Isaac Araya Solano | Carnet: 2018151703

Resumen #3

Apache Spark: A unified engine for data processing

The growth of data volumes brings great opportunities as well as great challenges. The data sizes have outspaced the capabilities of single machines, making cluster programming models explote. There have been some solutions as Dremel, Storm or Impala. Even with relational databases the trend has been to move away from "one-size-fits-all" systems.

In 2009 the project of Apache Spark was borned in the University of California. Spark has a programming model similar to MapReduce but with data-sharing abstraction called Resilient Distributed Datasets or RDDs. Using this they can unify processes that before required multiple engines. Those engines are replaced by libraries in a common engine, Spark, making them easy to compose.

Spark's generality has a lot of benefits:

- Apps developed with it are easier to develop since they use a unified API
- It is more efficient to combine processing tasks
- Spark can run diverse functions over the same data
- Enables new applications and streaming machine learning.

One powerful analogy is to compare this unification to a smartphone and all the separated devices that existed before like cameras, cellphones, GPS and others.

Spark has grown to be the most active open source project or big data processing. As parallel data processing becomes common, the composability is one of the main concerns for usability and performance.

Programming Model

The key are RDDs, which are fault-tolerant collections of objects partitioned across a cluster. RDDs are created by transformations(map, filter, groupBy). RDDs are exposed by APIs in different languages. Spark can fuse multiple filter or map operations into one pass. RDDs provide explicit support for data sharing among computations. RDDs are usually ephemeral, however, you can make them persistent in memory for rapid reuse. The data sharing features are the difference between spark and other models like MapReduce. Fault tolerance is another important feature since RDDs automatically recover from failures. RDDs use an aproach called "lineage" that works by making the RDD rerun the transformations that are tracked by the RDD to reconstruct any lost partitions. Lineage is significantly more efficient than replication, saving time and storage.

Higher level Libraries

The RDD programming model provides distributed collections of objects and functions to run on them, but there are also a lot of high level libraries for specific functions.

The 4 main libraries included by Spark:

- SQL and Dataframes: implements SQL Queries on Spark.
- Spark Streaming: implements incremental stream processing using a model called "dicretized streams"

- GraphX: provides a graph computation interface.
- MLlib: spark machine learning library. These libraries provide the ability to combine processing tasks and have higher performance since they run in the same engine.

Applications

More than 1000 companies use Spark, in areas from Web Services to biotechnology to finances. Here, we cover a few top use cases:

- Batch processing
- Interactive Queries
- Stream processing
- Scientific Applications
- Spark components used
- Deployment Environments

Why is the Spark Model General?

Expressiveness perspective: The study compares RDDs to MapReduce and asks what computations MapReduce can express. Despite MapReduce's limitations, it can emulate any distributed computation by using the map operation for local computation and reduce for all-to-all communication. MapReduce can emulate any distributed computation by breaking down work into time steps and using maps for local computation and reduce for communication. However, this emulation can be inefficient and slow. RDDs and Spark address these issues by making data sharing fast and reducing latency, allowing for more efficient emulation of distributed computations. RDDs build on MapReduce's ability to emulate computations while improving efficiency, but increased latency due to synchronization can still be a limitation.

Systems perspective: In analyzing Spark's generality, one can take a systems approach to identify the bottleneck resources in cluster computations and determine if RDDs are utilizing them efficiently. Though diverse, cluster applications are limited by the same underlying hardware properties, such as storage hierarchy. RDDs offer facilities for controlling data and computation placement in the network, as well as implementing placement strategies used in specialized systems. The most common bottleneck tends to be CPU time, which Spark addresses by running the same algorithms and libraries used in specialized systems on each node. Fault tolerance in Spark may incur extra costs over specialized systems, but Spark often performs competitively despite them, since many applications are bound by an I/O operation, beyond which optimizations add only modest benefits.

Ongoing work

Apache Spark is a popular distributed computing project that has grown significantly since 2013. The project has seen contributions from both industry and research, with more than 200 third-party packages available. To address issues related to suboptimal performance, the project introduced a more declarative API called DataFrames, based on the relational algebra. This API has quickly become popular, with 60% of respondents in a 2015 survey reporting its use. Performance optimizations have also been a major focus, with recent efforts such as Project Tungsten reducing Java Virtual Machine overhead. Additionally, the SparkR project was merged into Spark in 2015 to provide a programming interface in R, and the project has been used to build higher-level data processing libraries for neuroscience, genomics, and image processing in astronomy, among others.