ISIT312 Big Data Management

Spark Data Model

Dr Guoxin Su and Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Introduction to Spark

Outline

Resilient Distributed Data Sets (RDDs)

Resilient Distributed Datasets (RDDs) is the lowest level (and the oldest) data abstraction available to the users

An RDD represents an immutable, partitioned collection of elements that can be operated on in parallel

Every row in an RDD is a Java object

RDD does not need to have any schema defined in advance

It makes RDD very flexible for various applications but in the same moment makes the manipulations on data more complicated

Users must implement their own functions to perform simple tasks like for example aggregation functions: average, count, maximum, etc

RDDs provide more control on how data is distributed over a cluster and how it is operated

Resilient Distributed Data Sets (RDDs)

RDD is characterized by the following properties

- A list of partitions
- A function for computing each split
- A list of dependencies on other RDDs
- Optionally, a Partitioner for `key-value` RDDs
- Optionally, a list of preferred locations to compute each split

Specification of custom partitions may provide significant performance improvements when using `key-value` RDDs

The properties of RDDs determine how Spark schedules and processes the applications

Different RDDs may use different properties depending on the required outcomes

Resilient Distributed Data Sets (RDDs)

RDD can be created from a list

Creating a list of strings

```
val strings ="hello hello !".split(" ")
strings: Array[String] = Array(hello, hello, !)
```

Creating RDD

```
val rdset = spark.sparkContext.parallelize(strings);
rdset: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[1] at parallelize
                                                                   at :26
```

Listing RDD

Listing RDD

```
rdset.collect()
res1: Array[String] = Array(hello, hello, !)
```

Resilient Distributed Data Sets (RDDs)

Creating RDD from a file by reading line-by-line

Creating RDD

```
val lines =  spark.sparkContext.textFile("sales.txt")
lines: org.apache.spark.rdd.RDD[String] = sales.txt MapPartitionsRDD[1] at textFile at :24
```

Listing RDD

```
lines.collect()
res0: Array[String] = Array(bolt 45, bolt 5, drill 1, drill 1, screw 1, screw 2, screw 3)
```

Creating key-value pairs

```
val pairs = lines.map(s => (s, 1))
pairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[2] at map at :26)
```

Counting

```
val counts = pairs.reduceByKey((a, b) => a + b)
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[3] at reduceByKey at :28)
```

Listing RDD

```
counts.collect()
res1: Array[(String, Int)] = Array((bolt 5,1), (drill 1,2), (bolt 45,1), (screw 2,1), (screw 3,1),
(screw 1,1))
```

Introduction to Spark

Outline

DataFrames

A DataFrame is a table of data with rows and columns

A list of columns in DataFrame together with the types of columns is called as a schema

A DataFrame can span over many systems in a cluster

The concepts similar to DataFrame has been used in Python and R

A DataFrame is the simplest data model available in Spark

A DataFrame consist of zero or more partitions

- To parallelize data processing all data are divided into chunks called as partitions
- A partition is a collection of rows located on a single system in a cluster
- When operating DataFrame Spark simultaneously processes all relevant partitions
- Shuffle operation is performed to share data among the systems in a cluster

DataFrames

A DataFrame can be created in the following way

Creating a DataFrame

```
val dataFrame = spark.read.json("/bigdata/people.json")
```

The contents of a DataFrame can be listed in the following way

Listing results

```
dataFrame.show()

+----+-------+
| age| name  |
+----+-------+
|null|Michael|
| 30 | Andy  |
| 19 | Justin|
+----+-------+
```

DataFrames

A schema of a DataFrame can be listed in the following way

Listing a schema

```
dataFrame.printSchema()
```

```
root
|-- age: long (nullable = true)
|-- name: string (nullable = true)
```

Introduction to Spark

Outline

SQL Tables/Views

SQL can be used to operate on DataFrames

It is possible to register any DataFrame as a table or view (a temporary table) and apply SQL to it

There is no performance overhead when registering a DataFrame as SQL Table and when processing it

A DataFrame `dataFrame` can be registered as SQL temporary view `people` in the following way

DataFrame as SQL Table

```
dataFrame.createOrReplaceTempView("people")
val sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

Results

```
+----+-------+
| age| name  |
+----+-------+
|null|Michael|
|30  |Andy   |
|19  |Justin |
+----+-------+
```

SQL Tables/Views

SQL Tables are logically equivalent to DataFrames

A difference between SQL Tables and DataFrames is such that DataFrames are defined within the scope of a programming language while SQL Tables are defined within a database

When created, SQL table belongs to a default database

SQL table can be created in the following way

CREATE TABLE

```
CREATE TABLE flights(
 DEST_COUNTRY_NAME STRING,
 ORIGIN_COUNTRY_NAME STRING, count LONG)
   USING JSON OPTIONS ( path '/mnt/defg/chapter-1-data/json/2015-summary.json')
```

SQL view can be created in the following way

CREATE VIEW

```
CREATE VIEW just_usa_view AS
 SELECT *
 FROM flights
 WHERE dest_country_name = 'United States'
```

SQL Tables/Views

Global SQL view can be created in the following way

CREATE VIEW

```
CREATE GLOBAL VIEW just_usa_global AS
 SELECT *
 FROM flights
 WHERE dest_country_name = 'United States'
```

Temporary SQL view can be created in the following way

CREATE VIEW

```
CREATE TEMP VIEW just_usa_global AS
 SELECT *
 FROM flights
 WHERE dest_country_name = 'United States'
```

Global and Temporary SQL view can be created in the following way

CREATE VIEW

```
CREATE GLOBAL TEMP VIEW just_usa_global AS
 SELECT *
 FROM flights
 WHERE dest_country_name = 'United States'
```

SQL Tables/Views

Spark SQL view includes three core complex types: sets, lists, and structs

Structs allow to create nested data

CREATE VIEW

```
CREATE VIEW nested_data AS
 SELECT struct(DEST_COUNTRY_NAME, ORIGIN_COUNTRY_NAME) as country
 FROM flights
```

The functions `collect_set` and `collect_list functions` create sets and lists

CREATE VIEW

```
SELECT DEST_COUNTRY_NAME as new_name,
       collect_list(count) as flight_counts,
       collect_set(ORIGIN_COUNTRY_NAME) as origin_set
FROM flights
GROUP BY DEST_COUNTRY_NAME
```

SQL Tables/Views

Managed versus unmanaged tables

- Managed table is a table that stored data and metadata, it is equivalent to an internal table in Hive
- Unmanaged table is a table that stores only data, it is equivalent to an external table in Hive

Creating unmanaged (external) table in Spark

CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE hive_flights(
 DEST_COUNTRY_NAME STRING,
 ORIGIN_COUNTRY_NAME STRING,
 count LONG)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  LOCATION '/mnt/defg/flight-data-hive/'
```

Introduction to Spark

Outline

Datasets

A Dataset is a distributed collection of data

A DataFrame, is a Dataset of type `Row`

Datasets consists of the objects included in the rows; one object per row

Datasets are a strictly JVM language feature that only work with Scala and Java

A Dataset can be constructed from JVM and then manipulated using functional transformations

In Scala a `case class` object defines a schema of an object

Datasets use a specialized Encoder to serialize the objects for processing or transmitting over the network

Dataset supports all operations of DataFrame

Datasets

A Dataset can be defined, created, and used in the following way

Creating Dataset Schema

```
case class Person(name: String, age: Long)
```

Creating Dataset

```
val caseClassDS = Seq(Person("Andy", 32)).toDS()
```

Listing Dataset

```
caseClassDS.show()
```

Results

```
+----+---+
|name|age|
+----+---+
|Andy| 32|
+----+---+
```

Using a Dataset

```
caseClassDS.select($"name").show()
```

Results

```
+-----+
| name|
+-----+
|James|
+-----+
```

References

A Gentle Introduction to Spark, Databricks, (Available in `READINGS` folder)

RDD Programming Guide

Spark SQL, DataFrames and Datasets Guide

Karau H., Fast data processing with Spark Packt Publishing, 2013 (Available from UOW Library)

Srinivasa, K.G., Guide to High Performance Distributed Computing: Case Studies with Hadoop, Scalding and Spark Springer, 2015 (Available from UOW Library)

Chambers B., Zaharia M.,Spark: The Definitive Guide, O'Reilly 2017