Batch Processing

Casper Kristiansson, Nicole Wijkman (Group Therapy)

01

What is Batch Processing

Introduction to Batch Processing

Definition

Batch processing is a powerful and efficient solution that processes large volumes of data in batches without real-time interactions.

How does it work?

- Accumulates tasks over a specific time
- Executes tasks during periods of low computational demand
- Contrasts real-time processing which requires immediate analysis

Why is it important?

- Handles large datasets with minimal effect to operational delay
- While not offering real-time analysis, it's widely adopted across industries due to its efficiency

Basics & Fundamentals

Core Principles	Key Components
Volume Orientation: Manages data or transactions, using strategies like concurrent execution and task association. **Non-Interactivity:** Operates autonomously without real-time user interaction. Deferred Execution: Schedules jobs during low computational periods for optimized resource utilization.	**Jobs:** A single unit of work or task **Batches:** One or more jobs meant to be processed together. **Job Schedulers:** Determine the execution order and manage dependencies. **Resource Management:** Allocation of computational resources for efficiency. Applications: **Finance:** Managing transactions and data analysis. **Healthcare:** Handling tasks from billing to data analyses.

02

MapReduce: Simplified Data Processing on Large Clusters

MapReduce - Addressing Large-Scale Data Processing Challenges

Definition

MapReduce, presented by Jeffrey Dean and Sanjay Ghemawat, offers a simplified approach to parallelized large-scale data processing.

Challenges in Large-Scale Data Processing

- Managing petabytes or even exabytes of data.´
- Infrastructure limitations, including hardware and software constraints
- Complexities in parallel and distributed computing: synchronization, data consistency, data partitioning, and error handling
- Ensuring system reliability and fault tolerance in an environment where failures are common

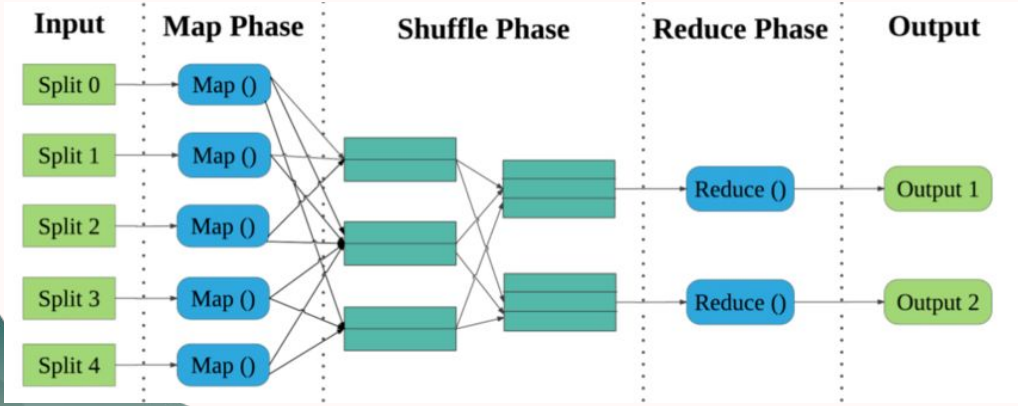
MapReduce Programming Model Simplification

Key Features

- Developers only need to specify two functions: Map and Reduce
  - **Map**: Processes input data to produce intermediate key-value pairs
  - **Reduce**: Aggregates these pairs to produce the final result
- A shuffling phase rearranges intermediate pairs to ensure keys are processed by the same reducer

Benefits

- Developers focus solely on computation, not system details
- MapReduce abstractly handles partitioning, scheduling, communication, and fault tolerance
- Adaptable to a wide variety of data types, structured or unstructured



Automatic Management and Scalability with MapReduce

Key Automations

- Dynamic task assignment: Input data is divided into chunks, processed in parallel
- Automated fault tolerance and recovery: Failed tasks are re-executed, and any intermediate data can be recovered from disk storage
- Optimized data placement: Minimized data movement across networks, and data is processed where stored

Scalability and Applicability

- Proven capability to process petabytes of data
- Efficient scaling across computing nodes
- Diverse applications: Web search, graph processing, and distributed sorting
- Demonstrated robustness against task and node failures, ensuring system stability and reliability

03

Resilient Distributed Datasets: A Fault Tolerant Abstraction for In Memory Cluster Computing

Resilient Distributed Datasets (RDDs)

Definition

RDDs, as presented by Matei Zaharia et al., offers a new solution for challenges in large-scale data processing & distributed computing.

Current Frameworks

Several inherent inefficiencies in existing distributed computing frameworks.

These inefficiencies become especially clear when reusing data across multiple computations.

Fault Tolerance

There is a need for robust fault tolerance mechanisms and a versatile programming model.

Ensuring system reliability is crucial as large-scale clusters become more complex and the data volume grows.

Streamlining

Key motivation can be summarised as the process of **streamlining data processing in large-scale scale cluster computing environments.**

Key Contributions of RDDs

An efficient data-sharing abstraction for in-memory cluster computing

Key Features

- RDDs serve as a fundamental building block for in-memory cluster computing.

- Addresses the challenge of reusing data across multiple computations.

- Created through coarse-grained transformations on distributed datasets
  - Example of transformations: **map**, **filter**, **join**

Benefits

- Offers lineage for efficient data recovery in the case of node failures.

- Improved Fault-Tolerance: Lineage-based recovery; minimizing the need for replication & disk I/O.

- Diverse Programming Model Support: Supports MapReduce, Dryad, Pregel, etc.

Strengths, Weaknesses, & Final Thoughts

What are the strengths and weaknesses of the paper, and what conclusion is drawn?

Strengths

- Significant contribution with RDDs.
- Supports diverse programming models.
- Open-sourced Spark system for collaboration.

Weaknesses

- Learning curve for RDDs.
- Potential limitations in handling specific algorithms.
- Unclear adaptability to dynamic environments.

Conclusion

Despite the few weaknesses, the paper and the concept of RDDs is a very valuable contribution to distributed computing with comprehensive evaluations.

Thank you!

Group Therapy

Casper Kristiansson, Nicole Wijkman