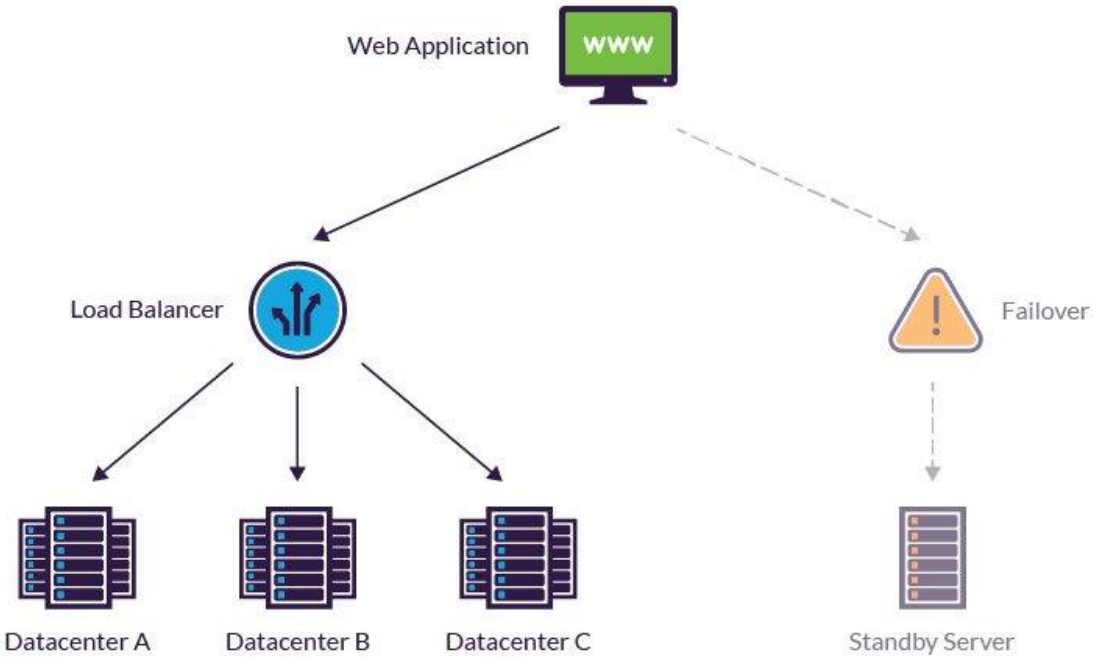
Experiment 10: Case Study on Fault Tolerance in Distributed Systems.

Learning Objective : Student should be able to examine how systems like Apache Hadoop and Apache Spark handle node failures and ensures the reliability of Computations **.**

Theory: Distributed systems are defined as a collection of multiple independent systems connected together as a single system. Every independent system has its own memory and resources and some common resources and peripheral devices that are common to devices connected together. The design of Distributed systems is a complex process where all the nodes or devices need to be connected together even if they are located at long distances. Challenges faced by distributed systems are Fault Tolerance, transparency, and communication primitives. Fault Tolerance is one of the major challenges faced by distributed systems.

Introduction: Distributed systems have revolutionized the way we process vast amounts of data, enabling parallelized computations across clusters of machines. However, as the scale of these systems increases, so does the likelihood of node failures. This case study delves into the intricate strategies employed by two leading distributed computing frameworks, Apache Hadoop and Apache Spark, to handle node failures and ensure the reliability of computations in large-scale environments.

System Architecture: Load balancing solutions enable applications to operate across multiple network nodes, mitigating the risk of a single point of failure. They optimize workload distribution, enhancing individual resilience to activity spikes and preventing slowdowns. In contrast, failover solutions are crucial during severe scenarios, like complete network failures. They automatically activate a secondary platform to keep applications running while the primary network is restored. For fault tolerance with zero downtime, "hot" failover instantly transfers workloads to a backup system. Alternatively, "warm" or "cold" failover involves a backup system with a delayed workload start-up.

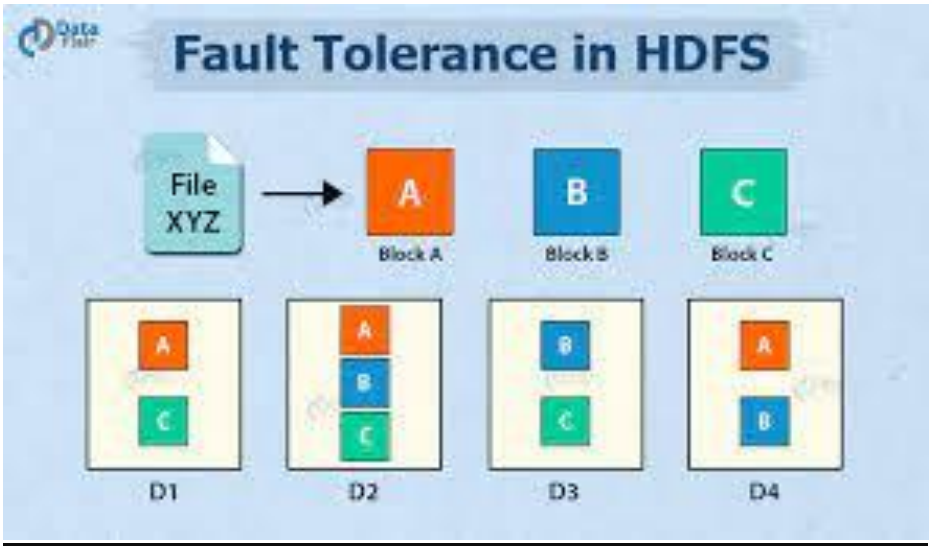


Background of Hadoop and Spark:

- Apache Hadoop: Known for its MapReduce programming model and Hadoop Distributed File System (HDFS), Hadoop divides large datasets into smaller chunks, processes them in parallel across a cluster, and stores them redundantly for fault tolerance.
- Apache Spark: Built on top of Hadoop, Spark introduces an in-memory data processing engine. It provides fault tolerance through lineage information, enabling the reconstruction of lost data partitions in case of node failures.

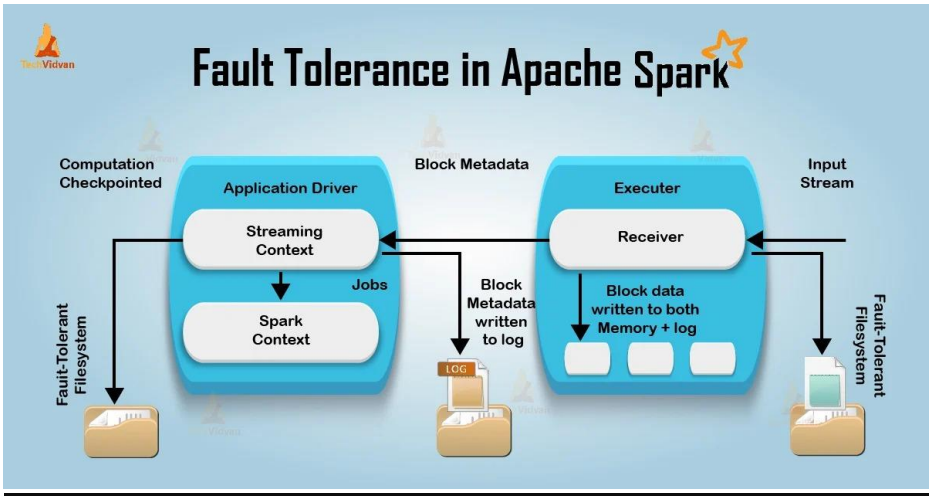
Fault Tolerance in Apache Hadoop:

- Data Replication: Hadoop achieves fault tolerance by replicating data across multiple nodes in the HDFS. Typically, data is replicated three times, and tasks are rerouted to replicas if a node fails during computation.
- Task Redundancy: MapReduce jobs in Hadoop consist of map and reduce tasks. Hadoop monitors task execution and reruns failed tasks on available nodes, ensuring the completion of the job.



Fault Tolerance in Apache Spark:

- Resilient Distributed Datasets (RDDs): Spark introduces RDDs, a fault-tolerant abstraction representing a distributed collection of data. RDDs maintain lineage information, allowing lost partitions to be recomputed from the original data.
- Directed Acyclic Graph (DAG): Spark jobs are represented as a directed acyclic graph (DAG) of stages. If a node fails, Spark can recompute only the affected partitions by referencing the lineage information in the DAG, minimizing the computational overhead.



Handling Node Failures:

- Hadoop: Upon node failure, Hadoop's ResourceManager reallocates tasks to healthy nodes. However, if the failed node contains data blocks, HDFS replicates the data to other nodes, ensuring availability.
- Spark: Spark's Driver and Cluster Manager monitor task execution. In case of node

failure, Spark recomputes lost partitions based on lineage information, ensuring the fault tolerance of computations.

Resource Management:

- Hadoop: Hadoop relies on a master-slave architecture, with the ResourceManager managing resources and the NodeManagers overseeing individual nodes.
- Spark: Spark utilizes a similar master-slave architecture with a Driver coordinating tasks and Executors managing node-level computations.

Comparative Analysis:

- Data Processing Speed: Spark generally outperforms Hadoop due to its in-memory processing capabilities, reducing the need for extensive disk I/O.
- Fault Tolerance Overheads: While both systems provide fault tolerance, Spark's lineage-based approach minimizes recomputation overhead compared to Hadoop's task reexecution.

Real world applications:

- Hadoop: Widely used for batch processing tasks, such as log analysis and data warehousing. Log Analysis: Suppose a large e-commerce platform wants to analyze its server logs to gain insights into user behavior. Hadoop can be employed to process and analyze these logs in batches, extracting valuable information such as popular products, user traffic patterns, and potential issues in the system.
- Spark: Suitable for iterative algorithms, machine learning, and interactive analytics, where in-memory processing is advantageous. Interactive Analytics: Consider a business intelligence scenario where an analyst needs to interactively explore and analyze data to make real-time decisions. Spark can be used for interactive analytics, allowing users to run queries, visualize data, and obtain insights on-demand. This is particularly beneficial in scenarios like marketing campaign performance analysis or real-time dashboards for monitoring.

Future considerations and Hybrid Approaches:

- The evolution of distributed computing is a dynamic process, and as technology advances, future considerations become pivotal for enhancing the efficiency and robustness of distributed systems.
- Two key areas of exploration involve the potential integration of hybrid approaches, combining the strengths of Apache Hadoop and Apache Spark, and the impact of advancements in hardware and network technologies on fault tolerance mechanisms.

The current landscape presents Apache Hadoop and Apache Spark as powerful yet distinct frameworks, each excelling in specific use cases.

- Future exploration may revolve around developing hybrid approaches that amalgamate the strengths of both frameworks. This could entail leveraging Hadoop's mature and reliable storage infrastructure while harnessing Spark's in-memory processing capabilities for enhanced computational speed. Hybrid architectures could be designed to dynamically allocate tasks based on the nature of computations. For instance, batch processing tasks might leverage Hadoop's efficiency, while more iterative and interactive workloads could benefit from Spark's faster in-memory processing.
- The seamless integration of these frameworks may require innovations in job scheduling, resource management, and data orchestration to optimize the overall performance of distributed systems.

Conclusion:

FOR FACULTY USE

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				