ollama dit 2
Architecture for Big Data

This document provides a comprehensive overview of key big data technologies: Hadoop, Spark (including
Spark Streaming), and Storm. It's structured as a study guide to help you understand their architectures, features, and differences.

Hadoop

HDFS (Hadoop Distributed File System)

- Distributed Storage:
  - Splits large files into smaller blocks (default size is 128 MB or 256 MB).
  - Each block is stored across multiple nodes in the cluster, with data replication to ensure fault tolerance.
- Fault Tolerance:
  - Uses replication factor to maintain multiple copies of each block on different nodes.
  - If a node fails, HDFS automatically re-replicates blocks from failed nodes to healthy ones.
- Scalability:
  - Designed for horizontal scaling; adding more nodes increases storage capacity and processing power.
- Data Locality:
  - Optimizes network bandwidth by scheduling tasks on nodes where data resides, reducing data transfer overhead.

MapReduce

- Map Phase:
  - Reads input data from HDFS and processes it to generate intermediate key-value pairs.
  - Each map task operates independently, allowing parallel processing.
- Shuffle Phase:
  - Intermediate key-value pairs are sorted and grouped by keys.
  - Data is transferred across nodes to ensure that all values for a particular key are sent to the same reducer.

- **Reduce Phase**:
  - Aggregates or processes intermediate data associated with each key.
  - Outputs the final result back to HDFS.
- **Fault Tolerance**:
  - Automatically retries failed tasks and re-executes them on different nodes if necessary.

YARN (Yet Another Resource Negotiator)

- **Resource Management**:
  - Centralized management of compute resources in clusters supporting various processing approaches.
- **Job Scheduling**:
  - Uses a scheduler to allocate cluster resources based on demand, capacity, and priority.
- **Node Manager & ResourceManager**:
  - Node Manager runs on each node, managing containers for tasks and monitoring resource usage.
  - ResourceManager coordinates resources among all applications in the system.

Spark

Differences and Similarities with Hadoop

- **In-Memory Processing**:
  - Spark's RDDs allow for data processing to be held in memory, drastically reducing I/O operations compared to Hadoop's
    disk-based approach.
- **Fault Tolerance**:
  - Hadoop relies on data replication across nodes.
  - Spark uses lineage information of RDDs to recompute lost data.
- **Use Cases**:
  - Hadoop is more suited for large-scale batch processing, like ETL tasks.
  - Spark excels in iterative algorithms and interactive queries due to its in-memory capabilities.

RDD Persistence

- **Resilient Distributed Datasets (RDDs)**:

- Immutable collections of objects partitioned across a cluster that can be cached in memory or on disk.

- Persistence Levels:
  - MEMORY_ONLY: Stores RDD in memory, recomputes if lost.
  - MEMORY_AND_DISK: Stores RDD in memory and spills to disk when necessary.
  - DISK_ONLY: Stores RDD only on disk.
  - OFF_HEAP: Uses off-heap storage for large datasets.

Broadcast Variables

- Data Distribution:
  - Broadcast variables allow the distribution of a large dataset efficiently across nodes without sending multiple copies to each task.
- Use Case:
  - Ideal for sharing lookup tables or model parameters with all worker nodes, reducing network overhead.

Accumulator Variables

- Aggregation:
  - Used for performing distributed aggregation operations like summing up values across a cluster.
- Fault Tolerance:
  - Only the driver program can modify accumulators, preventing race conditions in concurrent environments.

Cluster Mode

- Standalone Mode:
  - Spark runs as an independent service without requiring external resource managers.
- YARN Mode:
  - Integrates with Hadoop YARN for cluster management, leveraging existing Hadoop infrastructure.
- Mesos Mode:
  - Uses Apache Mesos for resource scheduling across a distributed system.

Storm

Architecture and Pipeline

- **Distributed Real-time Computation**:
  - Processes data streams in real time using a directed acyclic graph (DAG) of computation nodes called topologies.
- **Spouts and Bolts**:
  - **Spouts**: Sources of data streams; emit tuples to bolts for processing.
    - Examples include reading from message queues like Kafka or sockets.
  - **Bolts**: Perform operations on incoming stream data, such as filtering, aggregation, or joining with other streams.
- **Topology**:
  - Defines the flow and transformation of streams through spouts and bolts.
  - Topologies are configured declaratively using a directed graph where edges define the tuple flow between components.
- **Fault Tolerance**:
  - Provides guarantees like at-least-once processing semantics to ensure data reliability in case of failures.

This detailed explanation should provide a comprehensive understanding of Hadoop, Spark, and Storm architectures. Each technology
has unique strengths suited for different types of big data challenges.