS) is a file system that can **store large files spread across multiple nodes of a cluster**.

Examples: Google File System (GFS), Hadoop Distributed File System (HDFS), Amazon S3, and Azure Blob Storage.

2.5 Database Systems for Big Data

2.5.1 Relational Database Management Systems (RDBMS)

A relational database management system (RDBMS) is a product that presents a view of data as **a collection of rows and columns**.

SQL (structured query language) is the standard language for **querying** and **maintaining** the database.

A transaction symbolizes **a unit of work** that is performed against a database, and treated in a **coherent and reliable way** independent of other transactions.

2.5.2 NoSQL Databases

A NoSQL database is a **non-relational database** that provides a mechanism for **storage and retrieval of data** that is **highly scalable, fault-tolerant** and specifically designed to house **semi-structured** and **unstructured** data.

Examples of NoSQL database types:

- Key-Value stores (e.g., Redis)
- Document stores (e.g., MongoDB)
- Wide-Column stores (e.g., Cassandra)
- Graph databases (e.g., Pregel)

2.6 Distributed Data Management Strategies

2.6.1 Sharding

Sharding is the process of **horizontally partitioning** a large dataset into a collection of smaller, more manageable pieces called **shards**.

Each shard is stored on a **separate node**, and is responsible for **only the data stored on that node**.

All shards share **the same schema**, but each shard contains a **subset of the data**.

How Sharding Works in Practice

1. Each shard can **independently** service reads and writes for the **specific subset** of data that it is responsible for.
2. Depending on the query, data may need to be fetched from **multiple shards**.

Key Concept

Benefits of Sharding:

- Horizontal scalability: Sharding allows for **scaling out** by adding more nodes to the cluster, which can help to **distribute the load** and improve performance.
- Sharding provides **partial tolerance towards failure**.

Important Note

Concerns of Sharding:

- Queries requiring data from multiple shards can be **slower** and will impose performance penalties.
- To mitigate such performance penalties, **data locality** keeps commonly accessed data in the same shard.

2.6.2 Replication

Replication stores multiple copies of a dataset on multiple nodes.

Methods of Replication

- Master-slave replication
- Peer-to-peer replication

Master-Slave Replication

All data is written to a **master node**. Once saved, the data is replicated over to multiple **slave nodes**.

- **Write requests**, including insert, update, and delete, occur on the **master node**.
- **Read requests** can occur on either the **master node** or any of the **slave nodes**.

Master-slave replication is ideal for **read-intensive loads**. Growing read demands can be managed by **horizontal scaling** to add more slave nodes.

Writes are **consistent**:

- All writes are coordinated by the **master node**.
- However, **write performance will suffer** as the amount of writes increases.

If the **master node fails**:

- Reads are still possible from the **slave nodes**.
- Writes are **not possible** until a new master node is reestablished.

Recovery options:

- Resurrect the master node from a **backup**.
- Choose a new master node from the **slave nodes**.

Peer-to-Peer Replication

All nodes operate at the **same level**. Each peer is **equally capable** of handling read and write requests. Each **write** is copied to **all peers**.

> **Important Note**
>
> Concern: **Read/write inconsistency**

Strategies to address inconsistency:

- **Pessimistic concurrency** is a **proactive** strategy, using **locking** to ensure that only one update to a record can occur at a time. However, this is **detrimental to availability** since the database record being updated remains unavailable to other users until the lock is released.
- **Optimistic concurrency** is a **reactive** strategy that **does not use locking**. Instead, it **allows inconsistency** to occur with knowledge that **eventually** consistency will be achieved after all updates have propagated.

2.6.3 Combined Approaches for Data Distribution

- **Sharding and master-slave replication:** Each node acts both as a **master** and a **slave** for **different shards**.
- **Sharding and peer-to-peer replication:** Each node contains **replicas** of two different **shards**.

2.7 Theoretical Foundations and Design Principles

2.7.1 CAP Theorem

A distributed system may **wish** to provide three guarantees:

- **C**onsistency: A read from any node results in the **same, most recently written** data across multiple nodes.
- **A**vailability: A read/write request will **always be acknowledged** in the form of a success or failure, regardless of the state of the system.
- **P**artition tolerance: The database system can **tolerate communication outages** that split the cluster into multiple silos and can still **service read/write requests**.

Key Concept

The CAP theorem states that a distributed system can only provide **two** of the three guarantees at any given time.

2.7.2 Database Design Principles

ACID

ACID is a **traditional** database design principle for **transaction management**.

- **A**tomicity: Ensures that all transactions will always succeed or fail **completely**.
- **C**onsistency: Ensures that only data that **conforms to the constraints of the database schema** can be written to the database.
- **I**solation: Ensures that the results of a transaction are **not visible** to other transactions until the transaction is committed.
- **D**urability: Ensures the results of a transaction are **permanent**, regardless of any system failures.

Traditional databases leverage pessimistic concurrency controls (i.e., locking) to ensure ACID compliance. Database systems providing traditional ACID guarantees choose **consistency** over **availability**.

BASE

BASE is a database design principle leveraged by many **distributed** systems.

- **B**asically **A**vailable: The system is **always available** to service read-/write requests, even if the data is not consistent.
- **S**oft state: The system may be in a **temporary inconsistent state** at any given time, but will eventually become consistent.
- **E**ventual consistency: The system will **eventually become consistent** after all updates have propagated.

When a database supports BASE, it favors **availability** over **consistency**. BASE leverages optimistic concurrency by relaxing the strong consistency constraints mandated by the ACID properties.

ACID vs BASE Comparison

ACID ensures **immediate consistency at the expense of availability** due to record locking.

BASE emphasizes **availability over immediate consistency**.

Chapter 3

Distributed Computing Models

Chapter 4

Real-Time Analytics

Chapter 5

Big Data Visualization

Chapter 6

Case Studies and Applications

Appendix A

Glossary of Terms

Appendix B

Important Formulas and Theorems