Large Scale Information Storage and Retrieval – Detailed Study Guide

Lecture 01: Introduction

Key Challenges in Large-Scale Data Storage and Retrieval

1. **Volume** – Managing massive datasets that do not fit in memory.
2. **Velocity** – Handling real-time or near-real-time data ingestion.
3. **Variety** – Storing structured, semi-structured, and unstructured data.
4. **Scalability** – Expanding storage and retrieval capabilities without performance degradation.
5. **Consistency vs. Availability** – Trade-offs in distributed databases (CAP theorem).

Storage Models

- **Relational Databases (SQL-based)** – Use structured schemas (e.g., PostgreSQL, MySQL).
- **NoSQL Databases** – Designed for flexibility and scalability:
  - Key-Value Stores (e.g., Redis) – Simple, fast lookups.
  - Document Stores (e.g., MongoDB) – Store JSON-like documents.
  - Column-Family Stores (e.g., Apache Cassandra) – Optimized for big data analytics.
  - Graph Databases (e.g., Neo4j) – Store relationships effectively.

Module 02: Data Structure's Effects on Performance

Binary Search Trees (BSTs)

A BST is a tree where:

- Each node has up to two children.
- Left subtree contains keys smaller than the root.
- Right subtree contains keys larger than the root.

Operations & Time Complexity

Operation	Average Case	Worst Case
Search	O(log n)	O(n) (unbalanced)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

Problems with BSTs

- **Unbalanced BSTs degrade performance to O(n)**, behaving like linked lists.
- **BSTs do not guarantee efficient retrieval**, requiring balancing mechanisms.

AVL Trees (Self-Balancing BSTs)

- **AVL trees ensure O(log n) height** by enforcing balance conditions.
- Each node maintains a **balance factor** = (height of left subtree) **-** (height of right subtree).
- **Rotations** restore balance after insertion or deletion.

Types of Rotations

1. **Right Rotation (Single Rotation – LL Case)** – Performed when the left subtree is too tall.
2. **Left Rotation (Single Rotation – RR Case)** – Performed when the right subtree is too tall.
3. **Left-Right Rotation (LR Case)** – Combination of left and right rotation.
4. **Right-Left Rotation (RL Case)** – Combination of right and left rotation.

Advantages of AVL Trees

- Faster search times than unbalanced BSTs.
- Used where frequent lookups are needed.

B+ Trees (Used in Databases and File Systems)

B+ Trees are **multi-way self-balancing search trees** optimized for disk-based storage.

Properties of B+ Trees

- Each node has **multiple children** (degree d), unlike BSTs with only 2 children.

- **All values are stored in leaf nodes**, making range queries efficient.
- **Internal nodes store only keys** for routing, reducing search time.
- **Efficient disk access** – Reduces the number of disk reads compared to BSTs and AVL trees.

Operations

Operation	Time Complexity
Search	O(log_d N)
Insert	O(log_d N)
Delete	O(log_d N)

Why B+ Trees Are Used in Databases

- **Fast range queries** due to linked leaf nodes.
- **Disk-friendly** because fewer node accesses are needed.
- **Scalability** – Handles large datasets efficiently.

Module 03: Moving Beyond the Relational Model

Limitations of Relational Databases

1. **Scalability Issues** – Difficult to scale horizontally.
2. **Schema Rigidity** – Changing structure requires altering schemas.
3. **Performance Bottlenecks** – High latency for complex queries.

NoSQL Database Types

1. **Key-Value Stores** (e.g., Redis) – Simple and fast.
2. **Document Stores** (e.g., MongoDB) – Flexible JSON-like storage.
3. **Column-Family Stores** (e.g., Cassandra) – Best for analytical workloads.
4. **Graph Databases** (e.g., Neo4j) – Best for relationship-based data.

Module 05: NoSQL and Key-Value Databases

Key-Value Stores (Redis Example)

- **Data stored as (key, value) pairs**.

- **Operations**:
  - `SET key value` – Store a value.
  - `GET key` – Retrieve a value.
  - `DEL key` – Remove a value.
- **Used for caching, session storage, and real-time analytics**.

Module 06: Redis + Python

- **Use `redis-py` to connect Python applications to Redis**.

Common Python Redis commands:

```
import redis
r = redis.Redis(host='localhost', port=6379, db=0)
r.set('foo', 'bar')
print(r.get('foo'))  # Output: b'bar'
```

-

Module 07-08: MongoDB and PyMongo

MongoDB Basics

- Stores data as **JSON-like documents**.
- **No predefined schema**, allowing flexibility.

Using PyMongo in Python

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['testdb']
collection = db['testcollection']
collection.insert_one({"name": "Alice", "age": 25})
print(list(collection.find()))
```

- **CRUD Operations**:
  - **Insert**: `collection.insert_one({...})`
  - **Find**: `collection.find({...})`
  - **Update**: `collection.update_one({...})`

- ○ **Delete**: `collection.delete_one({...})`

Module 09: Graph Data Model

- **Graph databases store relationships explicitly** using nodes and edges.
- **Use cases**:
    - **Social networks** (friends, followers).
    - **Recommendation engines** (movie recommendations).
    - **Fraud detection** (detecting transaction anomalies).

Module 10: Docker Compose and Neo4j

- **Neo4j is a graph database** that uses the Cypher query language.
- **Docker Compose simplifies database deployments**.

Module 11-12: AWS (EC2 & Lambda)

Amazon EC2 (Elastic Compute Cloud)

- Provides virtual machines (VMs) in the cloud.
- **Use cases**: Running web servers, machine learning models.

AWS Lambda (Serverless Computing)

- Runs code **without managing servers**.

Example:

```
def lambda_handler(event, context):
   return "Hello from Lambda!"
```

-

Conclusion

- **Relational databases are structured but rigid**, while **NoSQL databases offer scalability and flexibility**.
- **B+ Trees are crucial for database indexing** because of their efficient disk access.
- **Redis is optimized for fast, in-memory key-value storage**.
- **MongoDB is a flexible, document-oriented database**.
- **Graph databases like Neo4j model relationships explicitly**, making them ideal for social and networked data.
- **AWS services (EC2, Lambda) provide scalable computing resources**.
