Curso de extensão em Data Science

GERÊNCIA DE INFRAESTRUTURA PARA BIG DATA

Prof. Tiago Ferreto – tiago.ferreto@pucrs.br

ESCOLA POLITÉCNICA

Spark Introduction

- Started in 2009 – Berkeley RAD Lab (University of California)
- Created as an alternative to MapReduce on Hadoop
  - MapReduce is unsuited for interactive queries or real-time, low latency applications
    - MR persists intermediate data to disk between Map and Reduce processing phases
  - Spark benefits: better performance, extensibility, better support for different scenarios (SQL access, streaming data processing, graph and NoSQL processing, machine learning, etc)
- ASF project - http://spark.apache.org/
  - Several contributors: Facebook, Yahoo!, Intel, Netflix, Databricks, etc
- Written in Scala (built on top of the JVM and Java runtime)
  - Cross-platform (supports Windows and Linux)
- Enables developers to create complex, multi-stage data processing routines
  - Provides a high-level API and fault-tolerant framework
- Spark implements a distributed, fault tolerant, in-memory structure called RDD (Resilient Distributed Dataset)
  - Maximizes use of memory across machines → improves performance by orders of magnitude

Typical Spark applications

- Extract-transform-load (ETL) operations
- Predictive analytics and machine learning
- Data access operations (such as SQL queries and visualizations)
- Text mining and text processing
- Real-time event processing
- Graph applications
- Pattern recognition
- Recommendation engines

Programming interfaces

- Provides native support for
  - Scala
  - Python
  - Java
  - SQL
  - R
  - And others (Clojure, Julia, etc)

Interaction with Spark

- Spark provides interactive shells in Python (**PySpark**), Scala, R and SQL

Non-interactive use

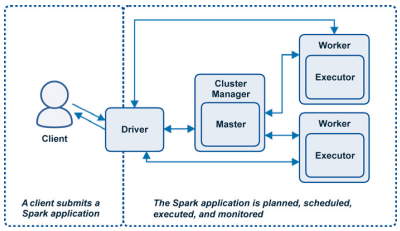
- Spark provides the spark-submit command to execute non-interactive applications

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn-cluster \
--num-executors 4 \
--driver-memory 10g \
--executor-memory 10g \
--executor-cores 1 \
lib/spark-examples*.jar 10
```


Input/Output types

- Supports several source/target systems
  - Local or network filesystems
  - Object storage such as Amazon S3 or Ceph
  - Relational database systems
  - NoSQL stores, including Apache Cassandra, HBase, and others
  - Messaging systems such as Kafka

Spark Architecture



Spark Driver

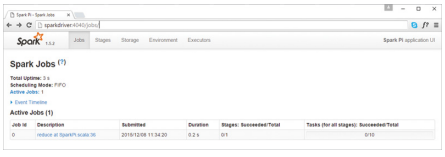
- Represented by the process used by clients to submit applications
- Manages the lifecycle of Spark applications
  - Creates a SparkContext
    - Application instance representing the connection to the Spark master and the Spark executors
    - It is initiated at the beginning of a Spark application and used till program termination

Spark Driver

- Plans and coordinates the execution of the Spark program
  - Planning – creates a DAG (Directed Acyclic Graph) based on requested transformations (data manipulation operations) and actions (output requests)
    - DAG consists of tasks and stages
      - Tasks are the smallest unit of schedulable work
      - Stages are set of tasks that can be run together
      - Stages are dependent upon on another
  - Scheduling
    - Coordinates the running of stages and tasks
    - Main activities
      - Keep track of available resources to execute tasks
      - Schedule tasks close to the data
      - Coordinates the location and movement of data between processing stages

Spark Driver

- Responsible for returning results from an application
  - Can be return code or data
- Serves the Application UI (port 4040)



Spark Executors

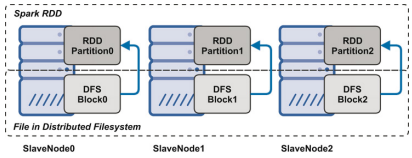
- Consist of host processes on which tasks from a Spark DAG are executed
- Executors reserve CPU and memory resources on slave nodes or workers in a Spark cluster
- They are dedicated to a specific Spark application and terminated when the application completes.
  - A Spark executor can run hundreds or thousands of tasks within a Spark program.
- Spark executors are hosted in JVMs with a configurable heap memory
- Executors store output data from tasks in memory or on disk
  - Workers and executors are only aware of the tasks allocated to them, whereas the driver is responsible for understanding the complete set of tasks and their respective dependencies that comprise an application

RDD – Resilient Distributed Dataset

- Datasets within a Spark application, including the initial dataset(s) loaded, any intermediate datasets, and the final resultant dataset(s)
- Use various types of elements: integers, floats, strings, lists, hashes, nested objects, serializazed Scala and Java objects, etc
- Normally stored in memory (but can also be persisted to disk)
- Characteristics
  - Resilient – can be reconstructed if a node is lost → Spark knows how to generate every RDD
  - Distributed – data in RDDs is divided into one or more partitions and are distributed as in-memory collections of objects across worker nodes in the cluster
  - Dataset – RDDs consist of records, which are identifiable data collections within a dataset
  - Shared nothing – RDDs are partitioned such that each partition contains a unique set of records and can be operated on independently
  - Immutability – RDDs cannot be updated after being instantiated and populated with data → new RDDs are created through transformations

Data Locality with RDDs

- Spark reads data into an RDD from the nodes that are close to it
- Since Spark usually accesses distributed partitioned data (from HDFS), it creates partitions to hold underlying blocks from the distributed filesystem
- RDDs can also be loaded from other data sources (from relational databases to simple Python/Scala objects)

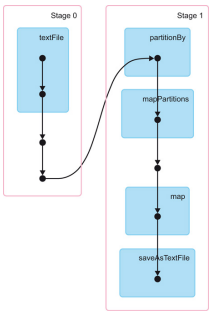


RDD Persistence and Re-use

- Normally RDDs are transient objects that exist only while they are required
- Performance impact when an RDD is required for more than one action
- Spark provides API methods to persist, cache and checkpoint RDDs

RDD Lineage

- Spark keeps track of each RDD's lineage → sequence of operations that resulted in the RDD
- Every RDD operation recomputes the entire lineage by default unless RDD persistence is requested
- Spark creates a DAG consisting of dependencies between RDDs
  - RDDs are processed in stages → sets of transformations
- Spark Application UI allows visualizing DAGs in a Spark application



Fault Tolerance with RDDs

- In case of any failure, any RDD can be reconstructed using Spark records of the RDD lineage
- Since RDDs are distributed, they can tolerate and recover from the failure of any single node

Transformations and Actions in Spark

- **Transformations** – operations performed against RDDs resulting in new RDDs
  - Common transformations: map and filter functions
  - Example

```
originalrdd = sc.parallelize([0, 1, 2, 3, 4, 5, 6, 7, 8])
newrdd = originalrdd.filter(lambda x: x % 2)
```

- **Actions** – operations that produce output from an RDD or save the content of an RDD to a filesystem (local, HDFS, S3, or other)
  - Example

```
newrdd.collect() # will return [1, 3, 5, 7]
```


Lazy Evaluation

- Processing is postponed until an action is called (i.e., when an output is required)
- Several transformations to RDDs can be performed before any processing starts
- Each statement is parsed for syntax and object references, until an action is requested (e.g., count() or saveAsTextFile())
- A DAG is created along with logical and physical execution plans, which are orchestrated and managed across executors by the driver
- This approach enables reducing processing stages and minimizing the amount of data transferred between Spark executors (shuffle)

Spark Extensions

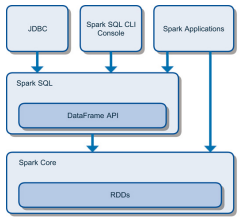
- SparkSQL – provides SQL-like abstraction to Spark
- Spark Streaming – enables processing streams of data
- SparkR – scalable runtime engine for R
- MLlib – machine learning library integrated into Spark
- GraphX – graph processing with Spark

SparkSQL

- Provides SQL-like abstraction to the Spark core API
- DataFrame – extension to RDD implementing an in-memory columnar storage scheme optimized for SQL access
- Example

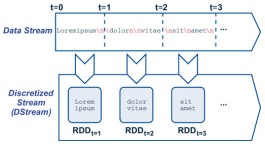
```
sql_cmd = """SELECT name, lat, long
FROM stations
WHERE landmark = 'San Jose'"""
df = sqlContext.sql(sql_cmd)
df.show()
```

```
+--------------------+----------+-----------+
|                name|       lat|       long|
+--------------------+----------+-----------+
|    Adobe on Almaden|37.331415|  -121.8932|
|    San Pedro Square|37.336721|-121.894074|
|Paseo de San Antonio|37.333798|-121.886943|
| San Salvador at 1st|37.330165|-121.885831|
|           Japantown|37.348742|-121.894715|
|                 ...|       ...|        ...|
+--------------------+----------+-----------+
```



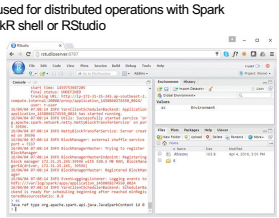
Spark Streaming

- Introduces objects and functions designed to process streams of data
- DStream (discretized stream) – RDD abstraction comprised of streams of data batched into RDDs based on time intervals
- Spark transformations can be applied to Dstreams applying functions to each underlying RDD in the Dstream
- Supports stream processing operations (state and sliding window operations)
  - Example: operations on data "windows" (hour, day, etc)



SparkR

- Spark engine for R
  - R → programming language and environment for statistical computing, visual analytics, and predictive modeling
- Provides access to Spark and distributed data frame operations from an R environment using the R language
- R data frames can be used for distributed operations with Spark
- Available through SparkR shell or RStudio



MLlib

- Spark integrated library for machine learning
- API built upon the RDD model
- Includes common machine learning algorithms and utilities for data preparation, feature extraction, model training, and testing

```
from pyspark.mllib.tree import DecisionTree
model = DecisionTree.trainClassifier(data=data,
          numClasses=2,
          categoricalFeaturesInfo={0: 3})
...
print(model.toDebugString())
If (feature 2 <= 80.0)
        If (feature 1 <= 65.0)
        If (feature 1 <= 64.0)
         Predict: 1.0
        Else (feature 1 > 64.0)
         Predict: 0.0
        Else (feature 1 > 65.0)
         Predict: 1.0
      Else (feature 2 > 80.0)
        If (feature 0 in {0.0})
         Predict: 0.0
        Else (feature 0 not in {0.0})
        If (feature 1 <= 71.0)
          If (feature 1 <= 70.0)
           Predict: 1.0
          Else (feature 1 > 70.0)
           Predict: 0.0
        Else (feature 1 > 71.0)
          Predict: 1.0
```


GraphX

- Graph processing using Spark
  - Enables modeling relationships between different entities or discrete data items
- Provides a full set of graph data abstractions, transformations, and algorithms for graph processing
- GraphFrames – RDD abstraction for graph processing

- Example using PageRank algorithm (used in web search engines)
  - `graph = GraphRDD(vertices, edges)`
    `ranks = graph.pageRank(0.0001).vertices`

SPARK – HANDS-ON