

Big data: architectures and data analytics

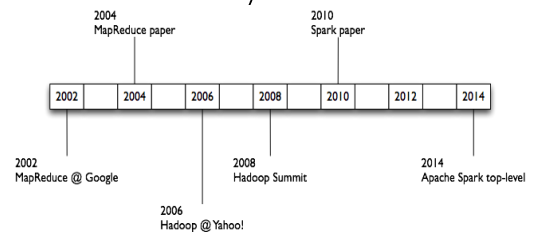
Introduction to Spark

Spark

- Apache Spark™ is a fast and general engine for large-scale data processing
- Spark aims at achieving the following goals in the Big data context
 - Generality: diverse workloads, operators, job sizes
 - Low latency: sub-second
 - Fault tolerance: faults are the norm, not the exception
 - Simplicity: often comes from generality

Spark History

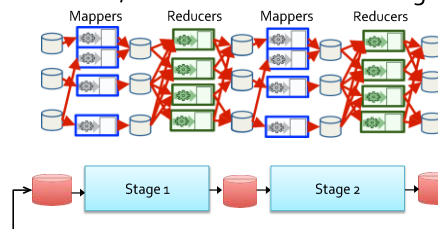
- Originally developed at the University of California - Berkeley's AMPLab



Spark: Motivations

MapReduce and Iterative Jobs

- Iterative jobs, with MapReduce, involve a lot of disk I/O for each iteration and stage



MapReduce and Iterative Jobs

- Disk I/O is very slow (even if it is a local I/O)

The diagram illustrates the MapReduce process. At the top, it shows a sequence of Mappers and Reducers. Mappers read data from disks and write it to Reducers. Reducers then process the data and write it back to disks. Below this, a flow diagram shows 'Stage 1' and 'Stage 2' as blue boxes. Red cylinders representing data disks are connected to these stages, showing the flow of data between them.

Apache Spark: Motivation and Opportunity

- Motivation
 - Using MapReduce for complex **iterative jobs** or **multiple jobs on the same data** involves lots of disk I/O
- Opportunity
 - The **cost of main memory decreased**
 - Hence, large main memories are available in each server
- Solution
 - Keep **more data in main memory**
 - Basic idea of Spark

From MapReduce to Spark

- MapReduce: Iterative job

The diagram shows an iterative MapReduce job. It starts with an 'Input' disk. 'Iteration 1' involves an 'HDFS read' from the input and an 'HDFS write' to a green disk. 'Iteration 2' involves an 'HDFS read' from the green disk and an 'HDFS write' to another green disk. This pattern repeats, showing significant disk I/O between iterations.

From MapReduce to Spark

- Spark: Iterative job

The diagram shows an iterative Spark job. It starts with an 'Input' disk. 'Iteration 1' involves an 'HDFS read' from the input and data being stored in main memory (represented by RAM icons). 'Iteration 2' involves reading data from the main memory and writing it back to main memory. This shows that data is shared between iterations without needing to be written to disk.

- Data are shared between the iterations by using the main memory
 - Or at least part of them
- 10 to 100 times faster than disk

From MapReduce to Spark

- MapReduce: Multiple analyses of the same data

The diagram shows multiple analyses of the same data in MapReduce. An 'Input' disk is read multiple times by 'query 1', 'query 2', and 'query 3'. Each query produces its own 'result' (result 1, result 2, result 3). This illustrates the inefficiency of reading the same data from disk multiple times.

From MapReduce to Spark

- Spark: Multiple analyses of the same data

The diagram shows multiple analyses of the same data in Spark. An 'Input' disk is read once, and the data is stored in 'Distributed memory' (RAM icons). 'query 1', 'query 2', and 'query 3' all access the data from the distributed memory to produce 'result 1', 'result 2', and 'result 3' respectively. This shows that data is read only once and shared across queries.

- Data are read only once from HDFS and stored in main memory
 - Split of the data across the main memory of each server

Spark: Resilient Distributed Data sets (RDDs)

- Data are represented as Resilient Distributed Datasets (RDDs)
 - Partitioned/Distributed collections of objects spread across the nodes of a clusters
 - Stored in main memory (when it is possible) or on local disk
- Spark programs are written in terms of operations on resilient distributed data sets

Spark: Resilient Distributed Data sets (RDDs)

- RDDs are built and manipulated through a set of parallel
 - Transformations
 - map, filter, join, ...
 - Actions
 - count, collect, save, ...
- RDDs are automatically rebuilt on machine failure

Spark Computing Framework

- Provides a programming abstraction (based on RDDs) and transparent mechanisms to execute code in parallel on RDDs
 - Hides complexities of fault-tolerance and slow machines
 - Manages scheduling and synchronization of the jobs

MapReduce vs Spark

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc...
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

MapReduce vs Spark

- Lower overhead for starting jobs
- Less expensive shuffles

In-Memory RDDs Can Make a Big Difference

- Two iterative Machine Learning algorithms:



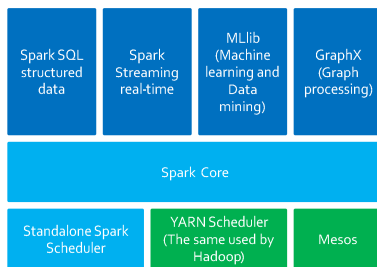
Petabyte Sort Challenge

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

Daytona Gray 100 TB sort benchmark record (tied for 1st place)

Spark: Main components

Spark Components



Spark Components

- Spark is based on a basic component (the Spark Core component) that is exploited by all the high-level data analytics components
 - This solution provides a more uniform and efficient solution with respect to Hadoop where many non-integrated tools are available
- When the efficiency of the core component is increased also the efficiency of the other high-level components increases

22

Spark Components

- Spark Core
 - Contains the basic functionalities of Spark exploited by all components
 - Task scheduling
 - Memory management
 - Fault recovery
 - ...
 - Provides the API that are used to create RDDs and apply transformations and actions on them

23

Spark Components

- Spark SQL structured data
 - This component that is used to interact with structured datasets by means of SQL
 - It supports also
 - Hive Query Language (HQL)
 - It interacts with many data sources
 - Hive Tables
 - Parquet
 - JSON

24

Spark Components

- Spark Streaming real-time
 - It is used to process live streams of data in real-time
 - The APIs of the Streaming real-time components operated on RDDs and are similar to the ones used to process standard RDDs associated with "static" data sources

25

Spark Components

- MLlib
 - It is a machine learning/data mining library
 - It can be used to apply the parallel versions of many machine learning/data mining algorithms
 - Data preprocessing and dimensional reduction
 - Classification algorithms
 - Clustering algorithms
 - Itemset mining
 -

26

Spark Components

- GraphX
 - A graph processing library
 - Provides many algorithms for manipulating graphs
 - Subgraph searching
 - PageRank
 -

27

Spark Schedulers

- Spark can exploit many schedulers to execute its applications
 - Hadoop YARN
 - Standard scheduler of Hadoop
 - Mesos cluster
 - Another popular scheduler
 - Standalone Spark Scheduler
 - A simple cluster scheduler included in Spark

28