> Bases de Datos 2, Resumen 3
> Miguel Ku Liang - 2019061913

Apache Spark: A Unified Engine for Big Data Processing

As data sizes have outplaced the capabilities of machines, users have needed new systems to scale out computations to multiple nodes. There has been an explosion of new cluster programming models targeting diverse computing workloads. Spark has a programming model similar to MapReduce, but extents it with Resilient Distributed Datasets. Applications are easier to develop because they use a unified API and it is more efficient to combine processing tasks. People have built an integrated standard library over Spark, with functions from data import to machine learning. This libraries have been used in many applications.

Programming Model

The programming abstraction is RDDs. Users create RDDs by applying operations called transformations to data. Spark evaluates RDDs lazily. When an action is called, Spark looks at the graph of transformations used to create an execution plan. RDDs provides explicit support for data sharing among computations. Users can persist selected RDDs in memory. This data sharing provides large speedups for queries and iterative algorithms.

Fault tolerance: RDDs automatically recovers from failures. Spark uses lineage. It tracks the graph of transformations to rerun the operations to reconstruct the lost partitions.
Integration with storage systems: Spark is designed to be used with multiple external systems for persistent storage. Spark's design as a storage-system-agnostic engine makes it easy for users to run computations against existing data and join diverse data sources.

Higher-Level Libraries

The RDD provides only distributed collections of objects and functions to run on them. Higher-level libraries targets many use cases of specialized computing engines.

- SQL and Dataframes: Spark SQL and Shark implements queries on Spark using techniques similar to analytical databases, compressed columnar storage. The Spark SQL engine provides a higher-level abstraction for basic data transformations called Dataframes. Dataframes are a common abstraction for tabular data in R and Python.
- Spark Streaming: implements incremental stream processing using discretized streams. Fault recovery is less expensive due to using lineage.
- GraphX: it provides a graph comuptation interface similar to Pregel and GraphLab, implementing the same placement optimizations.
- MLlib: it is Spark's machine learning library, implements algorithms for distributed model training.
- Combining processing tasks: Spark's libraries all operate on RDDs as the data abstraction, making them easy to combine in applications. Spark can optimize across processing libraries.
- Performance: it can match the performance of specialized engines.

Applications

Spark is used in areas from Web services to biotechnology to finance.

- Batch processing: Spark is used for batch processing on large datasets like ETL workloads.
- Interactive queries: the use of Spark falls into three main classes. Organizations use Spark SQL for relational queries, developers and data scientists can use Spark's Scala, Python and R interfaces through visual notebook environments, and several vendors have developed domain-specific interactive applications that run on Spark.
- Stream processing: real-time precessing is a popular use case in analytics and real-time decision making applications.
- Scientific applications: Spark has been used in several scientific domains like large-scale spam detection, image processing and genomic data processing.
- Spark components used: organizations use multiple components, over 60% at least using three of Spark's APIs.
- Deployment environments: there is growing diversity in where Spark applications run and what data sources they connect to.

Why Is the Spark Model General?

- Expressiveness perspective: MapReduce can simulate many computations in the Parallel Random Access Machine model. Repeated MapReduce is equivalent to the Bulk Synchronous Parallel model. There are two problems: MapReduce is inefficient at sharing data across timesteps because it relies on replicated external storage systems for this purpose and the latency of the MapReduce steps determines how well the emulation will match a real network. RDDs and Spark address these limitations. RDDs make data sharing fast by avoiding replication of intermediate data and can closely emulate the in-memory data sharing across time. Spark can run MapReduce-like steps on large clusters with 100ms latency.
- Systems perspective: current datacenters have a steep storage hierarchy that limits most applications. The most important performance concern in many applications is the placement of data and computation in the network. RDDs provide the facilities to control this placement. Spark can run the same algorithms and libraries used in specialized systems on each node to reduce CPU bottleneck.

Spark may incur extra costs over some of today’s specialized systems due to fault tolerance.

Ongoing Work

- Dataframes and more declarative APIs: Spark API was based on functional programming over distributed collections that contain arbitrary types of Scala, Java, or Python objects. It made programs more difficult to analyze and optimize. Adding dataframes based on the relational algebra would address the problem. Spark’s dataframes offer a similar API to single-node packages but automatically parallelize and optimize the computation using Spark SQL’s query planner. Datasets lets Java and Scala programmers view dataframes as statically typed collections of Java objects and receive relational optimizations.
- Performance optimizations: Project Tungsten removes Java Virtual Machine overhead from Spark’s code paths by using code generation and non-garbage-collected memory. It affected all Spark's libraries, so machine learning, streaming and SQL became faster.
- R language support: SparkR project provides a programming interface in R. It is based on dataframes.
- Research libraries: these libraries have been merged into the main codebase.