ECE:569 - Database System Engineering

Final Project Paper

A Performance Comparison Hadoop Distributed File System vs Relational Database

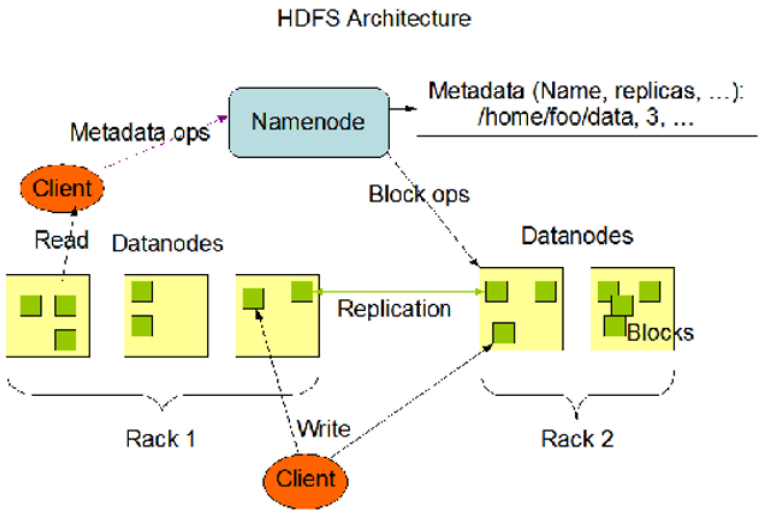
Victor Sankar Ghosh (vsg23)
Sai Revanth Tummala (st1109)

Introduction:

The management and processing of large-scale data have become critical challenges in modern data-driven environments. Hadoop Distributed File System (HDFS) and relational databases are two widely used solutions for storing and analyzing vast amounts of data.
This paper presents a comparative analysis of the performance characteristics of HDFS and relational databases, focusing on factors such as scalability, throughput, data processing speed, fault tolerance, and ease of use. Through a series of experiments and benchmarking tests, we evaluate the strengths and weaknesses of each system and provide insights into their suitability for different use cases. The findings of this study aim to assist organizations in making informed decisions regarding the selection of the appropriate data storage and processing solutions based on their specific requirements.

HDFS:

HDFS stands for Hadoop Distributed File System. It's a distributed file system designed to run on commodity hardware. HDFS is one of the core components of the Apache Hadoop software framework, which is widely used for distributed storage and processing of large data sets across clusters of computers.
HDFS is designed to be fault-tolerant and scalable. It stores data across multiple machines in a cluster, breaking large files into smaller blocks and replicating them across different nodes in the cluster to ensure data reliability and availability. This distributed nature allows HDFS to handle petabytes of data.
In addition to storing data, HDFS also provides mechanisms for data access and processing through the Hadoop ecosystem, including tools like MapReduce, Apache Spark, and Apache Hive, enabling users to perform various types of data analysis and processing tasks on large datasets stored in HDFS.



Components of HDFS:

Here are some of the main components of HDFS:

1. **NameNode:**
   The NameNode is the central metadata management component of HDFS. It stores the metadata information about the file system, including the directory structure, file names, permissions, and the mapping of data blocks to DataNodes. The NameNode maintains the namespace and the file-to-block mappings in memory for fast access. It is a single point of failure in HDFS, so it is crucial for the NameNode to be highly available and reliable.

2. **DataNode:**
   DataNodes are responsible for storing the actual data blocks of files. Each DataNode manages the storage attached to the machine it runs on and serves read and write requests from clients.
   DataNodes periodically send heartbeats and block reports to the NameNode to report their health status and the list of blocks they are storing. DataNodes can be added or removed dynamically from the cluster without interrupting the overall operation of the file system.

3. **Client:**
   Clients are the entities that interact with the HDFS cluster to read from or write to files. Clients can be any application or user program that uses the Hadoop Distributed File System API, such as Hadoop MapReduce, **Apache Spark**, or Hadoop Streaming.
   Clients communicate directly with the NameNode for metadata operations (e.g., file creation, deletion, listing) and with the DataNodes for data read and write operations.

4. **Block:**
   The fundamental unit of storage in HDFS is the block. HDFS breaks files into fixed-size blocks (typically 128 MB or 256 MB) and distributes these blocks across the cluster. Each block is replicated across multiple DataNodes for fault tolerance and reliability. The default replication factor is three, meaning each block is stored on three different DataNodes.

5. **MapReduce:**
   Mapper: Processes input data and generates intermediate key-value pairs.
   Reducer: Aggregates and processes intermediate key-value pairs based on keys.
   Partitioner: Determines how intermediate key-value pairs are distributed among reducers.

6. **YARN (Yet Another Resource Negotiator):**
   ResourceManager: Manages and allocates cluster resources to applications.
   NodeManager: Manages resources on individual nodes, executes containers, and monitors node health.
   ApplicationMaster: Negotiates resources with ResourceManager, manages execution of application tasks, and handles task failures.
   Container: Lightweight, isolated execution environment for running application tasks.

Apache Spark:

Apache Spark is an open-source, distributed computing framework that provides an efficient and flexible platform for large-scale data processing and analytics. It is designed to handle a wide variety of workloads, including batch processing, real-time stream processing, machine learning, and graph analytics, making it a versatile tool for data-driven applications.

Key features of Apache Spark include:

1. **Speed:** Spark is known for its fast and in-memory processing capabilities, which enable it to perform computations up to 100 times faster than traditional disk-based processing frameworks like Hadoop MapReduce. This speed is achieved by leveraging in-memory caching and optimization techniques.

2. **Ease of Use:** Spark provides a high-level API in multiple programming languages, including Scala, Java, Python, and R, making it accessible to a wide range of developers and data scientists. It also offers interactive shells for exploratory data analysis and development.

3. **Versatility:** Spark supports a wide range of data processing and analytics tasks, including batch processing, interactive queries, real-time stream processing, machine learning, and graph analytics. It provides dedicated libraries and APIs for each of these use cases, allowing users to perform complex computations with ease.

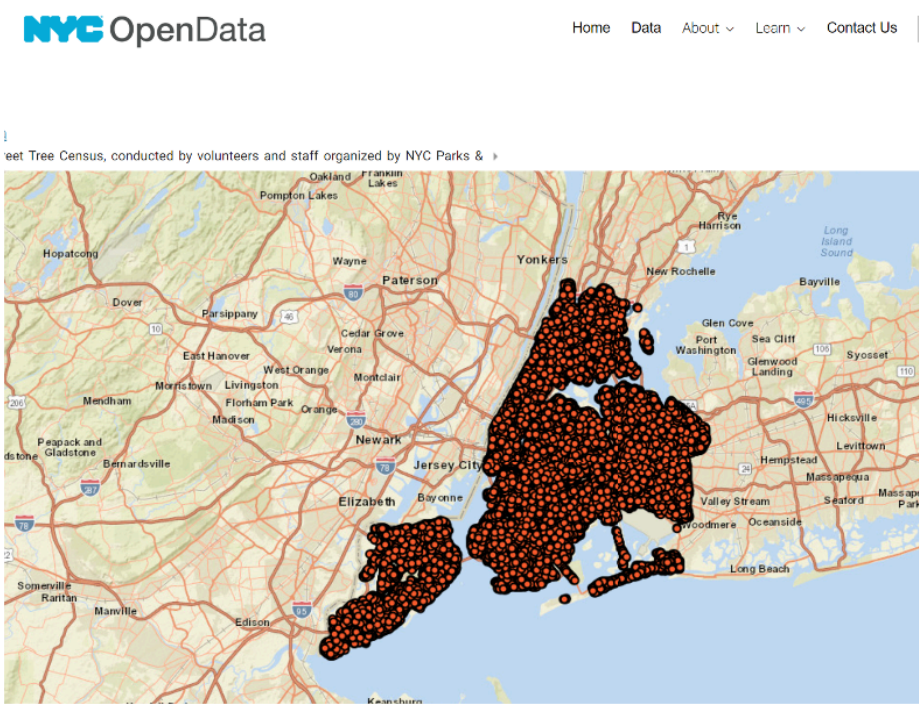
4. **Fault Tolerance:** Spark provides built-in fault tolerance mechanisms, such as lineage information and resilient distributed datasets (RDDs), which enable it to recover from failures gracefully without data loss. This ensures that computations are resilient to node failures and other system errors.

5. **Scalability:** Spark is designed to scale horizontally across large clusters of commodity hardware, allowing it to process petabytes of data efficiently. It achieves scalability through distributed data processing and task parallelism, enabling users to harness the full potential of their computing resources.

6. **Integration:** Spark seamlessly integrates with other components of the Hadoop ecosystem, including **HDFS**, YARN, and HBase, as well as with other data storage and processing systems like Amazon S3, Apache Kafka, and Apache Cassandra. This integration allows users to leverage existing data infrastructure and tools within their Spark workflows.

Dataset:

[NYC 2015 Street Tree Census Report](#) - The size of this data is **236MB** and has **683K** rows.



Performance Comparison (HDFS vs Relational Database):

	Loading Data to DB	Read Data	Update Data	Truncate Data
HDFS	6.163386 sec	3.630672 sec	1:08.25047 mins	0.29 secs
Relational Database	15.359 sec	3.734 sec	1:19.203 mins	0.031 sec

Weather-Dataset-US - The size of this data is **8.37GB** and has **155M** rows.

This is just an additional dataset we considered and did some performance evaluation.

No	SQL Query	MySQL (in sec)	PySpark (in sec)
1	SELECT COUNT(DISTINCT ID) FROM weather	163.79	37.57

Conclusion:

In conclusion, the comparative analysis between the Hadoop Distributed File System (HDFS) and relational databases has shed light on the strengths, weaknesses, and suitability of each solution for different data storage and processing scenarios.

HDFS, with its distributed architecture and fault-tolerant design, excels in handling large volumes of data across clusters of commodity hardware. Its scalability, reliability, and integration with the broader Hadoop ecosystem make it well-suited for batch processing, data warehousing, and analytics use cases. However, HDFS may face challenges in scenarios requiring low-latency access to structured data or complex query processing.

On the other hand, relational databases offer a mature and feature-rich solution for managing structured data with support for transactions, SQL queries, and ACID properties. Relational databases are preferred for applications with strict consistency requirements, real-time data processing needs, and complex query workloads. However, they may encounter scalability limitations when dealing with massive datasets or distributed processing tasks.

The choice between HDFS and relational databases ultimately depends on the specific requirements and constraints of the application, including data volume, access patterns, performance goals, and existing infrastructure. Organizations must carefully evaluate the trade-offs and considerations associated with each solution before making an informed decision.

Looking ahead, advancements in distributed computing technologies, cloud-native architectures, and hybrid data management approaches are likely to shape the future of data storage and processing. Solutions that offer the best of both worlds, combining the scalability of HDFS with the flexibility of relational databases, are expected to gain prominence in the evolving data landscape.