Spark and Dataflow Programming: Comprehensive Review

DSC 208R - Data Management for Analytics

June 2025

Apache Spark

Apache Spark is a widely adopted dataflow programming model that subsumes most of relational algebra and MapReduce functionalities. It was inspired by the Python Pandas style of chaining functions and aims to provide a unified platform for various data processing tasks.

Key Innovations and Features

- **Unified Storage**: Handles relations, text, and other data types uniformly, allowing for custom programs.
- **In-Memory Computing**: A key idea differentiating Spark from Hadoop's MapReduce is its ability to exploit distributed memory to cache data, significantly speeding up iterative algorithms and interactive queries.
- **Lineage-Based Fault Tolerance**: A novel approach to fault tolerance. Instead of replicating data extensively (like HDFS) or recomputing the entire job on failure, Spark keeps a "lineage" graph of transformations applied to data. If a partition of an RDD (Resilient Distributed Dataset) is lost, it can be recomputed from its parent RDDs.
- **Open-Source and Commercialization**: Open-sourced to Apache and later commercialized by Databricks.

Distributed Architecture of Spark

Spark operates with a manager-worker architecture.

- **Driver Program**: The main program that runs on a single node and creates the 'SparkContext'. It defines the transformations and actions, coordinates tasks, and distributes them to workers.
- **Cluster Manager**: External service (e.g., YARN, Mesos, Kubernetes, or Spark's own standalone manager) that acquires resources on worker nodes.
- **Worker Node**: A node in the cluster where computations are performed.
- **Executor**: A process running on a worker node that performs tasks and stores cached data. Each worker node can run multiple executors.
- **Cache**: Executors can cache RDD partitions in memory for faster reuse.
- **Task**: The smallest unit of work sent to an executor.

Spark's Dataflow Programming Model

Spark processes data through a series of transformations and actions on Resilient Distributed Datasets (RDDs), DataFrames, or Datasets. This model supports lazy evaluation.

Transformations vs. Actions

- **Transformations**: Operations that create a new RDD from an existing one. They are *lazy*, meaning they don't execute immediately but instead build a logical plan (lineage graph). Examples include 'map()', 'filter()', 'flatMap()', 'union()', 'join()', 'groupByKey()', 'reduceByKey()', 'sortByKey()'.
- **Actions**: Operations that trigger the execution of the transformations (the lineage graph) and return a result to the driver program or write data to an external storage system. Examples include 'collect()', 'count()', 'reduce()', 'saveAsTextFile()'.

Types of RDD Operations (Narrow vs. Wide)

- **Narrow Transformations**: Each partition of the parent RDD contributes to at most one partition of the new RDD. This means no data shuffle across the network is required. Examples: 'map()', 'filter()'. This is highly efficient.
- **Wide Transformations**: Each partition of the parent RDD may contribute to multiple partitions of the new RDD. This requires a shuffle of data across the network (e.g., a "shuffle-and-sort" operation in MapReduce context). Examples: 'groupByKey()', 'reduceByKey()', 'sortByKey()', 'join()'. These are generally more expensive due to network I/O.

The DAGScheduler in Spark optimizes the execution by combining narrow transformations into stages, reducing shuffles.

Dataflow Systems vs. Task-Parallel Systems

- **Dataflow Systems (e.g., MapReduce, Spark)**: Primarily designed for batch processing of large datasets. They excel at operations that can be parallelized by partitioning data and applying the same operation to each partition. The focus is on data movement and transformations on that data.
- **Task-Parallel Systems (e.g., Dask, Ray)**: More general-purpose, designed for arbitrary computations that can be broken down into independent tasks, which may or may not operate on large datasets. They excel at orchestrating complex workflows with dependencies between tasks, where data movement might not be the primary concern.

Dataflow systems often rely on a manager-worker architecture for coordination.

New Paradigm of Data "Lakehouse"

- **Data Lake**: A concept of loosely coupled data file formats and data/query processing stacks, in contrast to the tight coupling in RDBMSs. It allows for various frontends (tools/engines) to interact with diverse data formats stored in a centralized repository (often a distributed file system like HDFS or cloud object storage).
- **Data Lakehouse**: An evolution of the data lake, aiming to combine the flexibility and scalability of data lakes with the data management features (e.g., transactions, schema enforcement, data quality) traditionally found in data warehouses. It tries to offer the best of both worlds, enabling both analytics and machine learning workloads on the same data.