Part A. Big Data Technology
3. Distributed Computing Models for Scalable Batch Computing

Section 2: In-Memory Cluster Computing

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs
- Quiz 2
  - We will discuss Quiz 2 at the end of this class

Topics of Today’s Class
- 3. Distributed Computing Models for Scalable Batch Computing
  - Data Frame
  - Spark SQL
  - Datasets

In-Memory Cluster Computing: Apache Spark
SQL, DataFrames and Datasets

What is the Spark SQL?
- Spark module for structured data processing
  - Interface is provided by Spark
  - SQL and the Dataset API
- Spark SQL is to execute SQL queries
  - Available with the command-line or over JDBC/ODBC

What is the Datasets?
- Dataset is a distributed collection of data
- New interface added in Spark 1.6 provides
  - Benefits of RDDs (Storing typing, ability to use lambda functions)
  - Benefits of Spark SQL’s optimized execution engine
- Available in Scala and Java
- Python does not support Datasets APIs!

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 1
What is the DataFrame?

- DataFrame is a Dataset organized into named columns
- Like a table in a relational database or a data frame in R/Python
- Strengthened optimization scheme
- Available with Scala, Java, Python, and R

In-Memory Cluster Computing: Apache Spark
SQL, DataFrames and Datasets
Getting Started

Create a SparkSession: Starting Point

SparkSession

The entry point into all functionality in Spark

```scala
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

// For implicit conversions like converting RDDs to DataFrames
import spark.implicits._
```

Find full example code at the Spark repo
examples/streaming/apache/spark/examples/sparkSQLExample.scala

Creating DataFrames

- With a SparkSession, applications can create DataFrames from
  - Existing RDD
  - Hive table
  - Spark data sources

```scala
val df = spark.read.json("examples/src/main/resources/people.json")
// Displays the content of the DataFrame to stdout
df.show()
// +----+
// | age|
// +----+
// |null|
// | 30|
// | 19|
// +----+
```

Untyped Dataset Operation

- A.K.A. DataFrame Operations
- DataFrames are just Dataset of Rows in Scala and Java API
- Untyped transformations
  - "typed operations"?
  - Strongly typed Scala/Java Datasets

```scala
val spark = sparkSession
  .localBuilder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

// Print the schema as a tree format
df.printSchema()
// root
// |-- age: long (nullable = true)
// |-- name: string (nullable = true)
```

Untyped Dataset Operation

```scala
df.select only the `name` column
df.select("name").show()
// +----+
// | name|
// +----+
// | Michael|
// | Andy|
// | John|
// +----+
```
Untyped Dataset Operation

```scala
// Select everybody, but increment the age by 1
df.select("name", "$age + 1").show();
```

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>null</td>
</tr>
<tr>
<td>Andy</td>
<td>31</td>
</tr>
<tr>
<td>Justin</td>
<td>20</td>
</tr>
</tbody>
</table>

Untyped Dataset Operation

```scala
// Select people older than 21
df.filter("age > 21").show();
```

<table>
<thead>
<tr>
<th>age</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Andy</td>
</tr>
<tr>
<td>19</td>
<td>Justin</td>
</tr>
</tbody>
</table>

Untyped Dataset Operation

```scala
// Count people by age
df.groupBy("age").count().show();
```

<table>
<thead>
<tr>
<th>age</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
</tr>
</tbody>
</table>

Running SQL Queries

```scala
val sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show();
```

<table>
<thead>
<tr>
<th>age</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Michael</td>
</tr>
<tr>
<td>30</td>
<td>Andy</td>
</tr>
<tr>
<td>19</td>
<td>Justin</td>
</tr>
</tbody>
</table>

Global Temporary View

- **Temporary views** in Spark SQL
  - Session-scoped
  - Will disappear if the session that creates it terminates

- **Global temporary view**
  - Shared among all sessions and keep alive until the Spark application terminates
  - A system preserved database

Global Temporary View

```scala
// Global temporary view is tied to a system preserved database
spark.sql("SELECT * FROM global_temp.people").show();
```

<table>
<thead>
<tr>
<th>age</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Michael</td>
</tr>
<tr>
<td>30</td>
<td>Andy</td>
</tr>
<tr>
<td>19</td>
<td>Justin</td>
</tr>
</tbody>
</table>

http://www.cs.colostate.edu/~cs535
Global Temporary View

```scala
// Global temporary view is cross-session
spark.newSession().sql("SELECT * FROM global_temp.people").show()
```

Creating Datasets

```scala
// Datasets are similar to RDDs
// Serializes object with Encoder (not standard java/Kryo serialization)
// Datasets are using non-standard serialization library (Spark encoder)
// Many of Spark Dataset operations can be performed without deserializing object

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

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()

// Case classes can also be nested or contain complex types such as Seqs or Arrays
val casesDS = Seq(Person().name("Michael").age(19)).toDS()

// Many datasets are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()

// Creating datasets from case classes

// In-Memory Cluster Computing: Apache Spark
```

Interoperating with RDDs

- Converting RDDs into Datasets
  - Case 1: Using reflections to infer the schema of an RDD
  - Case 2: Using a programmatic interface to construct a schema and then apply it to an existing RDD

Interoperating with RDDs: 1. Using Reflection

- Automatic converting of an RDD (containing case classes) to a DataFrame
  - The case class defines the schema of the table
  - E.g. the names of the arguments to the case class are read using reflection
  - become the names of the columns
  - Case classes can also be nested or contain complex types such as Seqs or Arrays
  - RDD will be implicitly converted to a DataFrame and then be registered as a table

http://www.cs.colostate.edu/~cs535
Interoperating with RDDs: 1. Using Reflection

// For implicit conversions from RDDs to DataFrames
import spark.implicits._

// Generate the schema based on the string of schema
val schemaString = "name age"

// The schema is encoded in a string
val schema = StructType.fromJson(schemaString)

// Apply the schema to the RDD
results.foreach { r => val row = r
  val name = row.getAs[String]("name")
  val age = row.getAs[Int]("age")
  println(s"Name: $name, Age: $age")
}

// Create an RDD
val peopleRDD = spark.sparkContext
  .textFile("/main/resources/people.txt")
  .map(_.split(\"\"))
  .map(attributes => Person(attributes(0), attributes(1), trim(attributes)))
  .toDF(schema)

// Register the DataFrame as a temporary view
teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 18")

// SQL statements can be run by using the sql method provided by Spark
val results = teenagersDF.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 18")

// The results of SQL queries are DataFrames and support all the DataFrame operations
results.foreach { r => val row = r
  val name = row.getAs[String]("name")
  val age = row.getAs[Int]("age")
  println(s"Name: $name, Age: $age")
}

// Create a temporary view using the DataFrame
val resultsDF = teenagersDF.createTempView("teenagers")

// SQL can be run over a temporary view created using Dataset
val results2 = spark.sql("SELECT name FROM teenagers")

// The results of SQL queries are DataFrames and support all the DataFrame operations
results2.foreach { r => val row = r
  val name = row.getAs[String]("name")
  println(s"Name: $name")
}

import org.apache.spark.sql.expressions

// Class to demonstrate implicit conversions
class Person(name: String, age: Int, trim(name: String))

// Implicit conversions from RDDs to DataFrames
import spark.implicits._

// Apply the schema to the RDD
results.foreach { r => val row = r
  val name = row.getAs[String]("name")
  val age = row.getAs[Int]("age")
  println(s"Name: $name, Age: $age")
}

// Create an RDD
val peopleRDD = spark.sparkContext
  .textFile("/main/resources/people.txt")
  .map(_.split(\"\"))
  .map(attributes => Person(attributes(0), attributes(1), trim(attributes)))
  .toDF(schema)

// Register the DataFrame as a temporary view
teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 18")

// SQL statements can be run by using the sql method provided by Spark
val results = teenagersDF.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 18")

// The results of SQL queries are DataFrames and support all the DataFrame operations
results.foreach { r => val row = r
  val name = row.getAs[String]("name")
  val age = row.getAs[Int]("age")
  println(s"Name: $name, Age: $age")
}

// Create a temporary view using the DataFrame
val resultsDF = teenagersDF.createTempView("teenagers")

// SQL can be run over a temporary view created using Dataset
val results2 = spark.sql("SELECT name FROM teenagers")

// The results of SQL queries are DataFrames and support all the DataFrame operations
results2.foreach { r => val row = r
  val name = row.getAs[String]("name")
  println(s"Name: $name")
}

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 5
Aggregations

- Case 1. Untyped User-Defined Aggregate Functions

- Case 2. Type-Safe User-Defined Aggregate Functions

http://www.cs.colostate.edu/~cs535
Aggregations 2. Type-Safe User-Defined Aggregate functions

```scala
// Transform the output of the reduction
val finish(reduction: Average) = reduction.sum.toDouble / reduction.count

// Specify the encoder for the intermediate value type
val encoderIntermediate: Encoder[Average] = Encoders.product

// Specify the encoder for the final output value type
val encoderOutput: Encoder[Double] = Encoders.scalaDouble
```

Datasets and Type Safety

- Datasets are composed of typed objects
  - Transformation syntax errors (e.g., typo in the method name) and analysis errors (e.g., incorrect input variable type) can be caught at compile time
- DataFrames are composed of untyped Row objects
  - Only syntax errors can be caught at compile time
- Spark SQL is composed of a string
  - Syntax errors and analysis errors are only caught at runtime

In-Memory Cluster Computing: Apache Spark SQL, DataFrames and Datasets

Execution Model

Spark Execution Model

- Three phases of the Spark Execution Model
  - Creating the logical plan
  - Translating that into a physical plan
  - Executing the tasks on a cluster

Logical Plan

- Shows which steps will be executed when an action gets applied
  - When you apply a transformation to a dataset, what will happen?
    - Creating a lineage (directed acyclic graph) for how Spark will execute these transformations

http://www.cs.colostate.edu/~cs535
Physical Plan
- Actions trigger the translation of the logical DAG into a physical execution plan
- Spark Catalyst Query Optimizer
  - Creates the physical execution plan for DataFrames
  - Identifies resources (e.g., memory partitions and compute tasks)

Executing the Tasks on a Cluster
- The scheduler splits the graph into stages
  - Based on the transformations
  - The narrow transformations (transformations without data movement) will be grouped (pipe-lined) together into a single stage
- Each stage is comprised of tasks
  - Based on the partitions of data
  - Same computation in parallel

Executing the Tasks on a Cluster
- The scheduler submits the stage task set to the task scheduler
  - Launches tasks via a cluster manager (Yarn, Mesos)

Summary of the components of execution
- Task
  - A unit of execution that runs on a single machine
- Stage
  - A group of tasks, based on partitions of the input data
- Job
  - Has one or more stages
- Pipelining
  - Collapsing of datasets into a single stage
  - When dataset transformations can be computed without data movement
- DAG
  - Logical graph of dataset operations
How to interpret the Web UI

- Job Id 2 is the job that was triggered by the collect action on df3

- Clicking the link in the Description column on the Jobs page
  - Job Details page
  - Number of tasks correspond to the partitions
    - After reading file: 2 partitions
    - After a shuffle: 200 partitions

- Under the "Stages" tab

- Detailed information about your RDDs

Quiz 2
- Multiple choices and/or T/F
- What is Spark? Why Spark?
- RDDs
  - Transformations. (e.g. What is the difference between map and filter?, 2 RDD transformations)
  - Actions
  - Persistence
  - Pair RDDs
  - Spark Cluster
  - Lazy Evaluation
  - Spark Lineage
  - RDD Dependency
  - Generating job stages
  - Closure
  - Dataset/Dataframes/SparkSQL
  - Spark Execution Model

Questions?