Week4_Intro_to_DataFrames_SparkSQL

RDDs in Parallel Programming & Spark

RDD

- Ares Spark's primary data abstraction
- Are partitioned across the nodes of the cluster

RDD Transformations

Transformations:

- Create a new RDD from existing one
- Are "lazy" because the results are only computed when evaluated by actions

The map transformation passes each element of a dataset through a function and returns a new RDD `map()`

RDD Actions

Actions return a value to driver program after running a computation
`reduce()` - An action that aggregates all RDD elements

The Directed Acyclic Graph (DAG)

- A graphical data structure with edges and vertices
- Every new edge is obtained from an older vertex
- In apache spark DAG, the vertices represents RDDs and the edges represent operations such as transformations or actions
- If a node goes down, Spark replicates the dag and restores the node.

Transformations & Actions

1. Spark creates the DAG when creating an RDD
2. Spark enables the DAG Scheduler to perform a transformation and updates the dag
3. The DAG now points to the new RDD
4. The pointer that transforms RDD is returned to the Spark driver program
5. If there is an action, the driver program that calls the action evaluates the DAG only after Spark completes the action

Transformation examples

Transformation	Description
map (*func*)	Returns a new distributed dataset formed by passing each element of the source through a function *func*
filter (*func*)	Returns a new dataset formed by selecting those
distinct ([numTasks]))	Returns a new dataset that contains the distinct elements of the source dataset
flatmap (*func*)	Similar to map (*func*) Can map each input item to zero or more output items *Func* should return a Seq rather than a single item

Action Examples

Action	Description
reduce(*func*) aggregates dataset elements using the function *func*	*func* takes two arguments and returns one • Is commutative • Is associative • Can be computed correctly in parallel
take(*n*)	Returns an array with the first *n* element
collect()	Returns all the elements as an array WARNING: Make sure that ? will fit in driver program
takeOrdered (*n*,key=*func*)	Returns *n* elements ordered in ascending order or as specified by the optional key function

Data-frames and Datasets

Datasets

A dataset is a distributed collection of data that:

- Consists of a collection of strongly typed JVM objects
- Provides the combined benefits of both RDDs and Spark SQL

Datasets Features

- Are immutable, meaning that data cannot be deleted or lost
- Feature an encoder that converts JVM objects to a tabular representation
- Extend DataFrame type-safe and object-oriented API capabilities
- Work with both Scala and Java APIs

Datasets in Spark - Benefits

- Provide compile-time type safety
- Compute faster than RDDs
- Offer the benefits of Spark SQL and DataFrames
- Optimize queries using Catalyst and Tungsten
- Enable improved memory usage and caching
- Use dataset API functions for aggregate operations including sum, average, join and group by

Creating a dataset

- Apply the `toDS()` function to create a dataset from a sequence

```
// create a dataset from a sequence of primitive datatype - /  scala

val ds = Seq("Alpha","Beta","Gamma").toDS()
```

- Create a dataset from a text file

```
val ds = spark.read.text("/text_folder/file.txt").as[String]
```

- Create a dataset using a JSON file

```
case class Customer(name: String, id:Int, phone:Double)
val ds_cust = spark.read.json("/customer.json").as[Customer]
```

Datasets & DataFrames Compared

Datasets are	DataFrames are
Strongly-typed	Not typesafe
Use unified Java and Scala APIs	Use APIs in Java, Scala, Python and R
Built on top of dataframes and the latest data abstraction added to Spark	Built on top of RDDs and added in the earlier spark versions

Catalyst & Tungsten

Spark SQL Optimization goals

Reduce query time and memory consumption, saving organizations time and money,

Catalyst defined

- Is the Spark SQL built-in sule-based query optimizer
- Based on functional programming constructs in Scala

- Supports the addition of new optimization techniques and features
- Enables developers to add data source-specific rules and support new data types

SQL Optimization explained
Rule-based optimization -> defines how to run the query

Examples:

- Is the table indexed?
- Does the query contain only the required columns?

Cost-based optimization -> equals time + memory a query consumes

Example
What are the best paths for multiple datasets to use when querying data?

Catalyst query optimization

- Uses a tree data structure and a set of rules
- Four major phases of query execution
  - Analysis
  - Logistical Optimization
  - Physical Planning
  - Code Generation

Tungsten defined
Spark's cost-based optimizer that maximizes CPU and memory performance

Tungsten Features

- Manages memory explicitly and does not rely on the JVM object model or garbage collection
- Enables cache-friendly computation of algorithms and data structures using both STRIDE-based memory access
- Supports on-demand JVM byte code generation
- Does not generate virtual function dispatches
- Places intermediate data in CPU registers
- Enables Loop unrolling

ETL with DataFrames

Basic DataFrame operations

- **Read** the data

- **Analyze** the data
- **Transform** the data
- **Load** data into a database
- **Write** data back to disk

Process commonly described as ETL

Read the data

- Create a DataFrame
- Create a DataFrame from an existing DataFrame

```
import pandas as pd
mtcars = pd.read_csv('mtcars.csv')
sdf = spark.createDataFrame(mtcars)
```

Analyze the data using printschema

View the schema

`sdf.printSchema()`

Apply the show function

`sdf.show(5)`

Apply the select() function to view a specific column

`sdf.select('mpg').show(5)`

Transform the data - guidelines

- Keep only the relevant data
- Apply filters, joins, sources and tables, column operations, grouping and aggregations and other functions
- Apply domain-specific data augmentation processes

Transform the data using a filter

`sdf.filter(sdf['mpg'] < 18).show()`

```
car_counts = sdf.groupby(['cyl']).agg({"wt":"count"}).sort("count(wt)",
ascending=False).show(5)
```

Loading or Exporting the data

Final step of the ETL pipeline

- Export to another database
- Export to disk as JSON files
- Save the data to a Postgres database
- Use an API to export data

Real world-usage

Creating a View in SparkSQL

- Creating a table view in SparkSQL is required to run SQL queries programmatically on a DataFrame
- A view is a temporary table to run SQL queries
  - A temporary view provides *local scope within the current Spark session*
  - A Global Temporary view *provides global scope within the spark application*

Creating a View in Spark SQL

```
#create a dataframe from file
df = spark.read.json("people.json")

#create a temp view
df.createTempView("people")

#run sql query
spark.sql("select * from people").show()
```

```
#creating a global view
df.createGlobalTempView("people")
# run sql query
spark.sql("SELECT * FROM global_temp.people").show()
```

Aggregating Data

Used to aggregate data over columns

- DataFrames contain inbuilt common aggregation functions - `count()`, `countDistinct()`, `avg()`, `max()`, `min()` and others
- Alternatively, aggregate using SQL queries and tableviews

```
import pandas as pd
mtcars = pd.read_csv("mtcars.csv")
sdf = spark.createDataFrame(mtcars)
sdf.select('mpg').show(5)
```

```
sdf.createTempView("cars")
sql("select cyl, COUNT(*) FROM cars GROUPBY cyl ORDER by 2 DESC")
```

Spark SQL Data Sources

- Parquet files

    - Supports reading/writing and preserving data schema
    - Spark SQL can also run queries without loading the file
- JSON datasets
    - Spark infers the schema and laods the dataset as a DataFrame
- Hive tables:
    - Spark supports reading and writing data stored in Apache Hive

Summary

RDDs are Spark's primary data abstraction partitioned across the nodes of the cluster. Transformations leave existing RDDs intact and create new RDDs based on the transformation function. With a variety of available options, apply functions to transformations perform operations. Next, actions return computed values to the driver program. Transformations undergo lazy evaluation, meaning they are only evaluated when the driver function calls an action.

A dataset is a distributed collection of data that provides the combined benefits of both RDDs and SparkSQL. Consisting of strongly typed JVM objects, datasets make use of DataFrame typesafe capabilities and extend object-oriented API capabilities. Datasets work with both Scala and Java APIs. DataFrames are not typesafe. You can use APIs in Java, Scala, Python. Datasets are Spark's latest data abstraction.

The primary goal of Spark SQL Optimization is to improve the run-time performance of a SQL query, by reducing the query's time and memory consumption, saving organizations time and money. Catalyst is the Spark SQL built-in rule-based query optimizer. Catalyst performs analysis, logical optimization, physical planning, and code generation. Tungsten is the Spark built-in cost-based optimizer for CPU and memory usage that enables cache-friendly computation of algorithms and data structures.

Basic DataFrame operations are reading, analysis, transformation, loading, and writing. You can use a Pandas DataFrame in Python to load a dataset and apply the print schema, select function, or show function for data analysis. For transform tasks, keep only relevant data and apply functions such as filters, joins, column operations, grouping and aggregations, and other functions.

Spark SQL consists of Spark modules for structured data processing that can run SQL queries on Spark DataFrames and are usable in Java, Scala, Python and R. Spark SQL supports both temporary views and global temporary views. Use a DataFrame function or an SQL Query + Table View for data aggregation. Spark SQL supports Parquet files, JSON datasets and Hive tables.