DEV 3600 – Developing Apache Spark Applications (Spark v2.1)
Slide Guide

Revision 2.2

For use with the following courses:

DEV 3600 – Developing Apache Spark Applications (Spark v2.1)
DEV 360 – Introduction to Apache Spark (Spark v2.1)
DEV 361 – Build and Monitor Apache Spark Applications (Spark v2.1)
DEV 362 – Advanced Apache Spark (Spark v2.1)

This Guide is protected under U.S. and international copyright laws, and is the exclusive property of MapR Technologies, Inc.

© 2018, MapR Technologies, Inc. All rights reserved. All other trademarks cited here are the property of their respective owners.

Welcome to DEV 360, Introduction to Apache Spark, Lesson 1. This lesson describes Apache Spark, lists the benefits of using Spark, and defines the Spark components.

Learning Goals
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-2

Learning Goals

1.1 Describe Features of Apache Spark

1.2 Define Spark Components

1.3 Explain Spark Data Pipeline Use Cases

© 2018 MapR Technologies

At the end of this lesson, you will be able to describe the features of Spark, such as how Spark fits in the big data ecosystem, and why Spark and Hadoop fit together. You will also be able to define the Spark components and describe some data pipeline use cases.

Learning Goals

1.1 Describe Features of Apache Spark

1.2 Define Spark Components

1.3 Explain Spark Data Pipeline Use Cases

© 2018 MapR Technologies

First, let's discuss some of the features of Apache Spark.

What is Apache Spark?

- Cluster computing platform on top of storage layer
- Extends MapReduce with support for more components
  - Streaming
  - Interactive analytics
- Runs in-memory

© 2018 MapR Technologies

Spark is a cluster computing platform on top of a storage layer. It extends MapReduce and YARN with support for more types of components such as streaming and interactive analytics. It offers the ability to run computations in-memory, but is also more efficient than Hadoop MapReduce for running on disk.

Why Apache Spark?

Fast	Ease of Development	Deployment Flexibility	Unified Stack	Multi-language support
• 10x faster on disk • 100x faster in memory				

© 2018 MapR Technologies

Spark provides reliable in-memory performance. Iterative algorithms are faster as data is not being written to disks between jobs. In-memory data sharing across directed acyclic graphs, or DAGs, makes it possible for different jobs to work with the same data quickly.

Spark processes data 10 times faster than Hadoop MapReduce on disk, and 100 times faster in-memory.

Why Apache Spark?

Fast	Ease of Development	Deployment Flexibility	Unified Stack	Multi-language support
• 10x faster on disk • 100x faster in memory	• Write programs quickly • More operators • Interactive Shell • Less code			

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-7

You can easily build complex algorithms for data processing very quickly in Spark.

Spark provides support for many more operators than MapReduce. Spark also provides the ability to write programs interactively using the Spark Interactive Shell, which is available for Scala and Python. You can compose non-trivial algorithms with little code.

Why Apache Spark?

Fast	Ease of Development	Deployment Flexibility	Unified Stack	Multi-language support
• 10x faster on disk • 100x faster in memory	• Write programs quickly • More operators • Interactive Shell • Less code	• Deployment – Mesos – YARN – Standalone • Storage – MapR-XD – HDFS – S3		

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-8

You can continue to use your existing big data investments. Spark is fully compatible with Hadoop. It can run in YARN, and access data from sources including HDFS, MapR-XD, HBase, MapR-DB, and Hive. In addition, Spark can also use the more general resource manager Mesos.

Why Apache Spark?

Fast	Ease of Development	Deployment Flexibility	Unified Stack	Multi-language support
• 10x faster on disk • 100x faster in memory	• Write programs quickly • More operators • Interactive Shell • Less code	• Deployment – Mesos – YARN – Standalone • Storage – MapR-XD – HDFS – S3	Builds applications combining different processing models – Batch – Streaming – Interactive Analytics	

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-9

Spark has an integrated framework for advanced analytics like graph processing, advanced queries, stream processing, and machine learning. Spark combines these libraries into a unified stack so you can use a single programming language through the entire workflow.

Why Apache Spark?

Fast	Ease of Development	Deployment Flexibility	Unified Stack	Multi-language support
• 10x faster on disk • 100x faster in memory	• Write programs quickly • More operators • Interactive Shell • Less code	• Deployment – Mesos – YARN – Standalone • Storage – MapR-XD – HDFS – S3	Builds applications combining different processing models – Batch – Streaming – Interactive Analytics	• Scala • Python • Java • SparkR

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-10

Developers have the choice of using Scala, Python, Java, and SparkR.

Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: **Spark**, MapReduce, Hive, Pig	Dashboard
	Stream Processing: **Spark Streaming,** MapR-ES, Storm	
Apps	Machine Learning: **Spark MLlib**, Mahout	Query/ Advanced Analytics
	Queries: **Spark SQL**, Drill, Hive	
Web Services	Graph Processing: **Spark GraphFrames**, Giraph, Graphlab	Enterprise Data Warehouse

© 2018 MapR Technologies

This table depicts where Spark fits in the big data application stack.

Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: **Spark**, MapReduce, Hive, Pig	Dashboard
	Stream Processing: **Spark Streaming**, MapR-ES, Storm	
Apps	Machine Learning: **Spark MLlib**, Mahout	Query/ Advanced Analytics
Web Services	Queries: **Spark SQL**, Drill, Hive	Enterprise Data Warehouse
	Graph Processing: **Spark GraphFrames**, Giraph, Graphlab	

© 2018 MapR Technologies

On the left we see different data sources. There are multiple ways of getting data in and using different industry standards, such as the NFS web service or existing big data tools.

Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: **Spark**, MapReduce, Hive, Pig	Dashboard
	Stream Processing: **Spark Streaming**, MapR-ES, Storm	
Apps	Machine Learning: **Spark MLlib**, Mahout	Query/ Advanced Analytics
Web Services	Queries: **Spark SQL**, Drill, Hive	Enterprise Data Warehouse
	Graph Processing: **Spark GraphFrames**, Giraph, Graphlab	

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-13

The stack in the middle represents various big data processing workflows and tools that are commonly used. You may have just one of these workflows in your application, or a combination of many. Any of these workflows could read/write to or from the storage layer. As you can see here, with Spark, you have a unified stack. You can use Spark for any of the workflows.

Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: **Spark**, MapReduce, Hive, Pig	Dashboard
	Stream Processing: **Spark Streaming**, MapR-ES, Storm	
Apps	Machine Learning: **Spark MLlib**, Mahout	Query/ Advanced Analytics
Web Services	Queries: **Spark SQL**, Drill, Hive	Enterprise Data Warehouse
	Graph Processing: **Spark GraphFrames**, Giraph, Graphlab	

© 2018 MapR Technologies

The output can then be used to create real-time dashboards and alerting systems for querying and advanced analytics, and loading into an enterprise data warehouse.

Hadoop

- Unlimited scale
- Multi-tenant
- Reliable
- Range of applications
- Files, semi-structured data, databases

© 2018 MapR Technologies

Hadoop
Hadoop has grown into a multi-tenant, reliable, enterprise-grade platform with a wide range of applications that can handle files, databases, and semi-structured data.

Spark

- Parallel in-memory processing across platform
- Multiple data sources, applications, users

© 2018 MapR Technologies

Spark
Spark is another enterprise-level computational platform that provides parallel, in-memory processing and can accommodate multiple data sources, applications, and users.

Spark on Hadoop

- Analytics over operational Hadoop applications
- Reliable unlimited scale

© 2018 MapR Technologies

Spark + Hadoop
The combination of Spark and Hadoop takes advantage of both platforms, providing reliable, scalable, and fast parallel, in-memory processing. Additionally, you can easily combine different kinds of workflows to provide analytics over your Hadoop and other operational applications.

Spark vs. MapReduce

- In-memory compute vs. read/write to disk
- Any combination of map/reduce operations vs. only one map/reduce per job

© 2018 MapR Technologies

Spark vs. MapReduce
The main difference between Spark and Hadoop MapReduce is that Spark keeps everything in-memory as much as possible, while MapReduce reads from and writes to disk for every job. You can also have only one map and one reduce program per MapReduce job, but in Spark, you can have a combination of any number of map and reduce operations.

Spark vs. MapReduce Example: Word Count

```
val dataDS = spark.read.text("/path/to/wordcount.txt").as[String]
val wordsDS = dataDS.flatMap(value => value.split("\\s+"))
val groupDS = wordsDS.groupByKey(_.toLowerCase)
val wordCountDS = groupDS.count()
wordCountDS.show()
```


© 2018 MapR Technologies

As an example, let's compare the code blocks that would be used for a simple word count operation. Here is the code that would give us the count of words in Scala using Spark.

Spark vs. MapReduce Example: Word Count

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

    public static class ReduceClass extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException {
            int sum = 0;
            for(IntWritable value : values) {
                sum += value.get();
            }
            con.write(word, new IntWritable(sum));
        }
    }

    public static void main(String [] args) throws Exception {
        Configuration conf=new Configuration();
        String[] files=new GenericOptionsParser(conf,args).getRemainingArgs();
        Path input=new Path(files[0]);
        Path output=new Path(files[1]);
        Job job=new Job(conf,"wordcount");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

R2.2 MAPR ACADEMY ESSENTIALS © 2018 MapR Technologies L1-20

In contrast, here is the code that would need to be used to run a Hadoop MapReduce program for the same word count operation.

Knowledge Check
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-21

Knowledge Check

What are the advantages of using Apache Spark?

A. Compatible with Hadoop

B. Ease of development

C. Fast

D. Multiple language support

E. Unified stack

F. All of the above

© 2018 MapR Technologies

Knowledge Check

What are the advantages of using Apache Spark?

A. Compatible with Hadoop

B. Ease of development

C. Fast

D. Multiple language support

E. Unified stack

F. All of the above

© 2018 MapR Technologies

Answer: F

Learning Goals
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-24

Learning Goals

1.1 Describe Features of Apache Spark

1.2 Define Spark Components

1.3 Describe Spark Data Pipeline Use Cases

© 2018 MapR Technologies

In this section, we take a look at Spark components and libraries.

A Spark cluster consists of two processes: a driver program and multiple worker nodes, each running an executor process.

The driver program runs on the driver machine, which is commonly an edge node, and the worker programs run on cluster nodes or in local threads.

Spark Components

© 2018 MapR Technologies

The first thing a program does is to create a SparkSession object. This tells Spark how and where to access a cluster.

Spark Components

© 2018 MapR Technologies

SparkSession connects to cluster manager. Cluster manager allocates resources across applications.

Once connected, Spark acquires executors in the worker nodes. An executor is a process that runs computations and stores data for your application.

Jar or Python files passed to the SparkSession are sent to the executors.

SparkSession will then send the tasks for the executor to run.

The worker nodes can access data storage sources to ingest and output data as needed. Next, we'll take a look at how this all comes together.

Spark Datasets

- Primary abstraction in Spark
- Collection of objects distributed across nodes in a cluster
- Data operations performed on datasets
- Once created, Datasets are immutable
- Can persist, or cache, in-memory or on disk
- Fault-tolerant

© 2018 MapR Technologies

Datasets are the primary abstraction in Spark. They are a collection of objects that are distributed across nodes in a cluster, and data operations are performed on them.

Once created, Datasets are immutable. You can also persist, or cache, Datasets in-memory or on disk. Spark Datasets are also fault-tolerant. If a given node or task fails, the Dataset can be reconstructed automatically on the remaining nodes and the job will complete.

Spark Dataset Operations

TRANSFORMATION

ACTION

© 2018 MapR Technologies

There are two types of data operations you can perform on a Dataset: transformations and actions.

Spark Dataset Transformations

© 2018 MapR Technologies

A transformation operates on an existing Dataset, and returns a new Dataset. A new Dataset is required because Datasets are immutable.

Spark Dataset Actions

© 2018 MapR Technologies

An action will return a value.

Spark Operations

© 2018 MapR Technologies

You can combine transformations and actions in any order, to process and analyze your data.

Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: **Spark**, MapReduce, Hive, Pig	Dashboard
	Stream Processing: **Spark Streaming**, MapR-ES, Storm	
Apps	Machine Learning: **Spark MLlib**, Mahout	Query/ Advanced Analytics
Web Services	Queries: **Spark SQL**, Drill, Hive	Enterprise Data Warehouse
	Graph Processing: **Spark GraphFrames**, Giraph, Graphlab	

© 2018 MapR Technologies

Now, let's discuss the Spark core and some of the libraries of transformations and actions that can be used.

We just took a look at this representation of a big data application stack, which list some available Spark libraries.

Spark Libraries

Component	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrames	• Graph processing • Graph parallel computations

Spark Core
- Task scheduling
- Memory management
- Fault recovery
- Interacting with storage systems

© 2018 MapR Technologies

The Spark core is a computational engine that is responsible for task scheduling, memory management, fault recovery, and interacting with storage systems. The Spark core contains the functionality of Spark. It also contains the APIs that are used to define Datasets and manipulate them.

The Spark libraries shown here support the Spark core and are all integrated into this stack.

Spark Libraries

Component	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrames	• Graph processing • Graph parallel computations
Spark Core • Task scheduling • Memory management • Fault recovery • Interacting with storage systems	

© 2018 MapR Technologies

Spark SQL can be used for working with structured data. You can query this data via SQL or HiveQL. Spark SQL supports many types of data sources such as structured Hive tables and complex JSON data.

Spark Libraries

Component	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrames	• Graph processing • Graph parallel computations
Spark Core • Task scheduling • Memory management • Fault recovery • Interacting with storage systems	

© 2018 MapR Technologies

Spark Streaming enables processing of live streams of data and doing real-time analytics.

Spark Libraries

Component	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrames	• Graph processing • Graph parallel computations
Spark Core • Task scheduling • Memory management • Fault recovery • Interacting with storage systems	

© 2018 MapR Technologies

MLlib is a machine learning library that provides multiple types of machine learning algorithms such as classification, regression, and clustering.

Spark Libraries

Component	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrames	• Graph processing • Graph parallel computations

Spark Core
- Task scheduling
- Memory management
- Fault recovery
- Interacting with storage systems

© 2018 MapR Technologies

GraphFrames is a library for manipulating graphs and performing graph-parallel computations.

Knowledge Check
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-42

Knowledge Check

Match the following:

- Responsible for task scheduling, memory management
- Tells Spark how and where to access a cluster
- Collection of objects distributed across many nodes in a cluster

A. SparkSession

B. Datasets

C. Spark Core

© 2018 MapR Technologies

Knowledge Check

Match the following:

C. **Spark Core:** Responsible for task scheduling, memory management

A. **SparkSession:** Tells Spark how and where to access a cluster

B. **Datasets:** Collection of objects distributed across many nodes in a cluster

© 2018 MapR Technologies

Answers:
A -> 2
B -> 3
C -> 1

The **SparkSession** tells Spark how and where to access a cluster.

Datasets are a collection of objects that are distributed across many nodes in a cluster.

The **Spark Core** is responsible for task scheduling and memory management.

Learning Goals
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-45

Learning Goals

1.1 Describe Features of Apache Spark

1.2 Define Spark Components

1.3 Describe Spark Data Pipeline Use Cases

© 2018 MapR Technologies

This section provides some use case examples of data pipeline applications using Apache Spark.

Use Case: OLAP Analytics

- Service provider using MapR and Spark delivers real-time multi-dimensional OLAP analytics
- Must accept data of any type/format
- Perform rigorous analytics across large datasets

© 2018 MapR Technologies

A service provider offers services in customer analytics, technology, and decision support to their clients. They are providing real-time, multi-dimensional OLAP analytics. They should be able to accept data of any type or format and perform rigorous analytics across datasets that can go up to 1-2 TBs in size.

Use Case: Operational Analytics

- Health Insurance company uses MapR to store patient information
- Spark computes patient re-admittance probability
- Real-time analytics over NoSQL
- Spark with MapR-DB

© 2018 MapR Technologies

A health insurance company is using MapR to store patient information, which is combined with clinical records. Using real-time analytics over NoSQL, they are able to compute re-admittance probability. If this probability is high enough, additional services, such as in-home nursing, are deployed.

Use Case: Complex Data Pipelining

- Pharmaceutical company uses Spark on MapR for gene sequencing analysis
- Spark reduces processing from weeks to hours
- Complex machine learning without MapReduce

© 2018 MapR Technologies

A leading pharmaceutical company uses Spark to improve gene sequencing analysis capabilities, resulting in faster time to market. Before Spark, it would take several weeks to align chemical compounds with genes. With ADAM, a genomics analysis platform, running on Spark, gene alignment only takes a few hours.

Use Case: Security Services Provider

- Security services stream sensor data through Spark Streaming to find new threats
- Graph processing and machine learning are used to create security predictions
- Additional queries are supported using Spark SQL

© 2018 MapR Technologies

A managed security services provider is ingesting sensor data that is streamed in using Spark Streaming on MapR and is checked for first known threats. The data then goes through graph processing and machine learning for predictions. Additional querying such as results of graph algorithms, predictive models, and summary/aggregate data is done using Spark SQL.

Use Case: Streaming Flight Data

- Flight data streamed through Spark Streaming
- Graph processing analyzes airports and routes
- MLlib uses machine learning to predict delays

© 2018 MapR Technologies

In another use case, we begin with flight data being streamed in. We use Spark SQL to do some analytics on the streaming data and use graph processing to analyze airports and routes that serve them. Machine learning can then be used to predict delays using a classification or decision tree algorithm.

Knowledge Check
R2.2
MAPR ACADEMY ESSENTIALS
© 2018 MapR Technologies
L1-52

Knowledge Check

We have a real-time Twitter feed. We need to build an application that is near real-time and classifies the Twitter feeds based on relevant and not relevant, where "relevant" means that it contains the words "FIFA", "Women's", and "World Cup". Which of the following Apache Spark libraries could we use in the application?

A. Spark SQL

B. Spark Streaming

C. Spark MLlib

D. Spark GraphFrames

© 2018 MapR Technologies

Knowledge Check

We have a real-time Twitter feed. We need to build an application that is near real-time and classifies the Twitter feeds based on relevant and not relevant, where "relevant" means that it contains the words "FIFA", "Women's", and "World Cup". Which of the following Apache Spark libraries could we use in the application?

A. **Spark SQL**

B. **Spark Streaming**

C. **Spark MLlib**

D. Spark GraphFrames

© 2018 MapR Technologies

Use Spark SQL to query the data in DataFrames, Spark Streaming to ingest the live feeds and Spark MLlib to do the classification.

Congratulations! You have completed Lesson 1. Let's move on to Lesson 2, where you will learn to create Datasets.

Welcome to DEV 360, Introduction to Apache Spark, Lesson 2: Create Datasets.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L2-2

Learning Goals

2.1 Define Data Sources, Structures, and Schemas

2.2 Create Datasets and DataFrames

2.3 Convert DataFrames into Datasets

© 2018 MapR Technologies

At the end of this lesson, you will understand the key differences between Datasets and DataFrames, and describe the different data sources and formats available to use with Apache Spark. You will also learn how to define data sources, types, and schemas, load data to create Datasets, and convert DataFrames into Datasets.

Learning Goals

2.1 Define Data Sources, Structures, and Schemas

2.2 Create Datasets and DataFrames

2.3 Convert DataFrames into Datasets

© 2018 MapR Technologies

Let's discuss the importance of defining the schema and data types.

Data Sources

Supports ANY storage source supported by Hadoop

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-5

Before we load data into Spark, let's take a look at some of the data sources that can be used in a Dataset.

You can load data from any storage source that is supported by Hadoop. You can upload data from the local file system, HDFS, MapR-XD, Hive, HBase or MapR-DB, JDBC databases, cloud storage, and any other data source that you are using with your cluster.

Spark provides wrappers for text files, JSON, CSV, Parquet files, SequenceFiles, protocol buffers, and object files. Spark can also interact with any Hadoop-supported formats.

Review: Datasets

- Primary abstraction in Spark
- Collection of objects distributed across a cluster
- Immutable once created
- Fault-tolerant
- Persist or cache in memory or on disk
- Data stored in tabular format

R2.2 © 2018 MapR Technologies

Remember from the previous lesson that Datasets are the primary abstraction in Spark. They are a collection of objects that are distributed across many nodes in a cluster. Once created they cannot be changed, and they are fault-tolerant. If a given node or task fails, the Dataset is reconstructed automatically on the remaining nodes and the job will complete.

You can also persist, or cache, Datasets in-memory or on disk. Datasets store data in a tabular format where each row has multiple columns.

Spark DataFrames

- Primary abstraction in Spark
- Collection of objects distributed across a cluster
- Immutable once created
- Fault-tolerant
- Persist or cache in memory or on disk
- Data stored in tabular format

Unknown Data Type

`Dataset[Row]`

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-8

DataFrame is the name for an untyped Dataset of type row, where “row” is a collection of generic objects. For example, a DataFrame does not understand the difference between a string and integer at compile time, but once it is computed, it can be converted into a Dataset if the schema is defined.

Spark DataFrames

Unknown Schema
Exception: Programmatically assigned schema

Known Schema

© 2018 MapR Technologies

In most cases, the primary distinction between Datasets and DataFrames is whether it has a defined schema. However, there are instances where you can have a DataFrame construct with a defined schema.

In these cases, Spark handles these structures exactly the same. The only difference between them is the name of the object as a Dataset or DataFrame. By standard practice, the variable names associated with either construct should match the object as appropriate, with either DS or DF.

Spark DataFrames

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-10

If our data type and schema are not explicitly defined, the construct will automatically be a DataFrame.

DataFrames can be converted into a Dataset of type “T” using the Case Class and inferring the schema by reflection. Once we know the schema and type for each column, we can call the df.as[T] method to convert it into a Dataset.

Resilient Distributed Datasets

© 2018 MapR Technologies

Resilient Distributed Datasets, or RDDs, are the basic building blocks of Spark. When we build Datasets in our applications, Spark uses RDDs under the hood for final computation.

RDDs can be used for advanced operations in Spark, but these will not be covered in this course. For more information, review the documentation available online.

Scenario: San Francisco Police Department (SFPD) Data

Incident Num	Category	Descript	DayOf Week	Date	Time	PdDistrict	Address
150599321	OTHER_ OFFENSES	POSSESSION_OF_BURGLARY_TOOLS	Thurs	7/9/15	23:45	CENTRAL	JACKSON_ST/ POWELL_ST
156168837	LARCENY/ THEFT	PETTY_THEFT_OF_PROPERTY	Thurs	7/9/15	23:45	CENTRAL	300_Block_of_ POWELL_ST
150599224	OTHER_ OFFENSES	DRIVERS_LICENSE/SUSPENDED_OR_REVOKED	Thurs	7/9/15	23:36	PARK	MASONIC_AV/ GOLDEN_GATE_AV
150599230	VANDALISM	MALICIOUS_MISCHIEF/BREAKING_WINDOWS	Thurs	7/9/15	23:20	NORTHERN	1000_Block_of _POLK_ST

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-12

In this course we will load, test, and discuss data from the San Francisco Police Department, or SFPD, that is in a CSV file in the local file system.

Each row in this file represents an individual incident.

>>Every incident contains an incident number ID with a general category and description of the incident with specific supporting details stored in a tabular format of rows and columns, as shown.

What is Schema?

Schema

IncidentNum	Category	Descript	DayOfWeek
150599321	OTHER_OFFENSES	POSSESSION_OF_BURGLARY_TOOLS	Thursday
156168837	LARCENY/THEFT	PETTY_THEFT_OF_PROPERTY	Thursday
150599224	OTHER_OFFENSES	DRIVERS_LICENSE/SUSPENDED_OR_REVOKED	Thursday
150599230	VANDALISM	MALICIOUS_MISCHIEF/BREAKING_WINDOWS	Thursday

© 2018 MapR Technologies

The schema of a Dataset is the logical visualization of how data stored in your system is organized.

The schema is the structure, described in a formal language that is supported by the database. It provides the blueprint for the tables in the database, and describes the relationships between tables of data.

Why is it Important?

Schema

IncidentNum	Category	Descript	DayOfWeek
150599321	OTHER_OFFENSES	POSSESSION_OF_BURGLARY_TOOLS	Thursday
156168837	LARCENY/THEFT	PETTY_THEFT_OF_PROPERTY	Thursday
150599224	OTHER_OFFENSES	[illegible]RS_LICENSE/SUS[illegible]ED_OR_REVOKED	Thursday
150599230	VANDALISM	MISC[illegible]	[illegible]INDOWS	Thursday

© 2018 MapR Technologies

By default, Spark stores Datasets in a tabular format. This makes it easy to execute queries on data. Generally, there is no specific way you should organize your data but when using CSV or parquet files, be aware that the existing structure defining the table of data must match your defined schema.

Ways of Defining Schema

Construct schema using Case Class

- Use to construct Datasets when columns and types are known at runtime
- Scala Case Class restriction to 22 fields

Construct schema programmatically

- Use to construct Datasets or DataFrames when columns and types are not known until runtime
- Use if schema has more than 22 field

© 2018 MapR Technologies

There are two ways we can define schema in Spark.

Ways of Defining Schema

Construct schema using Case Class

- Use to construct Datasets when columns and types are known at runtime
- Scala Case Class restriction to 22 fields

Construct schema programmatically

- Use to construct Datasets or DataFrames when columns and types are not known until runtime
- Use if schema has more than 22 field

© 2018 MapR Technologies

The Case Class defines the table schema. The names of columns in the source data should match the Spark Case Class that you define.

Note that this method can only be used when the schema has 22 or fewer fields due to a Scala Case Class restriction. When converting a DataFrame into a Dataset, Case Class is also used and is read using reflection, which we will see in the next learning goal.

Ways of Defining Schema

Construct schema using Case Class

- Use to construct Datasets when columns and types are known at runtime
- Scala Case Class restriction to 22 fields

Construct schema programmatically

- Use to construct Datasets or DataFrames when columns and types are not known until runtime
- Use if schema has more than 22 field

© 2018 MapR Technologies

You can also use the programmatic interface to construct the schema for Datasets or DataFrames. You can also use this method when your schema exceeds 22 fields.

Identifying the schema of our data is one of the first steps we take when creating Datasets or DataFrames, which we will discuss in more detail next.

Learning Goals
R2.2
© 2018 MapR Technologies
L2-18

Learning Goals

2.1 Define Data Sources, Structures, and Schemas

2.2 Create Datasets and DataFrames

2.3 Convert DataFrames into Datasets

© 2018 MapR Technologies

Now, let's take a look at some of the ways we can create Datasets and DataFrames, and learn how to use the Spark Interactive Shell.

Datasets vs. DataFrames

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-20

Now that we know that the main differences between a Dataset and DataFrame is when the schema is identified, it is easy to see how the creation of Datasets can branch out into different processes.

Both start with the Spark Interactive Shell, where we will load in data.
>>If we know the schema and data type, we can define the Case Class and it will automatically create a Dataset.

If we do not know the schema for our data, Spark will automatically create a DataFrame. Since we aren't using a known schema, the resulting DataFrame will use generic column headers of "_c0", "_c1", "_c2", and so on.

As mentioned earlier, we can programmatically create a DataFrame with a defined schema.

We can also convert a DataFrame into a Dataset at a later time, using Case Class by reflection with `.as[T]`, which we cover in the next learning goal.

We will discuss these processes in detail, but first let's look at creating a Dataset from the start.

Create Datasets by Importing Data

Operate on variety of data sources through Dataset interface

- Parquet
- JSON
- Hive Tables
- Relational Databases

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-25

A Dataset can be created when importing certain types of data.

Spark SQL can operate on a variety of data sources through the Dataset interface. It includes a schema inference algorithm for JSON and other semi-structured data that enables users to query the data right away.

Spark Interactive Shell

- Write programs interactively
- Uses Scala or Python REPL (Read-Evaluate-Print Loop)

Scala Python

© 2018 MapR Technologies

The Spark Interactive Shell allows you to write programs interactively, using Scala or Python. Note that in this course, we will only be using Scala.

Spark Interactive Shell

- Write programs interactively
- Uses Scala or Python REPL (Read-Evaluate-Print Loop)
- Instant feedback

1)
```
scala> val sfpdDS =
spark.read.text("/spark/lab2/sfpd.csv").as[String]
sfpdDS: org.apache.spark.sql.Dataset[String] =
[value: string]

scala>
```

2)
```
scala> val sfpdDSErr =
spark.read.text(/spark/lab2/sfpd.csv).as[String]
<console>:1: error: ')' expected but '.' found.
val sfpdDSErr =
spark.read.text(/spark/lab2/sfpd.csv).as[String]
                                ^

scala>
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-27

The Spark Interactive Shell provides instant feedback when working on Datasets.

After code is submitted by pressing the enter/return key, feedback is provided by the spark shell as shown in the first screen capture.

>>If there is any syntax error, as shown in image 2, it tells you immediately.

Spark Interactive Shell

- Write programs interactively
- Uses Scala or Python REPL (Read-Evaluate-Print Loop)
- Instant feedback
- **SparkSession** initialized on shell start up

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0-mapr-1707
      /_/
```

SP
Session

© 2018 MapR Technologies

When the shell starts, SparkSession is initialized and is then available to use as the variable "`spark`".

SparkSession can be used to load data into Spark, but is also responsible for launching Spark applications, which we will discuss in lesson 4.

Creating Datasets Using the Spark Interactive Shell

Loading data into Spark using SparkSession, will automatically create a Dataset.

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-29

>>When you load data into Spark using SparkSession, it automatically creates a Dataset.

>>The first Dataset you create is usually called the Base Dataset. Next we will use the Spark Interactive Shell to load data into Spark.

Knowledge Check
R2.2
© 2018 MapR Technologies
L2-30

Knowledge Check

Which of the following is true of the Spark Interactive Shell?

A. Initializes SparkSession and makes it available

B. Available in Java

C. Provides instant feedback as code is entered

D. Allows you to write programs interactively

© 2018 MapR Technologies

Knowledge Check

Which of the following is true of the Spark Interactive Shell?

A. **Initializes SparkSession and makes it available**

B. Available in Java

C. **Provides instant feedback as code is entered**

D. **Allows you to write programs interactively**

© 2018 MapR Technologies

Answers: 1,3,4

Note: Instant Feedback only works with Datasets.

Here are some of the questions we'd like to answer when looking at our data.

1. Which five addresses have the most incidents?
2. Which five districts have the most incidents?
3. What are the top 10 resolutions?
4. What are the top 10 categories?

We use Spark to load and inspect this data in order to answer these questions. Later, we will write machine learning code to do the same thing.

Start Spark Interactive Shell

Scala /opt/mapr/spark/spark-<version>/bin/spark-shell

```
[mapr@maprdemo ~]$ /opt/mapr/spark/spark-2.1.0/bin/spark-shell
17/10/25 02:01:12 WARN SparkContext: Support for Java 7 is deprecated as of Spark 2.0.0
17/10/25 02:01:28 WARN ObjectStore: Failed to get database global_temp, returning
NoSuchObjectException
Spark context Web UI available at http://192.168.100.128.4040
Spark context available as 'sc" (master = local, app id = local-1508922074643).
Spark session available as 'spark'.
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0-mapr-1707
      /_/
```

```
Using Scala version 2.11.8 (Open JDK 64-Bit Server VM, Java 1.7.0_79)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-34

Start the Interactive Shell by running spark-shell from the bin in the Spark install directory.

Create a Dataset

1. Import Classes
2. Define Case Class
3. Load Data
4. Register Dataset as View (optional)

© 2018 MapR Technologies

Once you've started the interactive shell, here are the remaining steps to create a Dataset. Let's look more closely at each of these.

Step 1: Import Classes

```
import spark.implicits._

case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:Double,
Y:Double, pdid:String)

val sfpdDS = spark.read.csv(”/path to file/sfpd.csv").as[Incidents]

sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Once we have started our shell, we need to import `spark.implicits` and all its subclasses.

`spark.implicits` is a built-in library file that contains the definitions of many Spark functions.

Step 2: Define Case Class

```
import spark.implicits._

case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:Double,
Y:Double, pdid:String)

val sfpdDS = spark.read.csv(”/path to file/sfpd.csv").as[Incidents]

sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

We then define the Case Class, as shown here. Remember from earlier that the Case Class is responsible for identifying the schema.

Remember, if your data has more than 22 fields in its schema, you must define your schema programmatically.

Step 3: Load Data

```
import spark.implicits._

case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:Double,
Y:Double, pdid:String)

val sfpdDS = spark.read.csv(”/path to file/sfpd.csv").as[Incidents]

sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

We next load the data into Spark. Note that `Spark` in the code refers to SparkSession. Here, we are using the `Load` method spark.read.csv, since our data is stored in a CSV file.

Loading data without the `.as[T]` function, creates a DataFrame, not a Dataset.

Referring to `.as[Incidents]`, as we do here, defines the schema of the data, creating the resulting Dataset.

Step 4: Register Dataset as View (optional)

```
import spark.implicits._

case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:Double,
Y:Double, pdid:String)

val sfpdDS = spark.read.csv(”/path to file/sfpd.csv").as[Incidents]

sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Finally, you may want to register the Dataset as a view. In this case, we’re calling it sfpd. This is an optional step.

A view allows us to query or visualize the data using our favorite SQL or BI tools.

Details: Load Data

- `spark.read.load("path/filename.parquet")`
  - Data type: Parquet (default)
  - Loads data in path. Default data type is parquet. For other formats, use the `load(path).format` method. Default data type can be configured using `spark.sql.sources.default` property.
- `spark.read.load(path).format(type)`
  - Data type: JSON, Parquet, CSV
  - Loads data in path with type specified in the format method.

Default data source can be configured here: spark.sql.sources.default

© 2018 MapR Technologies

Data can also be loaded directly into a Dataset using generic load methods as shown here.

The default data source is Parquet and will be used for all operations unless otherwise configured as shown here.

Details: Load Data

- `spark.read.text`
  - Data type: Text File
  - Loads a text file and returns Dataset[Row]
- `spark.read.textfile`
  - Data type: Text File
  - Loads a text file and returns Dataset[String]. This method can be used when you want the return type to be Dataset[String] instead of DataFrame (i.e. Dataset[Row])
- `spark.read.jdbc(URL, Table, Connection_Properties)`
  - Data type: Database Table
  - Returns a DataFrame with data from a database table. Used to load data directly from a database table using a JDBC connection.

© 2018 MapR Technologies

This table shows some of the other methods available to load data into Spark, based on the specified data types.

Details: Load Data

- `spark.read.csv(path_to_CSV_file)`
  - Data type: CSV
  - Loads CSV data in path. Similar to `spark.read.load(path).format("csv")`
- `spark.read.json(path_to_JSON_file)`
  - Data type: JSON
  - Loads JSON data in path. Similar to `spark.read.load(path).format("json")`
- `spark.read.parquet(path_to_parquet_file)`
  - Data type: Parquet
  - Loads parquet data in path. Similar to `spark.read.load(path).format("parquet")`

© 2018 MapR Technologies

This table shows some of the other methods available to load data into Spark, based on the specified data types.

Knowledge Check
R2.2
© 2018 MapR Technologies
L2-43

Knowledge Check

You can operate on the following data sources using the Dataset:

A. Parquet Tables

B. JSON

C. Streaming Data

D. JDBC

© 2018 MapR Technologies

Knowledge Check

You can operate on the following data sources using the Dataset:

A. **Parquet Tables**

B. **JSON**

C. Streaming Data

D. **JDBC**

© 2018 MapR Technologies

Answer: A,B,D

Now let's discuss how to create a DataFrame by defining the schema programmatically.

Construct Schema Programmatically

- **Defines table schema**
  - Schema defined at runtime
- **Can be used**
  - When dynamic, based on certain conditions
  - When your Dataset includes more than 22 fields

© 2018 MapR Technologies

If Case Classes cannot be defined ahead of time in order to create a Dataset, you can use this method to create a schema for a DataFrame programmatically.

This method is also used when the schema is dynamic, based on certain conditions. For example, when we want parts of a String to represent different fields, or if we want to parse a text Dataset based on the user.

You would also use this method if you have more than 22 fields, since the Scala Case Class is limited to 22 fields.

Create DataFrame and Construct Schema Programmatically

1. Import Classes
2. Create Schema Programmatically
3. Create DataFrame by Loading Data
4. Register the DataFrame as a Table

© 2018 MapR Technologies

A DataFrame can be created programmatically with the steps shown here.

Step 1: Import Classes

```
import spark.implicits._
import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
StringType,true), StructField("category",StringType,true),
StructField("description",StringType,true),
StructField("dayofweek",StringType,true),
StructField("date",StringType,true), StructField("time",StringType,true),
StructField("pddistrict",StringType,true),
StructField("resolution",StringType,true),
StructField("address",StringType,true), StructField("X",DoubleType,true),
StructField("Y",DoubleType,true), StructField("pdid",StringType,true)))
…
```


© 2018 MapR Technologies

First, we import the necessary classes shown here. We're importing the `spark.implicits` classes as before, and also the SQL data types required for defining `StructField`.

Step 2: Create Schema Programmatically

```
import spark.implicits._
import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
StringType,true), StructField("category",StringType,true),
StructField("description",StringType,true),
StructField("dayofweek",StringType,true),
StructField("date",StringType,true), StructField("time",StringType,true),
StructField("pddistrict",StringType,true),
StructField("resolution",StringType,true),
StructField("address",StringType,true), StructField("X",DoubleType,true),
StructField("Y",DoubleType,true), StructField("pdid",StringType,true)))
...
```


© 2018 MapR Technologies

In this next step, we programmatically define the full schema for `sfpdSchema`. Since the sfpd.csv file has 11 columns, we must specify 11 columns in our initial schema definition, as shown. Whether your file has more or fewer, you must identify this for each column in your original data source.

The `StructType` object creates an array, which is composed of `StructField` objects based on the schema we define. The `StructField` objects contain the column name along with the datatype and an indication of whether it is nullable.

Step 3: Create DataFrame

```
import spark.implicits._
import org.apache.spark.sql.types._
val sfpdSchema = StructType(Array(StructField("incidentnum",
…
val sfpdDF =
spark.read.format("csv").schema(sfpdSchema).load("/spark/lab4/sfpd.csv").t
oDF("incidentnum", "category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")
sfpdDF.createTempView("sfpd")
```


© 2018 MapR Technologies

In this line we create the sfpdDF DataFrame, by loading sfpd.csv data and applying the schema we defined previously.

We also use .toDF with names that match the values that were programmatically associated with sfpdSchema. If these do not match, then it will throw an error.

Step 4: Register the DataFrame as a Table (optional)

```
import spark.implicits._
import org.apache.spark.sql.types._
val sfpdSchema = StructType(Array(StructField("incidentnum",
…
val sfpdDF =
spark.read.format("csv").schema(sfpdSchema).load("/spark/lab4/sfpd.csv").t
oDF("incidentnum", "category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")
sfpdDF.createTempView("sfpd")
```


© 2018 MapR Technologies

Once the DataFrame is created, it can be registered as a view and queried using SQL.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L2-53

Learning Goals

2.1 Define Data Sources, Structures, and Schemas

2.2 Create Datasets and DataFrames

2.3 Convert DataFrames into Datasets

© 2018 MapR Technologies

We will now discuss the process of converting DataFrames into Datasets.

Remember from earlier that you can convert DataFrames to Datasets. To do this, we must define the schema using the Case Class.

Infer Schema by Reflection

Infer schema by reflection
- Using Case Classes
- Use when schema is known

Load Unknown Data
↓
Create DataFrame
↓
Define Case Class
↓
Convert to Dataset

© 2018 MapR Technologies

When a DataFrame uses Case Classes to define the schema of an object, the DataFrame will infer a fixed schema from the Case Classes. Remember that Scala Case Class schema can have to a maximum of 22 fields.

Infer Schema by Reflection: Case Class

Defines table schema

- Column names of DF are matched with names of Case Class using reflection
- Names become name of column

Incident Num	Category	Descript	DayOf Week
150599321	OTHER_OFFENSES	POSSESSION_OF_BURGLARY_TOOLS	Thurs
156168837	LARCENY/THEFT	PETTY_THEFT_OF_PROPERTY	Thurs
150599224	OTHER_OFFENSES	DRIVERS_LICENSE/SUSPENDED_OR_REVOKED	Thurs
150599230	VANDALISM	MALICIOUS_MISCHIEF/BREAKING_WINDOWS	Thurs

© 2018 MapR Technologies

The schema of the Dataset is defined by the source DataFrame used to create it. The names of columns in the DataFrame must match with the Case Class and are read using reflection.

Infer Schema by Reflection: Case Class

Can be

- Nested
- Contain complex data (Sequences or Arrays)

```
case class Address(pddistrict:String,
address:String, pdid:String)

case class Detail(address:Address,
incidentnum:String, category:String,
description:String, resolution:String, X:double,
Y:double)
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-58

Case Classes can be nested and can also contain complex data such as sequences or arrays. As an example, we are showing a code block example of the address Case Class that is nested into another Case Class named detail.

In Scala, a Case Class is equivalent to Plain Old Java Objects (POJO) or JavaBeans.

For more detailed information on this, you may check additional documentation online.

Infer Schema by Reflection

1. Import Classes
2. Create DataFrame by Loading Data
3. Define Case Class
4. Convert DataFrame (Dataset[Rows]) into Dataset (Dataset[T]) using Case Class
5. Register Dataset as view (optional)

© 2018 MapR Technologies

Here are the steps to create a DataFrame and convert it into a Dataset by inferring the schema by reflection.

As before, let's demo of each of these steps.

Step 1: Import Classes

```
import spark.implicits._
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:double,
Y:double, pdid:String)
sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Once you've started the Spark Interactive Shell, we can import the necessary classes such as `spark.implicits`, with all of its subclasses.

Step 2: Create DataFrame by Loading Data

```
import spark.implicits._
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:double,
Y:double, pdid:String)
sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-61

Next, create the base DataFrame by importing the CSV file with your data. The `.toDF` function creates a DataFrame with the specified column names.

Step 3: Define Case Class

```
import spark.implicits._
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:double,
Y:double, pdid:String)
sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Third, define the Case Class. The arguments of the Case Class define the schema of column names and data types of the resulting Dataset.

Step 4: Convert DataFrame to Dataset Using Case Class

```
import spark.implicits._
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:double,
Y:double, pdid:String)
sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Next, convert the base DataFrame into a Dataset using the `.as[T]` method, passing the Case Class name as a parameter to the method. In this example, the incidents Case Class we just defined. Note that the names between Dataset[Row], or .toDF, and the Case Class, or .as[T], or you will get an error.

We can now apply any operations and functions to this Dataset.

Step 5: Register Dataset as View (optional)

```
import spark.implicits._
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
case class Incidents(incidentnum:String, category:String,
description:String, dayofweek:String, date:String, time:String,
pddistrict:String, resolution:String, address:String, X:double,
Y:double, pdid:String)
sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```


© 2018 MapR Technologies

Finally, you can optionally register the Dataset as a view.

Knowledge Check
R2.2
© 2018 MapR Technologies
L2-65

Knowledge Check

Which of the following statements are true when converting a DataFrame to a Dataset?

A. Must infer the schema by reflection

B. Must infer the schema programmatically

C. Must know the schema

D. Must use the Case Class

E. Can be over 22 fields

© 2018 MapR Technologies

Knowledge Check

Which of the following statements are true when converting a DataFrame to a Dataset?

A. **Must infer the schema by reflection**

B. Must infer the schema programmatically

C. **Must know the schema**

D. **Must use the Case Class**

E. Can be over 22 fields

© 2018 MapR Technologies

Answers: A, C, D

Lazy Evaluation

DEFINE DATASET

```
val sfpdDS = spark.read.option("inferSchema",
true).csv("/spark/data/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y",
"pdid").as[Incidents]
```

- Location of data to load
- Defines schema

DEFINE TRANSFORMATIONS

RUN ACTION

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L2-68

>> Datasets are lazily evaluated. We have just defined the instructions to create our Dataset, but the Dataset has not actually been computed at this point.

This definition process tells Spark the location of the data that will be used to create the Dataset, along with the Case Class or schema information it requires.

>> Any additional operations like transformations will also be defined, and
>>then the results will finally get computed and returned only when Spark encounters an action.

In the next lesson, we discuss these additional operations more thoroughly.

Lab
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L2-69

Labs 2.3a and 2.3b

- Estimated time to complete: **50 minutes**
- This lab consists of two parts:
  - In Lab 2.3a (20 minutes) you will load data using the Scala shell, and create Datasets using reflection.
  - In Lab 2.3b (30 minutes) you will implement Word Count using Datasets. Start on this lab after completing 2.3a. If you do not finish Lab 2.3b, you can complete it on your own outside of class.

© 2018 MapR Technologies

Congratulations! You have completed Lesson 2: Create Datasets. Proceed to Lesson 3 to learn about Applying Operations on Datasets in Apache Spark.

Welcome to DEV 360, Introduction to Apache Spark, Lesson 3: Apply Operations on Datasets.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L3-2

Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

© 2018 MapR Technologies

At the end of this lesson, you will be able to apply operations on Datasets and learn how to improve the performance of your system by caching. You will also create user defined functions, and repartition data across your Datasets.

Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

© 2018 MapR Technologies

In this section, we will discuss two types of operations: transformations and actions, which can be applied to Datasets.

Review

Two types of data operations can be performed on a Dataset:

TRANSFORMATION

ACTION

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-5

Now that we have learned to define a Dataset in the previous lesson, we can apply some operations on it.

As we've discussed in Apache Spark Essentials, there are two types of operations you can perform: transformations and actions.

Exploring the Data

- What are the top five addresses with most incidents?
- What are the top five districts with most incidents?
- What are the top 10 resolutions?
- What are the top 10 categories of incidents?

© 2018 MapR Technologies

Here are some of the questions we'd like to ask of the SFPD data. Let's discuss how to apply operations on the Datasets in order to answer these questions.

Dataset Operations: Transformations

© 2018 MapR Technologies

Transformations create a new Dataset from an existing one.

As with the Dataset, transformations are also lazily evaluated, which means they are not computed immediately. A transformation is executed and the new Dataset is created only when an action runs on it.

Commonly Used Transformations

Transformation	Definition
`map()`	Returns new Dataset by applying func to each element of source
`filter()`	Returns new Dataset consisting of elements from source on which function is true
`groupBy()`	Returns Dataset (K, Iterable<V>) where the data is grouped by the given key func.
`reduce()`	Reduces the elements of this Dataset using the specified binary function.
`flatMap()`	Similar to map(), but function should return a sequence rather than a single item
`distinct()`	Returns new Dataset containing distinct elements of source

© 2018 MapR Technologies

Here is a list of some of the most commonly used transformations.

Commonly Used Transformations (cont.)

Transformation	Definition
`cache()`	Cache this Dataset
`columns()`	Returns all column names as an array
`explain()`	Only prints the physical plan to the console for debugging purposes
`persist()`	Persist this DataFrame
`printSchema()`	Prints the schema to the console in a tree format
`createTempView (viewName)`	Registers this Dataset as a temporary view using the given name

© 2018 MapR Technologies

Here are some more commonly used transformations.

Language Integrated Queries

Transformation	Definition
`agg(expr, exprs)`	Aggregates on the entire Dataset without groups
`filter(condition Expr)`	Filters based on given SQL expression
`groupBy(col1, cols)`	Groups Dataset using the specified columns so we can run aggregation on them
`select(cols)`	Selects a set of columns based on expressions

© 2018 MapR Technologies

This table lists commonly used language integrated queries in transformations. These methods are derived from SQL queries.

When a `GroupBy` transformation is executed on a Dataset, it will group the data according to the specified column, and return a new type of Dataset called a Relational Grouped Dataset, a specific object, as shown here.

Dataset Operations: Transformations

```
sfpdDS.groupBy("category")
```

sfpdDS

Dataset

incidentnm	category
150599321	OTHER_OFFENSES
156168837	LARCENY/THEFT
150599224	OTHER_OFFENSES
150599230	VANDALISM

TRANSFORMATIONS

groupBy()

categoryDS

Relational Grouped Dataset

category
OTHER_OFFENSES
LARCENY/THEFT
VANDALISM

© 2018 MapR Technologies

For example, when we execute GroupBy on sfpdDS passing in the parameter name of category, it will group the data by the column category and return a Relational Grouped Dataset, `categoryDS`.

Now recall the general definitions of Transformations and Actions: a transformation performs some operation on an existing Dataset and returns a new Dataset, and an action performs some operation on an existing Dataset and returns a value.

In these specific instances, a common action, let's say `Count` for example, that gets applied to a Relational Grouped Dataset is treated as transformation instead of an action, since it returns a new Dataset and not a direct integer value. To get the actual `Count` value, a final action, such as `Show` or `Collect`, would need to be called in order to return the results.

Class Discussion
R2.2
ACADEMYPRO
© 2018 MapR Technologies
L3-14

Class Discussion

- Scenario: A sample of data in `sfpd.csv` is shown here. Each line represents a particular incident in San Francisco. We only want to look at incidents in the Southern district.

Q. What transformation could we use to get incidents only in the Southern district?

150598652	WARRANTS	WARRANT_A	Thursday	7/9/15	19:32	SOUTHERN
150598652	ASSAULT	THREATS_AG	Thursday	7/9/15	19:32	SOUTHERN
150598652	WEAPON_LA	EXHIBITING_	Thursday	7/9/15	19:32	SOUTHERN
150599343	LARCENY/TH	PETTY_THEF	Thursday	7/9/15	19:30	SOUTHERN
150598583	LARCENY/TH	GRAND_THE	Thursday	7/9/15	19:30	MISSION
150598834	ASSAULT	THREATS_AG	Thursday	7/9/15	19:30	NORTHERN
150598834	OTHER_OFFE	VIOLATION_	Thursday	7/9/15	19:30	NORTHERN
150599014	LARCENY/TH	PETTY_THEF	Thursday	7/9/15	19:15	CENTRAL
156168398	NON-CRIMIN	LOST_PROPE	Thursday	7/9/15	19:15	CENTRAL

© 2018 MapR Technologies

A sample of data in sfpd.csv is shown here. Each line represents a particular incident in San Francisco. We only want to look at incidents in the Southern district.

Q. What transformation could we use to get incidents only in the Southern district?
A. `Filter`

Transformation: `filter()`

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-16

If we wanted to look at incidents only in the Southern district of our SFPD data, we would use the `Filter` transformation on `districtDS`. The `Filter` transformation filters out data based on the specified condition.

Transformation: `filter()`

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-17

Each element in `districtDS` is a line. So we apply the filter to each line of the Dataset.

Transformation: `filter()` with Anonymous Function

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-18

This `Filter` transformation is applying an anonymous function to each element of the Dataset, as indicated here.

The `anonymous function "=>" is preceding the line.contains` syntax, which means that we are applying a function where the input variable is to the left of the `anonymous syntax` operator. So in this example, we are using the input we have created, as line.

Anonymous functions can be used for short pieces of code.

Transformation: `filter()`

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-19

We are applying the `Filter` transformation to `districtDS` where the condition checks to see if the line contains the word “SOUTHERN”. Note, that this field is case sensitive.

Transformation: `filter()`

```
val southernDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-20

If the condition is true, then that line is added to the resulting Dataset, in this case, `southernDS`.

The `Filter` will return the result of the code to the right of the function operator. In this example the output is the result of calling `line.contains` with the condition: does it contain `SOUTHERN`. The variable name is not case sensitive, but the filter value that we are testing is.

Review
R2.2
© 2018 MapR Technologies
L3-21

This shows what we have done so far during the past two lessons:

>> We've created an input or Base Dataset by loading the sfpd.csv file via SparkSession read.text method. Again, this just provides the instructions on how to build the Dataset.

>> Now that we've built our Dataset, remember that it is immutable. We have also applied the `Filter` transformation, which results in another Dataset containing all the elements that satisfy the condition of the filter, in this case, `southernDS`. The `Filter` transformation is simply defining a set of instructions.

>> No transformations are actually executed until an action is called, which we will explore next.

Review

R2.2 © 2018 MapR Technologies L3-23

Remember from ESS 360 that an action on an Dataset returns values to the driver program after running the computation on the Dataset. These transformations are lazy, so they are only computed when an action requires a result to be returned to the driver program.

Commonly Used Actions

Action	Definition
`count()`	Returns the number of elements in the Dataset
`reduce(func)`	Aggregate elements of Dataset using function `func`
`collect()`	Returns all elements of Dataset as an array to driver program
`take(n)`	Returns first *n* elements of Dataset

© 2018 MapR Technologies

Here is a list of some commonly used actions.

Commonly Used Actions (cont.)

Action	Definition
`describe(cols)`	Computes statistics for numeric columns, including count, mean, stddev, min, and max
`show()`	Displays the first 20 rows of DataFrame in tabular form
`first(); head()`	Returns the first row of the Dataset
`takeAsList(n)`	Return first *n* elements of Dataset as list

© 2018 MapR Technologies

This table lists more commonly used actions.

Applying Transformations and Actions

1. Define Dataset
2. Define Transformation
3. Apply an Action
4. Return Value

© 2018 MapR Technologies

Let's put it all together and take a look at how transformations and actions produce a value.

First, step 1 defines what the Base Dataset will look like when it gets loaded in memory, based on our data file. This data file is stored on disk somewhere or as streaming data, depending on the data source.

Step 2 defines what the transformation will be, which in this case, is `Filter`.

Step 3 applies an action and now has to go back and create the Dataset.

Spark will read from the text file, sfpd.csv, and use it to create the Base Dataset. Then the `Filter` transformation is applied to the `Base Dataset` to produce the `New Dataset`, where the action performs the `Count`.

Finally, the `Count` action runs on `southernDS`, sending the value of the total number of elements in the `Dataset` to the driver program.

Review
R2.2
© 2018 MapR Technologies
L3-31

Review

DEFINE DATASET

```
val sfpdDS = spark.read.option("inferSchema",
true).csv("/spark/data/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y",
"pdid").as[Incidents]
```

DEFINE TRANSFORMATIONS

```
val districtDS = sfpdDS.filter("pddistrict = 'SOUTHERN'")
```

RUN ACTION

```
val southernDS = districtDS.count()
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-32

>> Recall from the previous lesson that Datasets are lazily evaluated. We had previously defined the instructions to create our Dataset and

>> we have now defined a `Filter` transformation to apply.

>> Results will only be computed and returned only when Spark encounters an action, which in this case, is `Count`.

Actions on Dataset

Once the action has run and the value returned, the data is no longer in memory.

© 2018 MapR Technologies

Once the action has run and the value returned, >> the data is no longer in memory.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-34

Knowledge Check

Match the Dataset operation to the result:

Result	Dataset Operation
A. Number of distinct categories	1. `incidentsDS.groupBy ”pdDistrict").count()`
B. Number of incidents per district	2. `sfpdDS.show()`
C. First 20 rows in Dataset	3. `sfpdDS.select(”category”). distinct().count()`

© 2018 MapR Technologies

Match the Dataset operation to the result:

Knowledge Check

Match the Dataset operation to the result:

Result	Dataset Operation
A. Number of distinct categories	B. `incidentsDS.groupBy ”pdDistrict").count()`
B. Number of incidents per district	C. `sfpdDS.show()`
C. First 20 rows in Dataset	A. `sfpdDS.select(”category”). distinct().count()`

© 2018 MapR Technologies

Match the Dataset operation to the result:
A -> 3
B -> 1
C -> 2

Top Five Addresses with Most Incidents: Scala

1. Create a Dataset: Group incidents by address
2. Count the number of incidents for each address
3. Sort the result of the previous step in descending order
4. Show the first five which is the top five addresses with the most incidents

© 2018 MapR Technologies

Now, let us use these transformations and actions to find the answers to some of the questions posed earlier. The first question we wanted to answer was:

Which are the five addresses with the most number of incidents?

To answer this, we need to follow these 4 steps:

First, we create a Dataset by grouping the incidents by address.
Then, we apply an action to Count the number of incidents for each address.
We next sort the result of the previous step in descending order,
And finally, we show the first five results which is the top five addresses with the most incidents.

Top Five Addresses with Most Incidents: Scala

```
val incByAddDS=sfpdDS.groupBy("address")
val numAddDS=incByAdd.count
val numAddDesc=numAdd.sort($"count".desc)
val top5Add=numAddDesc.show(5)
```


© 2018 MapR Technologies

The first line is grouping the incidents by address and the second line applies the `Count`. Recall that `GroupBy` returns a Relational Grouped Dataset, thus making the `Count` in this line, a transformation, not an action, because it returns a new Dataset for the called group of addresses.

After line two is executed, the `numAddDS` Dataset contains two columns: address and count.

We can then refer to this new column in line 3 for sorting by using the `$"col_name"` statement,
in this case `$"count"`. This statement is part of the `spark.implicits` library.

Lastly, we show the first five results which provides the top five addresses with the most incidents.

Top Five Addresses with Most Incidents: Scala

```
val incByAdd = sfpdDS.groupBy("address")
                      .count
                      .sort($"count".desc)
                      .show(5)
```


© 2018 MapR Technologies

You can also combine all the previously shown statements into one statement, as shown here.

Top Five Addresses with Most Incidents: SQL

```
val top5Addresses = spark.sql("SELECT address, count(incidentnum) AS
inccount FROM sfpd GROUP BY address ORDER BY inccount DESC LIMIT 5").show
```


© 2018 MapR Technologies

We can also answer this question using SQL. Note that we are selecting from a table in the Dataset registered as sfpd.

The key points to keep in mind here are:

1. One, you need spark.sql and
2. Two, you can use SQL queries against the Dataset registered as a table, hence sfpd here. In the previous statement we used sfpdDS.

Top Five Addresses with Most Incidents: Results

Scala

address	count
800_Block_of_BRYA...	10852
800_Block_of_MARK...	3671
1000_Block_of_POT...	2027
2000_Block_of_MIS...	1585
16TH_ST/MISSION_ST	1512

SQL

address	inccount
800_Block_of_BRYA...	10852
800_Block_of_MARK...	3671
1000_Block_of_POT...	2027
2000_Block_of_MIS...	1585
16TH_ST/MISSION_ST	1512

© 2018 MapR Technologies

As you can see, both return these matching results.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-42

Knowledge Check

Identify whether each statement describes a Transformation, or an Action:

Action

OR

Transformation

A. Returns a Dataset

B. Returns a value

C. Computed lazily

D. Examples include count, take

E. Examples include filter, map

© 2018 MapR Technologies

Knowledge Check

Identify whether each statement describes a Transformation, or an Action:

Action

B. Returns a value

D. Examples include count, take

Transformation

A. Returns a Dataset

C. Examples include count, take

E. Examples include filter, map

© 2018 MapR Technologies

Actions: B, D = 1, 2
Transformations: A, C, E = 3, 4, 5

Class Discussion
R2.2
© 2018 MapR Technologies
L3-45

Class Discussion

```
val incByAddDS = sfpdDF.groupBy("address”).count
.sort($"count".desc).show(5)
```

- What if you want the top 10?
- What if you want category information?

© 2018 MapR Technologies

Consider this statement:

```
val incByAdd = sfpdDF.groupBy("address”).count
.sort($"count".desc).show(5)
```

Ask: “What if you want the top 10?”
A: change show(5) to show(10)

Ask: “What if you want categories?”
A: `groupBy(“category”)`

Output Operations: `save()`

Operation	Description
`save (source, mode, options)`	Saves contents of Dataset based on given data sources, savemode, and set of options
`jdbc (url,name, overwrite)`	Saves contents of Dataset to JDBC at URL under table name table
`parquet (path)`	Saves contents of Dataset as parquet file
`saveAsTable (tablename, source, mode, options)`	Creates a table from contents of Dataset using data source, options, and mode

© 2018 MapR Technologies

There are times when we want to save the results of our queries. We can use these `Save` operations to save data from resulting Datasets using Spark data sources.

Output Operations

To save contents of `top5Addresses` Dataset:

```
top5Addresses.write.format("json").mode("overwrite").save("/user/
user01/test")
```


© 2018 MapR Technologies

As an example, we see that the contents of the `top5Addresses` Dataset is saved in JSON format in a folder called test that is created.

Note, that the parameter overwrite is referring to the file and not the directory. So if the folder exists and you do not include this parameter, you will get an error. With overwrite included, as we show here, you would not get the error.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-49

Knowledge Check

Which of the following will give the top 10 resolutions to the console, assuming that `sfpdDS` is the Dataset registered as a view named `sfpd`?

A. `spark.sql("SELECT resolution, count(incidentnum) AS inccount FROM sfpd GROUP BY resolution ORDER BY inccount DESC LIMIT 10")`

B. `sfpdDS.select("resolution").count.sort($"count".desc).show(10)`

C. `sfpdDS.groupBy("resolution").count.sort($"count".desc).show(10)`

© 2018 MapR Technologies

Knowledge Check

Which of the following will give the top 10 resolutions to the console, assuming that `sfpdDS` is the Dataset registered as a view named `sfpd`?

A. `spark.sql("SELECT resolution, count(incidentnum) AS inccount FROM sfpd GROUP BY resolution ORDER BY inccount DESC LIMIT 10")`

B. `sfpdDS.select("resolution").count.sort($"count".desc).show(10)`

C. `sfpdDS.groupBy("resolution").count.sort($"count".desc).show(10)`

© 2018 MapR Technologies

Answer: A, C
A→ using SQL
C→ using Dataset operations

B will only select/display all the column data

Lab
R2.2
© 2018 MapR Technologies
L3-52

Lab 3.1: Explore and Save SFPD Data

- Estimated time to complete: **20 minutes**
- In this lab, you will use SQL and Dataset operations to explore the SFPD data that you loaded in Lesson 2.

© 2018 MapR Technologies

We have answered the first question, but what about the others? We can use similar logic to answer the remaining questions which will be done in your lab exercises.

✓ What are the top five addresses with most incidents?

- What are the top five districts with most incidents?
- What are the top 10 resolutions?
- What are the top 10 categories of incidents?

In this lab, you will explore the dataset using Dataset operations and SQL queries. You will also `Save` from the Dataset.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L3-54

Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

© 2018 MapR Technologies

In the next section, we will discuss the reasons and process for, caching Datasets.

To understand why we should cache Datasets, let's walk through this lineage graph, which follows the flow of Spark code execution.

Since Datasets are lazily evaluated, the Base Dataset, `sfpdDS` in the example shown here, is not created until we call an action on any Dataset in this graph.

Every time we call an action on a Dataset, Spark will need to compute the Dataset and all its dependencies.

For example, when we call `Count` for the total number of incidents, data will be loaded into the `sfpdDS` and then the `Count` action is applied.

Lineage Graph: `count()`

R2.2 © 2018 MapR Technologies

Once the value is returned, the data is no longer in memory.

Similarly for the `Count` action, to get the total number of incidents in the Southern district, first Spark will compute both `sfpdDS` and `southernDS`, `going all the way back to retrieve the initial data from sfpd.csv`, and then perform `Count`.

Remember that you can combine transformations and actions in any order, to process and analyze your data.

So when you call the `Collect` function for the total incidents per category, or the `Reduce` function to find the max number of incidents, the data is loaded into `sfpdDS`, and `categoryDS` is computed.

Then the respective action operates on that New Dataset and the results are returned. The `Collect and Reduce` actions will each recompute any needed Datasets from the start.

When there is a branching in the lineage, and we are using the same Dataset multiple times, it is advisable to cache or persist the Dataset(s). The Dataset has already been computed and the data is already in memory.

As shown here, `sfpdDS` isn't in memory anymore, but since we've cached `categoryDS`, seen here in blue, we no longer need any previous Datasets that were originally required. We can reuse this Dataset without using any additional compute or memory resources.

We need to be careful not to cache everything, only those that are being iterated, as the cache behavior depends on available memory. If the file does not fit in memory, then `Count` reloads the data file, and then proceeds as normal.

Caching a Dataset

1. Import Classes
2. Define DataFrame
3. Define Transformation
4. Cache Dataset
5. Apply Action

© 2018 MapR Technologies

Much of this process will be familiar from the last lesson, as we start by loading necessary classes and data in the same way. To cache Datasets, we will follow these steps.

Step 1: Import Classes

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
```

R2.2 © 2018 MapR Technologies L3-64

Just as when loading data into a DataFrame, the first line imports the necessary classes.

Step 2: Define DataFrame

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
```


© 2018 MapR Technologies

The next lines define all of the instructions on how to create the DataFrame, though the file is not yet read.

Note, that in this case, we use the .toDF to provide names the column headers, which simply creates a more user-friendly DataFrame by allowing the values to be returned with these defined column headers. If we do not identify these names here, the column headers will appear generically as “_c0”, “_c1”, and so on, for each column.

Step 3: Define Transformation

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
```


© 2018 MapR Technologies

The third line says to apply a transformation to the Base DataFrame, `sfpdDF`, to create the new variable: category, as `categoryDS`. As we are using the GroupBy transformation, this creates a Relational Grouped Dataset.

Step 4: Cache Dataset

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
```


© 2018 MapR Technologies

The fourth line says to cache the contents of the new `categoryDS`.

We may also use `dataset.persist(MEMORY_ONLY)` or `dataset.persist(MEMORY_AND_DISK)`. The first method is the same as Cache, storing the cache in memory only, versus the second option here, which stores it on disk.

Step 5: Apply Action

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
```


© 2018 MapR Technologies

The last line calls forth the collect action on the `categoryDS` dataset after counting the number of rows in each category group. Once `Collect` is called, execution starts.

At this point, sfpd.csv is read, the transformation is applied, the data is cached, and the value collected.

The next time we call an action will now just use the data from the cache, rather than re-loading the file and performing the first transformation again.

Best Practices

Release unneeded cached Datasets:

```
unpersist()
```

If not enough memory is available, application may crash

© 2018 MapR Technologies

If you no longer need a cached Dataset, it is always better to release that memory for optimal performance. You can use `Unpersist` to uncache.

If there is not enough memory on your system, the application may crash when attempting to cache a Dataset.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-70

Knowledge Check

When considering the caching of Datasets, which of the following are true?

A. When there is branching in lineage, it is advisable to cache the Dataset

B. Use `dataset.cache()` to cache the Dataset

C. Cache behavior depends on available memory. If not enough memory, then action will reload from file instead of from cache

D. `dataset.persist(MEMORY_ONLY)` is the same as `dataset.cache()`

E. All of the above

© 2018 MapR Technologies

Knowledge Check

When considering the caching of Datasets, which of the following are true?

A. When there is branching in lineage, it is advisable to cache the Dataset

B. Use `dataset.cache()` to cache the Dataset

C. Cache behavior depends on available memory. If not enough memory, then action will reload from file instead of from cache

D. `dataset.persist(MEMORY_ONLY)` is the same as `dataset.cache()`

E. All of the above

© 2018 MapR Technologies

Answer: E

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L3-73

Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

© 2018 MapR Technologies

In this section, we will create and use user defined functions, also known as UDFs.

User Defined Functions (UDFs)

- Write your own functions
- In Spark, can define UDFs inline
- No complicated registration or packaging process

© 2018 MapR Technologies

User defined functions are custom functions written by you, the developer.

In Spark, you can define these functions inline and there is no complicated registration or packaging process.

UDF Types

- Two types of UDFs:
  - Used with Scala (Dataset operations)
  - Used with SQL

Scala

SQL

© 2018 MapR Technologies

There are two types of UDFs: one that can be used with Scala with Dataset operations, and the other that can be used with SQL.

Scenario: Find Incidents by Year

- Date in the format: “dd/mm/yy”
- Need to extract string after last slash
- Can then compute incidents by year

Incident Num	Category	Descript	Day OfWeek	Date	Time
150599321	OTHER_OFFENSES	POSSESSION_OF_BURGLARY_TOOLS	Thursday	7/9/15	23:45
156168837	LARCENY/THEFT	PETTY_THEFT_OF_PROPERTY	Thursday	7/9/15	23:45
150599224	OTHER_OFFENSES	DRIVERS_LICENSE/SUSPENDED_OR_REVOKED	Thursday	7/9/15	23:36
150599230	VANDALISM	MALICIOUS_MISCHIEF/BREAKING_WINDOWS	Thursday	7/9/15	23:20

© 2018 MapR Technologies

Let’s consider this example. We want to find the number of incidents by year. The date field in our SFPD data is a string in the form of day, month, year, or 7/9/15, for example.

In order to group or do any aggregation by year, we will first need to extract the year from the string, and then compute the number of incidents by year.

Scala: Define UDF

- Inline creation
  - Use `udf()`
- Used with Dataset operations

```
val getStr = udf((arguments)=>{function definition})
```

Scala

© 2018 MapR Technologies

You can define the function as a UDF inline.

Here we are creating a function called `getStr` by using `udf in Scala`.

Scala: Define Function

```
val getStr = udf((s:String)=>{
val lastS = s.substring(s.lastIndexOf('/')+1)
lastS
})
val yy = sfpdDS.groupBy(getStr(sfpdDS("date")))
              .count
              .show
```


© 2018 MapR Technologies

And here, we define the function.

Scala: Use UDF

```
val getStr = udf((s:String)=>{
val lastS = s.substring(s.lastIndexOf('/')+1)
lastS
})
val yy = sfpdDS.groupBy(getStr(sfpdDS("date")))
              .count
              .show
```


© 2018 MapR Technologies

Then, we simply use the UDF in Dataset operations.

Scala Results: Find Incidents by Year

```
scalaUDF(date) count
13             152830
14             150185
15             80760
```


© 2018 MapR Technologies

This shows the results: the number of incidents by year, from 2013 to 2015.

SQL: Define and Register UDF

- To register and create inline, use:

```
spark.udf.register("funcname", func definition)
```

- Use in SQL statements

SQL

© 2018 MapR Technologies

Now let's look at how to create a UDF using SQL.

You define the function inline in the registration using `spark.udf.register`.
This accepts two arguments: the function name as a literal, and the function definition.

SQL: Register UDF

```
spark.udf.register("getStr", (s:String)=>{
val strAfter=s.substring(s.lastIndexOf('/')+1)
strAfter
})
```


© 2018 MapR Technologies

First, we register the UDF using method `spark.udf.register`, which takes the two parameters: UDF name and UDF definition. Here the UDF name is "`getStr`".

SQL: Define Function

```
spark.udf.register("getStr", (s:String)=>{
val strAfter=s.substring(s.lastIndexOf('/')+1)
strAfter
})
```


© 2018 MapR Technologies

Next, we define the function of our UDF. In this case, we want to extract the year from the string.

As we've seen, the date in the SFPD data is a string in the format of day, month, year. Our custom UDF will now pull the last two characters from the date and return the value as string.

SQL: Use in SQL Statements

```
val numIncByYear = spark.sql("SELECT getStr(date),
count(incidentnum) AS countbyyear
FROM sfpd GROUP BY getStr(date)
ORDER BY countbyyear DESC
LIMIT 5")
```


© 2018 MapR Technologies

This custom function can now be used in SQL statements.

SQL Results: Find Incidents by Year

```
numIncByYear.show
[13,152830]
[14,150185]
[15,80760]
```


© 2018 MapR Technologies

As you can see, we get the same result as before for the number of incidents by year.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-87

Knowledge Check

To use a UDF in a SQL query, the function in Scala:

A. Must be defined inline using `udf(<function definition>)`

B. Must be registered using `spark.udf.register`

C. Only needs to be defined as a function

D. Only needs a definition

© 2018 MapR Technologies

Knowledge Check

To use a UDF in a SQL query, the function in Scala:

A. Must be defined inline using `udf(<function definition>)`

B. Must be registered using `spark.udf.register`

C. Only needs to be defined as a function

D. Only needs a definition

© 2018 MapR Technologies

Answer: B

Lab
R2.2
© 2018 MapR Technologies
L3-90

Lab 3.3: Create and Use User-Defined Functions

- Estimated time to complete: **20 minutes**
- In this lab, you will create a UDF that extracts the year from the data field in the SFPD data. You will then register and use the UDF to query SFPD data by year.

© 2018 MapR Technologies

In this activity, you will create and use a user defined function.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L3-92

Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

© 2018 MapR Technologies

In this section, we look at repartitioning Datasets.

Why Repartition?

- Spark partitions are basic units of parallelism, which help distribute data across the cluster
- Spark automatically partitions data, but this setting can be manually controlled when needed

R2.2 © 2018 MapR Technologies L3-94

A Dataset is divided into a set of partitions where each partition contains a subset of the data.

In Spark, partitions are basic units of parallelism which help to parallelize the computation by distributing data across the cluster and thus optimizing the performance.

Spark automatically partitions data but you may sometimes want to change the number of partitions depending on your particular setup. The number of cores in the executor nodes and/or the size of your data may be instances in which you'd want to manually adjust your partitioning to achieve optimal parallelism.

Partition Datasets

- Sets number of partitions in Datasets after shuffle:

```
spark.sql.shuffle.partitions
```

- Default value set to 200
- Can change parameter using:

```
spark.conf.set
```

- Internally SparkSQL partitions data for joins and aggregations
- If applying other operations on result of Dataset operations, can manually control partitioning

© 2018 MapR Technologies

In Spark SQL, there is a parameter called `spark.sql.shuffle.partitions`, which sets the number of partitions in a Dataset after a shuffle. By default, this value is set to 200. We can change this parameter by using the function `spark.conf.set`.

Internally, Spark SQL will add exchange operators to make sure that data is partitioned correctly for joins and aggregations.

If we want to apply other operations on the result of the Dataset operations, we have the option to manually control the partitioning.

Partition Dataset

Example – SFPD dataset with four partitions

P1			P2			P3			P4		
Incnum (Str)	Category (Str)	PdDistrict (Str)	Incnum (Str)	Category (Str)	PdDistrict (Str)	Incnum (Str)	Category (Str)	PdDistrict (Str)	Incnum (Str)	Category (Str)	PdDistrict (Str)
150598981	ASSAULT	CENTRAL	150599183	ASSAULT	SOUTHERN	150597701	ASSAULT	MISSION	150597400	ROBBERY	TARAVAL
150599161	BURGLARY	PARK	150599246	ASSAULT	CENTRAL	150597701	ROBBERY	INGLESIDE	150596468	FRAUD	SOUTHERN
150599127	SUSPICIOUS	SOUTHERN	150599246	WARRANTS	CENTRAL	150597701	ASSAULT	SOUTHERN	150597234	BURGLARY	SOUTHERN
150603455	VANDALISM	NORTHERN	150599246	WARRANTS	CENTRAL	150597591	ROBBERY	SOUTHERN	150596468	FRAUD	TARAVAL

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L3-96

Let's consider the example of our Dataset containing the SFPD data. Since this is a small amount of data, we would want to change the partitions from the default value of 200, down to something much smaller, like 4, to increase parallelism.

Partitioning

- To determine current number of partitions:

```
ds.rdd.partitions.size()
```

- To repartition, use

```
ds.repartition(numPartitions)
```


© 2018 MapR Technologies

First, we needed to determine or verify the current number of partitions, using the method shown here: `ds.rdd.partitions.size`.

Next, we would simply change the repartitions, by using `ds.repartition(4)` to repartition our Dataset down to four.

Knowledge Check
R2.2
© 2018 MapR Technologies
L3-98

Knowledge Check

In Scala, to find the number of partitions in the Dataset, use:

A. `ds.numPartitions()`

B. `ds.rdd.partitions.size()`

C. `df.partitions.size()`

D. `df.partitionnumber()`

© 2018 MapR Technologies

Knowledge Check

In Scala, to find the number of partitions in the Dataset, use:

A. `ds.numPartitions()`

B. **`ds.rdd.partitions.size()`**

C. `df.partitions.size()`

D. `df.partitionnumber()`

© 2018 MapR Technologies

Answer: B

Lab
R2.2
© 2018 MapR Technologies
L3-101

Lab 3.4: Analyze Data Using UDFs and Queries

- Estimated time to complete: **30 minutes**
- In this lab, you will build a standalone application that uses what you have learned so far. You will create a Dataset, create a UDF to extract the year 2015, query some SFPD data for that year, and save the results to a JSON file.

© 2018 MapR Technologies

Congratulations! You have completed Lesson 3: Apply Operations on Datasets.

Continue on to DEV 361 - Build and Monitor Apache Spark Applications, with Lesson 4: Build a Simple Apache Spark Application.

Welcome to DEV 361, Build and Monitor Apache Spark Applications – Lesson 4: Build a Simple Apache Spark Application.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-2

Learning Goals

4.1 Define the Spark Program Lifecycle

4.2 Define the Function of SparkSession

4.3 Describe Ways to Launch Spark Applications

4.4 Launch a Spark Application

© 2018 MapR Technologies

At the end of this lesson, you will understand the Spark architecture and be able to define the full Spark program lifecycle. You will be able to define the function of SparkSession, and describe the various modes used to launch a Spark application.

Learning Goals

4.1 Define the Spark Program Lifecycle

4.2 Define the Function of SparkSession

4.3 Describe Ways to Launch Spark Applications

4.4 Launch a Spark Application

© 2018 MapR Technologies

First, let's define the Spark program lifecycle and take a detailed look at the Spark execution process.

Review
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-5

We've learned that the overall lifecycle of a Spark program looks something like this.

First, we create some input Datasets from external data, or parallelize a collection of existing data into a Dataset.
These are lazily transformed to define new Datasets using transformations.
We can then ask Spark to `Cache` any intermediate Datasets that will need to be reused.
And lastly, we launch actions, such as `Count` and `Collect`, to kick off a parallel computation, which is then optimized and executed by Spark.

In Apache Spark Essentials, we reviewed this diagram of Spark components. Recall that a Spark cluster consists of two processes, a driver program and multiple worker nodes each running an executor process. The driver program runs on the driver machine, commonly an edge node, and the worker programs run on cluster nodes or in local threads.

Let's take a closer look at what's happening during this lifecycle and execution launch process.

Components of Distributed Spark Application

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-8

Spark uses a master-slave architecture where there is one central coordinator, known as the driver, and many distributed workers called executors.

The driver runs in its own Java process and each executor also runs in its own, individual Java process.

The driver, together with its executors, are referred to together as a Spark application.

To run a Spark application, we follow these basic steps. First, the user submits an application using `spark-submit`.

Next, `spark-submit` launches the driver program, which invokes the `Main` method. The `Main` method creates SparkSession which tells the driver the location of the cluster manager.

The driver then contacts the cluster manager to request resources and to launch executors.

The cluster manager launches executors for the driver program.

Now, the driver runs through the application: it creates SparkSession, and defines the Datasets and transformations, and then applies any actions. Work, in the form of Jar or Python files, is then sent to the executors in the form of tasks.

Tasks then run on an executor.

And finally, when the `Main` method exits or `SparkSession.stop` is called, the resources are released from cluster manager and the executors are terminated.

Knowledge Check
R2.2
© 2018 MapR Technologies
L4-16

Knowledge Check

Place the steps of a Spark application lifecycle in the right order

A. `val burglaries=sfpdDS.filter(line=>line.contains("BURGLARIES"))`

B. `val sfpdDS=spark.read.textFile("/user/user01/data/sfpd.csv")`

C. `val totburglaries=burglaries.count()`

D. `burglaries.cache()`

© 2018 MapR Technologies

Knowledge Check

Place the steps of a Spark application lifecycle in the right order:

```
val
sfpdDS=spark.read.textFile("/user/user01/data/sfpd.csv")

val
burglaries=sfpdDS.filter(line=>line.contains("BURGLARIES"
))

burglaries.cache()

val totburglaries=burglaries.count()
```


© 2018 MapR Technologies

Correct order shown here.
Previous slide answer key: B, A, D, C

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-19

Learning Goals

4.1 Define the Spark Program Lifecycle

4.2 Define the Function of SparkSession

4.3 Describe Ways to Launch Spark Applications

4.4 Launch a Spark Application

© 2018 MapR Technologies

In this section, we take a look at the function of SparkSession.

Review
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-21

As we established during our discussion on Spark components, SparkSession is the starting point of any Spark program. It tells Spark how and where to access the cluster.

To create and manipulate Datasets, we use various methods in SparkSession.

Build a Standalone Application

1. Import Class
2. Define Class
3. Create `SparkSession` Object
4. Import `spark.implicits` Using SparkSession
5. Declare New Variable and Assign Path to Data File
6. Create Dataset (using `read` method)

© 2018 MapR Technologies

When you are building an application through an IDE or driver machine, you must write out the complete program, compile, and execute it in its entirety in order to complete the job.

This is different from using the Spark Interactive Shell, which provides instant feedback during the process of creating a Dataset.

Let's demo of each of these steps.

Import Class

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
         ……
    }
}
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-24

If you are building a standalone application, you will need to create the SparkSession. In order to do this, you must first import `SparkSession` as shown here.

Define Class

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
         ……
    }
}
```


© 2018 MapR Technologies

The next step is to define the class, note that this object must have the same name as the file or you will get an error because it will not be able to compile the code.

Creating SparkSession Object

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
         ……
    }
}
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-26

We now create a `SparkSession` object within the `Main` method and assign it to the variable "spark."

When doing this, you create a new SparkSession object and can set certain parameters, such as the application name, as shown here.

Note, the application should define a Main method instead of extending `scala.App`.

Import `spark.implicits` Using `SparkSession`

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
         ……
    }
}
```


© 2018 MapR Technologies

Next, using `SparkSession`, we import `spark.implicits` and all of its subclasses.

Declare New Variable and Assign Path

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
          ……
    }
}
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-28

We declare a new variable, `sfpdFile`, that points to the location of the sfpd.csv.

This is the data that will be used to build the Base Dataset.

Create Dataset

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
     import spark.implicits._
              /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
          ……
    }
}
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-29

We then use the `SparkSession` `spark.read.format` method to create the DataFrame.

Note that in this example, we are choosing to cache the DataFrame, because we are expecting to reuse it. This is an optional addition. Next, we will discuss how to launch a Spark application.

Knowledge Check
R2.2
© 2018 MapR Technologies
L4-30

Knowledge Check

Which of the following statements about `SparkSession` are TRUE?

A. `SparkSession` is the starting point of any Spark program

B. In a standalone Spark application, you don't need to initialize `SparkSession`

C. Interactive Shell (Scala and Python) initializes `SparkSession`

© 2018 MapR Technologies

Knowledge Check

Which of the following statements about `SparkSession` are TRUE?

A. **`SparkSession` is the starting point of any Spark program**

B. In a standalone Spark application, you don't need to initialize `SparkSession`

C. **Interactive Shell (Scala and Python) initializes `SparkSession`**

© 2018 MapR Technologies

Answers: 1 and 3

Lab
R2.2
© 2018 MapR Technologies
L4-33

Lab 4.2: Import and Configure Application Files

- Estimated completion time: **25 minutes**
- In this lab, you will download application files and import them into the IntelliJ IDE for use in a subsequent lab.

© 2018 MapR Technologies

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-35

Learning Goals

4.1 Define the Spark Program Lifecycle

4.2 Define the Function of SparkSession

4.3 Describe Ways to Launch Spark Applications

4.4 Launch a Spark Application

© 2018 MapR Technologies

Now, let's see how we can launch Spark applications.

Ways to Launch a Spark Application

Launch Mode	Runtime Environment
1. Local	Runs in the same JVM
2. Standalone	Simple cluster manager
3. Hadoop YARN	Resource manager in Hadoop 2
4. Apache Mesos	General cluster manager

© 2018 MapR Technologies

To launch Spark applications on a cluster, Spark can either run in local mode, or use its own built-in standalone cluster manager. Because the cluster manager is a pluggable component in Spark, Spark can also run on various external managers, such as YARN or Mesos.

Mode 1: Local

- Driver program and workers in same JVM
- Datasets and variables in same memory space
- No central master
- Execution started by user

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-38

In local mode, there is only one JVM. The driver program and the workers are in the same JVM.

Within the program, any Datasets that are created are in the same memory space. There is no central master in this case and the execution is started by the user.

Mode 1: Local

Local mode is useful for prototyping, developing, debugging, and testing.

© 2018 MapR Technologies

Because everything is kept together in one JVM, local mode is useful for prototyping, developing, debugging, and testing.

Since this is mostly used for testing, only one node would be used as opposed to the typical cluster of nodes in a normal production environment. Because of this, we will only need specify the number of cores instead of nodes when launching an application, which will we cover later on in this lesson.

Mode 2: Standalone

- Simple standalone deploy mode
- Install by placing compiled version of Spark on each cluster node
- Can be used on multiple nodes

R2.2 © 2018 MapR Technologies

Spark provides a simple standalone deploy mode, which is the easiest to set up and is useful if only running Spark.

To install Spark Standalone mode, you simply place a compiled version of Spark on each node on the cluster. This mode can be used on multiple nodes.

Mode 2: Standalone

```
import org.apache.spark.sql.SparkSession

val spark =
SparkSession.builder.master("spark://<host_IP>:7077").appName(
"sfpdApp").getOrCreate()
```


© 2018 MapR Technologies

To run an application on the Spark cluster, simply pass the URL of the Spark master as a parameter to the SparkSession Builder's master method. The default port for Spark master is 7077.

Standalone: Deployment Modes

Cluster Mode	Client Mode
Driver launched from worker process inside cluster	Driver launches in the client process that submitted the job
Asynchronous: can quit without waiting for result	Synchronous: must wait for result

If using `spark-submit` → application automatically distributed to all worker nodes

© 2018 MapR Technologies

For standalone clusters, Spark supports two deploy modes: cluster and client modes.

In cluster mode, the driver is launched from one of the worker processes inside the cluster. The client process exits as soon as it fulfills its responsibility of submitting the application, without waiting for the application to finish.

Standalone: Deployment Modes

Cluster Mode	Client Mode
Driver launched from worker process inside cluster	Driver launches in the client process that submitted the job
Asynchronous: can quit without waiting for result	Synchronous: must wait for result

If using `spark-submit` → application automatically distributed to all worker nodes

© 2018 MapR Technologies

In client mode, the driver is launched in the same process as the client that submits the application.

For either mode, if your application is launched through `spark-submit`, then the application jar is automatically distributed to all worker nodes.

Mode 3: Hadoop YARN

- Run on YARN if you have Hadoop cluster
  - Uses existing Hadoop cluster
  - Uses features of YARN scheduler

© 2018 MapR Technologies

It is advantageous to run Spark on YARN if you have an existing Hadoop cluster. You can use the same cluster for your Hadoop jobs without having to maintain a separate Spark cluster.

In addition, you can take advantage of the features of the YARN scheduler for categorizing, isolating, and prioritizing workloads.

Hadoop YARN: Deployment Modes

Cluster Mode	Client Mode
Driver launched in Application Master in cluster or worker	Driver launched in the client process that submitted the job
Can quit without waiting for job results (async)	Need to wait for result when job finishes (sync)
Suitable for production deployments	Useful for Spark interactive shell or debugging

© 2018 MapR Technologies

Just as with Standalone mode, there are two deploy modes that can be used to launch Spark applications on YARN: cluster mode and client mode.

In YARN cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster. Running in YARN cluster mode is an asynchronous process, meaning that you can quit without waiting for the job results.

The YARN cluster mode is suitable for long running jobs, such as for production deployments.

Hadoop YARN: Deployment Modes

Cluster Mode	Client Mode
Driver launched in Application Master in cluster or worker	Driver launched in the client process that submitted the job
Can quit without waiting for job results (async)	Need to wait for result when job finishes (sync)
Suitable for production deployments	Useful for Spark interactive shell or debugging

© 2018 MapR Technologies

In YARN client mode, the driver runs in the client process and the application master is only used for requesting resources from YARN. Running in YARN client mode is a synchronous process, so you have to wait for the result when the job finishes.

YARN client mode is useful when using the interactive shell, for debugging, and testing.

Mode 4: Apache Mesos Cluster Manager

- Mesos Master replaces Spark Master as Cluster Manager
- Driver creates job and Mesos determines what machines handle what tasks
- Multiple frameworks can coexist on same cluster

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-47

The Apache Mesos cluster manager uses its own Mesos master to replace the Spark master in determining resource scheduling features for Spark and other applications.

Now when a driver creates a job and starts issuing tasks for scheduling, Mesos determines what machines handle which tasks.

Because it takes into account other frameworks when scheduling these many, short-lived tasks, multiple frameworks can coexist on the same cluster without resorting to a static partitioning of resources.

Apache Mesos: Deployment Modes

Coarse-grained Mode (Default)
Launches one task on each Mesos machine
No sharing between users
Much lower startup overhead

© 2018 MapR Technologies

Spark can run over Mesos in coarse-grained mode and dynamic resource allocation.

The default “coarse-grained” mode will launch only one long-running Spark task on each Mesos machine, and dynamically schedule its own “mini-tasks” within it. The benefit is much lower startup overhead, but at the cost of reserving the Mesos resources for the complete duration of the application.

Apache Mesos: Deployment Modes

Coarse-grained Mode (Default)
Launches one task on each Mesos machine
No sharing between users
Much lower startup overhead

Dynamic Resource Allocation
Each Spark task runs as a separate Mesos task

Benefits when enabled:

- Resources allocated dynamically for Spark tasks
- Scales up/down automatically based on running tasks

© 2018 MapR Technologies

Dynamic resource allocation is a mode available in coarse-grained mode.

By enabling dynamic resource allocation, resources get allocated dynamically for Spark tasks and once done, resources are released back to the pool. This is beneficial because it automatically scales up and down as required based on the running tasks.

Class Discussion
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-50

Class Discussion

Based on what you've learned about the different launch modes available, which mode would you use and why? What are some of the advantages offered by these different modes?

© 2018 MapR Technologies

Note for discussion: Label-based scheduling is available with YARN, offering one advantage to YARN launch mode.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-52

Learning Goals

4.1 Define the Spark Program Lifecycle

4.2 Define the Function of SparkSession

4.3 Describe Ways to Launch Spark Applications

4.4 Launch a Spark Application

© 2018 MapR Technologies

We will now learn how to launch a Spark application.

Launch a Program: Package Application

Package application and dependencies into a .jar file (SBT or Maven)

© 2018 MapR Technologies

The first step to launch a Spark application is to package your application and any dependencies into a .jar file. For Scala apps, you can use SBT or Maven.

Launch a Program: Use `spark-submit`

Use `./bin/spark-submit` to launch application

```
./bin/spark-submit \
  --class <main-class>
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```


© 2018 MapR Technologies

Once you have packaged your application, you should have the jar file, and you can now launch the application using `spark-submit`.

The `spark-submit` script is in the Spark home/bin directory, and is used to run an application in any of the modes we previously discussed.

We can specify many different options when calling `spark-submit`, with some common examples shown here. Depending on the mode in which you are deploying, you may also have other options.

For more detailed information, review the Apache Spark documentation online.

Mode 1: Local

Run local mode on n cores:

```
./bin/spark-submit --class <classpath> \
--master local[n] \
/path/to/application-jar
```


© 2018 MapR Technologies

Here is an example of running a Spark application in local mode on n number of cores. As we mentioned before, only cores need to be specified since this mode is used mostly for testing, and not in large scale production environments.

Mode 2: Standalone

Run Standalone client mode:

```
./bin/spark-submit --class <classpath>\
-- master spark:<master url> \
/path/to/application-jar
```


© 2018 MapR Technologies

This example shows you how to run in the Standalone client mode.

You can launch a standalone cluster either manually, by starting a master and workers by hand, or use launch scripts provided by Apache Spark.

It is also possible to run these daemons on a single machine for testing.

Mode 3: Hadoop YARN

Run on YARN cluster mode:

```
./bin/spark-submit --class <classpath> \
--master yarn-cluster \
/path/to/application-jar
```

Run on YARN client mode:

```
./bin/spark-submit --class <classpath> \
--master yarn-client \
/path/to/application-jar
```


© 2018 MapR Technologies

Unlike in Spark standalone mode, in which the master's address is specified in the "master" parameter, in YARN mode the ResourceManager's address is picked up from the Hadoop configuration. Thus, the master parameter is simply "yarn-client" or "yarn-cluster".

To launch a Spark application in yarn-cluster mode, use the code shown here.
`./bin/spark-submit` --class path.to.your.Class --master yarn-cluster [options] <app jar> [app options]

To launch a Spark application in yarn-client mode, do the same, but replace "yarn-cluster" with "yarn-client".

Mode 4: Apache Mesos

Run with Apache Mesos:

```
$ /opt/mapr/spark/spark-2.1.0/bin/spark-submit \
--class "AuctionsApp" \
--master mesos://<host_IP>:5050 \
/path/to/auctions-project_2.11-1.0.jar
```


© 2018 MapR Technologies

To launch a Spark application using Apache Mesos, you would use this code with your host IP. The default port for a Mesos cluster is 5050.

Monitor the Spark Job

SparkHistoryServer Direct Port Access
- http://<ip address>:18080

Monitor Spark Applications in Real-Time
- http://<ip address>:4040

MapR Control System (MCS)
- http://<ip address>:8443
- Spark History Server

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L4-60

You can monitor your executed Spark applications by launching the SparkHistoryServer port directly, at :18080, as shown here. To monitor Spark applications in real-time, use port :4040.

You can also use the MapR Control System, known as the MCS. To launch the MCS, open a browser and navigate to the URL shown here: http://<ip address>:8443, and select SparkHistoryServer from the left navigation pane.

More information about monitoring jobs will be covered next, in lesson 5.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L4-61

Knowledge Check

Select the most appropriate command to launch your Scala application, "IncidentsApp" in a YARN cluster mode, where the application with its dependencies is packaged to "/path/to/file/incidentsapp.jar" :

A. `./bin/spark-submit --class IncidentsApp --master cluster /path/to/file/incidentsapp.jar`

B. `./bin/spark-submit --class IncidentsApp spark://100.10.60.120:7077 /path/to/file/incidentsapp.jar`

C. `./bin/spark-submit --class IncidentsApp --master yarn-cluster /path/to/file/incidentsapp.jar`

© 2018 MapR Technologies

Knowledge Check

Select the most appropriate command to launch your Scala application, "IncidentsApp" in a YARN cluster mode, where the application with its dependencies is packaged to "/path/to/file/incidentsapp.jar" :

A. `./bin/spark-submit --class IncidentsApp --master cluster /path/to/file/incidentsapp.jar`

B. `./bin/spark-submit --class IncidentsApp spark://100.10.60.120:7077 /path/to/file/incidentsapp.jar`

C. **`./bin/spark-submit --class IncidentsApp --master yarn-cluster /path/to/file/incidentsapp.jar`**

© 2018 MapR Technologies

Answer: C

A is missing 'yarn' before cluster, B is missing both 'yarn-cluster'

Lab
R2.2
© 2018 MapR Technologies
L4-64

Lab 4.4: Complete, Package, and Launch the Application

- Estimated time to complete: **20 minutes**
- In this lab, you still start with a partially completed project. The code is marked with `TODO` labels; locate them and follow the instructions to complete the code. Your code will load SFPD data into a Dataset, cache it, and print several values to the console.

  The solutions directory contains completed code.

© 2018 MapR Technologies

In this lab, you will run the standalone Spark application.

Congratulations! You have completed Lesson 4: Build a Simple Spark Application.

Welcome to Dev 361, Build and Monitor Apache Spark Applications, Lesson 5 - Monitor Apache Spark Applications.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-2

Learning Goals

5.1 Describe Logical and Physical Plans of Spark Execution

5.2 Use Spark Web UI to Monitor Spark Applications

5.3 Debug and Tune Spark Applications

© 2018 MapR Technologies

At the end of this lesson, you will be able to describe how Spark creates a logical plan and then converts it into a physical plan for job execution. You'll also learn how to use the Spark Web UI to monitor job progress, along with how to debug and tune Spark applications.

Learning Goals

5.1 Describe Logical and Physical Plans of Spark Execution

5.2 Use Spark Web UI to Monitor Spark Applications

5.3 Debug and Tune Spark Applications

© 2018 MapR Technologies

Let's start by looking at logical and physical plans in the Spark execution model.

Review
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-5

Review

RDDs are the basic building blocks of Spark

© 2018 MapR Technologies

Recall from lesson 2 that Resilient Distributed Datasets, or RDDs, are the basic building blocks of Spark. When we build Datasets in our applications, Spark uses RDDs under the hood for final computation.

Also recall from previous lessons that when a `GroupBy` transformation is executed on a Dataset, it groups the data according to the specified column, and returns a new Relational Grouped Dataset object.

Since `GroupBy` returns a new Dataset, some standard actions, such as `Count` for example, are now treated as transformations instead of actions.

Components of Spark Execution

Component	Description
Tasks	Unit of work within a stage corresponds to one RDD partition
Stages	Group of tasks which perform the same computation in parallel
Shuffle	Transfer of data between stages
Jobs	Work required to compute RDD; has one or more stages
Pipelining	Collapsing of RDDs into a single stage when RDD transformations can be computed without data movement
Directed Acyclic Graph (DAG)	Logical graph of Dataset operations
Resilient Distributed Dataset (RDD)	Parallel Dataset with partitions
Dataset/DataFrame	Distributed collection of data and primary abstraction in Spark. Data is organized as named columns similar to a table in relational database.

© 2018 MapR Technologies

Here are the components of the Spark Execution Model:

- A task is a unit of work within a stage corresponding to one RDD partition.
- A stage is a group of tasks which perform the same computation in parallel.
- A shuffle is the transferring of data between stages.
- A set of stages for a particular action is a job.
- When an RDD is computed from the parent without movement of data, the scheduler will pipeline or collapse RDDs into single stage
- DAG or Directed Acyclic Graph is the logical graph of Dataset operations.
- Resilient Distributed Datasets, or RDDs, are parallel Datasets with partitions.
- Datasets and DataFrames are distributed collections of data and are primary abstractions in Spark. Data is organized as named columns similar to a table in relational database.

Phases During Spark Execution

PHASE 1
Create the Logical Plan: User code defines the DAG

PHASE 2
Actions responsible for translating DAG into physical execution plan

PHASE 3
Tasks scheduled and executed on cluster

R2.2 © 2018 MapR Technologies

From a high level, the Spark execution model can be defined in three phases: creating the logical plan, translating that into a physical plan, and then executing the tasks on a cluster.

Phases During Spark Execution

PHASE 1
Create the Logical Plan: User code defines the DAG

PHASE 2
Actions responsible for translating DAG into physical execution plan

PHASE 3
Tasks scheduled and executed on cluster

© 2018 MapR Technologies

In the first phase, we create the logical plan. This is the plan that shows which steps will be executed when an action gets applied.

The user code defines Datasets and operations on Datasets.

Recall that when you apply a transformation on a Dataset, a new Dataset is created. When this happens, that new Dataset points back to the parent, resulting in a DAG.

Phases During Spark Execution

PHASE 1
Create the Logical Plan: User code defines the DAG

PHASE 2
Actions responsible for translating DAG into physical execution plan

PHASE 3
Tasks scheduled and executed on cluster

© 2018 MapR Technologies

Next, actions translate the DAG into a physical execution plan. The physical plan identifies resources, such as compute and memory, that will perform each step of the plan.

Phases During Spark Execution

PHASE 1
Create the Logical Plan: User code defines the DAG

PHASE 2
Actions responsible for translating DAG into physical execution plan

PHASE 3
Tasks scheduled and executed on cluster

R2.2 © 2018 MapR Technologies L5-12

Lastly, the tasks are scheduled and executed on the cluster.

These phases are executed in order and the action is considered complete when the final phase in a job completes. This sequence can occur many times when new Datasets are created.

Spark Execution Model: Logical Plan

1. ```
   val sfpdDF = spark.read.format("csv").option("inferSchema",
   true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category",
   "description", "dayofweek", "date", "time", "pddistrict",
   "resolution", "address", "X", "Y", "pdid")
   ```
2. ```
   val categoryDS = sfpdDF.groupBy(”category")
   ```
3. ```
   val categoryCountDS = categoryDS.count()
   ```
4. ```
   categoryCountDS.collect()
   ```


© 2018 MapR Technologies

We are going to see how a user program translates into the units of physical execution. Let us first take a look at the logical plan.

Consider the example from previous lessons, where we loaded SFPD data from a CSV file. We will continue to use this as our example to visually walk through the components of the Spark execution model.

Logical Plan Step 1: Import Data

1.
```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict",
"resolution", "address", "X", "Y", "pdid")
```

© 2018 MapR Technologies

The first line creates a DataFrame called `sfpdDF` from the sfpd.csv file. The data is loaded in tabular format into `sfpdDF`.

Remember from previous lessons that using `.toDF`, as we do here, defines names of column headers to make DataFrames easier to read, as opposed to having only generic column headers, like "_c0."

Logical Plan Step 2: Apply `groupBy` Transformation

2. `val categoryDS = sfpdDF.groupBy("category")`

© 2018 MapR Technologies

The second line applies a transformation where the data is grouped by the column: `category`. Recall that this results in a Relational Grouped Dataset, which is `categoryDS` in this example.

Logical Plan Step 3: Apply `count` Transformation

3. `val categoryCount = categoryDS.count()`

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-16

Remember that since the previous `GroupBy` transformation results in a Relational Grouped Dataset, the next operation of `Count, typically an action, is instead treated as a` transformation. This creates a new Dataset, `categoryCount`, from `categoryDS`, which counts the number of rows in each group that was returned in step 2.

No actions have been performed yet. Once Spark executes these lines, it defines a DAG of these Dataset objects. Each Dataset maintains a pointer to its parent or parents, along with the metadata about the type of relationship. Datasets use these pointers to trace their ancestors.

Logical Plan Step 4: Apply `collect` Action

4. `categoryCount.collect()`

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-17

Now we add a `Collect` action on the `categoryCount` Dataset. The `Collect` action triggers the actual computation.

The Spark scheduler then creates a physical plan to compute the Datasets needed for the computation. When the `Collect` action is called, every partition of the RDD in the Dataset is materialized and transferred to the driver program.

The Spark scheduler then works backward from `categoryCount` to create the physical plan necessary to compute all the ancestor Datasets that led up to this step.

Physical Plan

When an action is encountered, the DAG is translated into a physical plan.

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-18

When an action is encountered, the DAG is translated into a physical plan to compute the Datasets needed to perform the action. This results in computing the ancestor or ancestors for each Dataset.

The scheduler submits a job for each action to compute the required Datasets. This job has one or more stages, which are groups of tasks that perform the same computations in parallel on partitions.

Stages are processed in order and individual tasks are scheduled and executed on the cluster.

Physical Plan: Stages and Tasks

R2.2 © 2018 MapR Technologies L5-19

A stage has tasks for each partition in that Dataset's RDD. A stage launches tasks that do the same thing but on specific partitions of data. The number of stages in a physical plan generally matches the number of a Datasets underlying RDDs in the DAG.

In general, each task in a stage performs the same kinds of steps:

1. Fetch input from data storage or existing Dataset, or shuffle outputs.
2. Perform necessary transformations and/or actions in order to compute the required Datasets.
3. Write output to shuffle, external storage, or back to the driver.

Physical Plan: Stages and Tasks

© 2018 MapR Technologies

For example, let's say that Stage 1 is responsible for loading data and creating the base Dataset. Inside a stage, all tasks perform the same thing together, so we have 4 tasks that concurrently run through the 3 steps listed here. They first read the data input, then perform some Map and Filter operations, and lastly, write the output to shuffle.

Stage 2 performs some computation on the Dataset and writes the result either to a file or new Dataset, with the example steps shown here.

Keep in mind that these stages are determined automatically by Spark based on your specific Datasets and partitions.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-21

Knowledge Check

The number of stages in a job is usually equal to the number of Datasets underlying RDDs in the DAG. However, the scheduler can truncate the lineage when:

A. There is no movement of data from the parent Dataset

B. There is a shuffle

C. The Dataset is cached or persisted

D. The Dataset was materialized due to an earlier shuffle

© 2018 MapR Technologies

Knowledge Check

The number of stages in a job is usually equal to the number of Datasets underlying RDDs in the DAG. However, the scheduler can truncate the lineage when:

A. **There is no movement of data from the parent Dataset**

B. There is a shuffle

C. **The Dataset is cached or persisted**

D. **The Dataset was materialized due to an earlier shuffle**

© 2018 MapR Technologies

Answers: A, C, D

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-24

Learning Goals

5.1 Describe Logical and Physical Plans of Spark Execution

5.2 Use Spark Web UI to Monitor Spark Applications

5.3 Debug and Tune Spark Applications

© 2018 MapR Technologies

In this section, we will use the Spark Web UI to monitor Spark Applications.

Spark Web UI

http://<driver-node>:4040

© 2018 MapR Technologies

The Spark Web UI provides detailed information about the progress and performance of Spark jobs. By default, this information is only available for the duration of the application. You can view the web UI after the event by setting `spark.eventLog.enabled` to true before starting the application.

The Spark Web UI is available on the machine where the driver is running. You can access the web UI by going to the IP address of the driver node, by default the web UI uses port 4040. If multiple `SparkSessions` are running on the same host, they will bind to successive ports beginning with 4040, then going to 4041, 4042, and so on.

Note that in YARN cluster mode, the UI can be accessed through the YARN ResourceManager, which proxies requests directly to the driver.

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
```

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-27

The Jobs page gives you detailed execution information for active and recently completed Spark jobs. It gives you the performance of a job and also the progress of running jobs, stages, and tasks.

In this example, Job Id 0 is the first job that was executed and corresponds to the first line of code, `spark.read.load method`. It consists of a single stage with a single task. Job Id 1 corresponds to creating the DataFrame, `incidentCountDF`, which contains the count of incidents for each `incidentnum` in sfpdDF. Even this job has a single stage with a single task.

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
```

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-28

Job Id 2 corresponds to the first `Collect` action. It consists of two stages and 201 tasks. Time taken is 12 seconds.

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
incidentCountDF.collect
```

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-29

Job Id 3 corresponds to the second `Collect` action. It consists of a single stage with 200 tasks. The time taken by this job is only 2 seconds.

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
incidentCountDF.cache()
incidentCountDF.collect
incidentCountDF.collect
incidentCountDF.count
```

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-30

Job Id 4 corresponds to the `Count` action. It consists of two stages with 201 tasks each. The time taken by this job is also only 2 seconds.

Class Discussion
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-31

Class Discussion

The second `Collect` and `Count` jobs had skipped one stage. Why was one stage "skipped" and why is the duration lesser than job 2?

Spark Jobs (?)

User: mapr
Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:28	2017/10/22 10:52:05	2 s	2/2 (1 skipped)	201/201 (1 skipped)
3	collect at <console>:28	2017/10/22 10:51:41	2 s	1/1 (1 skipped)	200/200 (1 skipped)
2	collect at <console>:28	2017/10/22 10:45:18	12 s	2/2	201/201
1	load at <console>:23	2017/10/22 10:43:46	3 s	1/1	1/1
0	load at <console>:23	2017/10/22 10:43:45	0.5 s	1/1	1/1

© 2018 MapR Technologies

Answer:

First `Collect` computes all DataFramess and then caches `incidentCountDF` → has two stages
Second `Collect` and the `Count` use this cached DataFrame; Scheduler truncates lineage
Also note the difference in time for job 2 (12 s) and job 3 (2 ms).

Spark Web UI: Job Details

Details for Job 2

Status: SUCCEEDED
Completed Stages: 2

- Event Timeline
- DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	collect at <console>:28 +details	2017/10/22 10:45:24	6 s	200/200			2.0 MB	
2	cache at <console>:28 +details	2017/10/22 10:45:18	6 s	1/1	55.1 MB			2.0 MB

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-33

Clicking the link in the Description column on the Jobs page takes you to the Job Details page. This page also gives you the progress of the running job, stages, and tasks.

Spark Web UI: Job Details

Details for Job 3

Status: SUCCEEDED
Completed Stages: 1
Skipped Stages: 1

- Event Timeline
- DAG Visualization

Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	collect at <console>:28 +details	2017/10/22 10:51:41	2 s	200/200	1515.9 KB			

Skipped Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	cache at <console>:28 +details	Unknown	Unknown	0/1				

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-34

Here are the details for job 3, which has the skipped stage. On this page, you can see the details for both the completed stage and the skipped stage.

The `Collect` action here took only 2 seconds.

Spark Web UI: Stage Details for Job 1

Details for Stage 3 (Attempt 0)

Total Time Across All Tasks: 3 s
Locality Level Summary: Any: 200
Shuffle Read: 2.0 MB / 295185

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	5 ms	6 ms	8 ms	12 ms	0.6 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	9.3 KB / 1340	10.1 KB / 1449	10.3 KB / 1476	10.5 KB / 1502	11.0 KB / 1571

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records
driver	192.168.100.128:37279	6 s	200	0	0	200	2.0 MB / 295185

Tasks (200)

Page: 1 2 > 2 Pages. Jump to 1 . Show 100 items in a page. Go

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	3	0	SUCCESS	ANY	driver / localhost	2017/10/22 10:45:24	0.6 s		10.1 KB / 1449	
1	4	0	SUCCESS	ANY	driver / localhost	2017/10/22 10:45:24	0.1 s		10.1 KB / 1436	

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-35

Once you have found the stage in which you are interested, click the link to drill down to the Stage Details page. You can also navigate to this page using the main menu bar for Stages. Here we have summary metrics, aggregated metrics for completed tasks, and metrics on all tasks.

Spark Web UI: Storage Page

Spark 2.1.0-mapr-1707 | Jobs | Stages | Storage | Environment | Executors | SQL | Spark shell application UI

Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
*HashAggregate(keys=[incidentnum#25], functions=[count(1)], output=[incidentnum#25, count#64L]) +- Exchange hashpartitioning(incidentnum#25, 200) +- *HashAggregate(keys=[incidentnum#25], functions=[partial_count(1)], output=[incidentnum#25, count#69L]) +- *Project [_c0#0 AS incidentnum#25] +- *FileScan csv [_c0#0] Batched: false, Format: CSV, Location: InMemoryFileIndex[maprfs:///spark/lab5/sfpd.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:int>	Memory Deserialized 1x Replicated	200	100%	1515.9 KB	0.0 B

© 2018 MapR Technologies

The Storage page provides information about persisted Datasets. The Dataset is persisted if you called `Persist` or `Cache` on the Dataset followed by an action to compute on that Dataset.

This page tells you which fraction of the Dataset's underlying RDD is cached and the quantity of data cached in various storage media.

Scan this page to see if important Datasets are fitting into memory. You can also click on the RDD Name to view more details about the persisted Dataset.

Spark Web UI: Storage Page

RDD Storage Info for *HashAggregate(keys=[incidentnum#25], functions=[count(1)], output=[incidentnum#25, count#64L]) +- Exchange hashpartitioning(incidentnum#25, 200) +- *HashAggregate(keys=[incidentnum#25], functions=[partial_count(1)], output=[incidentnum#25, count#69L]) +- *Project [_c0#0 AS incidentnum#25] +- *FileScan csv [_c0#0] Batched: false, Format: CSV, Location: InMemoryFileIndex[maprfs:///spark/lab5/sfpd.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:int>

Storage Level: Memory Deserialized 1x Replicated
Cached Partitions: 200
Total Partitions: 200
Memory Size: 1515.9 KB
Disk Size: 0.0 B

Data Distribution on 1 Executors

Host	Memory Usage	Disk Usage
192.168.100.128:37279	1515.9 KB (364.8 MB Remaining)	0.0 B

200 Partitions

Page: 1 2 > 2 Pages. Jump to 1 . Show 100 items in a page. Go

Block Name	Storage Level	Size in Memory	Size on Disk	Executors
rdd_11_0	Memory Deserialized 1x Replicated	7.5 KB	0.0 B	192.168.100.128:37279
rdd_11_1	Memory Deserialized 1x Replicated	7.4 KB	0.0 B	192.168.100.128:37279

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-37

After clicking on the RDD name on the Storage page, you can view details about the Dataset's underlying RDD partitions as displayed here.

Spark Web UI: Environment Page

Spark 2.1.0-mapr-1707 | Jobs | Stages | Storage | Environment | Executors | SQL | **Spark shell** application UI

Environment

Runtime Information

Name	Value
Java Home	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79.x86_64/jre
Java Version	1.7.0_79 (Oracle Corporation)
Scala Version	version 2.11.8

Spark Properties

Name	Value
spark.app.id	local-1508691998953
spark.app.name	Spark shell
spark.driver.host	192.168.100.128
spark.driver.port	41576
spark.eventLog.dir	maprfs:///apps/spark
spark.eventLog.enabled	true

 © 2018 MapR Technologies

The Environment page lists all the active properties of your Spark application environment. Use this page when you want to see which configuration flags are enabled.

Only values specified through `spark-defaults.conf`, SparkSession, or the command line will be displayed here. For all other configuration properties, the default value is used.

Spark Web UI: Executors Page

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(1)	201	1.6 MB / 384.1 MB	0.0 B	1	0	0	604	604	19 s (2 s)	118.7 MB	0.0 B	2.1 MB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(1)	201	1.6 MB / 384.1 MB	0.0 B	1	0	0	604	604	19 s (2 s)	118.7 MB	0.0 B	2.1 MB

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	192.168.100.128:37279	Active	201	1.6 MB / 384.1 MB	0.0 B	1	0	0	604	604	19 s (2 s)	118.7 MB	0.0 B	2.1 MB	Thread Dump

The Executors page lists the active executors in the application. It also includes some metrics about the processing and storage on each executor.

Use this page to confirm that your application has the amount of resources you were expecting.

Spark Web UI: SQL Page

Spark 2.1.0-mapr-1707 | Jobs | Stages | Storage | Environment | Executors | SQL | **Spark shell** application UI

SQL

Completed Queries

ID	Description		Submitted	Duration	Jobs
2	count at <console>:28	+details	2017/10/22 10:52:05	2 s	4
1	collect at <console>:28	+details	2017/10/22 10:51:41	2 s	3
0	collect at <console>:28	+details	2017/10/22 10:45:18	12 s	2

Access Query Plan Details

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-40

In the SQL Page you can view the list of completed queries. Click the links under the Description column to view details and the logical plan of the executed queries.

Spark Web UI: SQL Page

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-41

Click the details link to view the detailed logical plan. This returns full detailed information in text format.

Spark Web UI via MCS

For MapR Distribution, Spark UI can be accessed via the MCS

If you have a MapR Distribution, you can access the Spark Web UI through the MapR Control System, or MCS. Once you log into the MCS, you will see the SparkHistoryServer in the left navigation pane as shown here.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-43

Knowledge Check

What are some of the things you can monitor in the Spark Web UI?

A. Which stages are running slow

B. Your application has the resources as expected

C. If the datasets are fitting into memory

D. All of the above

© 2018 MapR Technologies

Knowledge Check

What are some of the things you can monitor in the Spark Web UI?

A. Which stages are running slow

B. Your application has the resources as expected

C. If the datasets are fitting into memory

D. All of the above

© 2018 MapR Technologies

Answer: D

You can use the Job Details page and the Stages page to see which stages are running slow; to compare the metrics for a stage, look at each task.
Look at the Executors page to see if your application has the resources as expected.
Use the Storage page to see if the datasets are fitting into memory and what fraction is cached.

Lab
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-46

Labs 5.2a and 5.2b

- Estimated time to complete: **25 minutes**
- In these two short labs you will explore different Spark interfaces and resources:
  - In Lab 5.2a (15 minutes), you will use the Spark interactive shell to load and transform data. You will then view information in the Spark History Server page and the Spark Web UI.
  - In Lab 5.2b (10 minutes), you will perform a more in-depth exploration of the Spark Web UI.

© 2018 MapR Technologies

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-48

Learning Goals

5.1 Describe Logical and Physical Plans of Spark Execution

5.2 Use Spark Web UI to Monitor Spark Applications

5.3 Debug and Tune Spark Applications

© 2018 MapR Technologies

We are now going to explore some of the ways to debug and tune your Spark applications.

Common Slow Performance Issues

Level of Parallelism

Serialization Format

Memory Management

© 2018 MapR Technologies

Common things that can impact performance are:

- The level of parallelism
- The serialization format used during shuffle operations, and
- How well memory is optimized for your application

We will look at each one of these in more depth.

Review: Partitioning

Level of Parallelism

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-51

A Dataset's underlying RDD is divided into a set of partitions where each partition contains a subset of the data. The scheduler creates a task for each partition. Each task requires a single core in the cluster.

Common Performance Issues: Level of Parallelism

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-52

How does the level of parallelism affect performance? If there is too little parallelism Spark might leave resources idle.

Common Performance Issues: Level of Parallelism

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-53

If there is too much parallelism, overheads associated with each partition adds up, slowing down the processing time of your jobs.

Common Performance Issues: Level of Parallelism

© 2018 MapR Technologies

The number of partitions need to be controlled so you don't underutilize or oversubscribe your resources.

Tuning Level of Parallelism

Level of Parallelism

1 Check the number of partitions:

```
ds.rdd.partitions.size()
```

or

```
ds.rdd.getNumPartitions()
```


© 2018 MapR Technologies

Once you understand what you need to do to adjust your parallelism, you need to tune your system.

First, identify the number of partitions you have by using `ds.rdd.partitions.size` or `ds.rdd.getNumPartitions`. This provides you with the number of partitions in the Dataset's underlying RDD. You can also do this through the Spark Web UI in the Stages page. Since a task in a stage maps to a single partition in the RDD, the total number of tasks will give you the number of partitions.

Tuning Level of Parallelism

Level of Parallelism

2

Tune level of parallelism:

Change the number of partitions

```
coalesce()
repartition()
```


© 2018 MapR Technologies

Next, tune the level of parallelism by redistributing the data in the RDD.

This can be done by decreasing or increasing the number of partitions. You can use the `Coalesce` method to decrease the number of partitions or `Repartition` to increase the number of partitions.

Generally, if you have many idle tasks, say about ~10,000, then use `Coalesce`. If you are not using all the slots in the cluster, then use `Repartition`.

Common Performance Issues: Serialization Format

Serialization Format

- During data transfer, Spark serializes data
- Happens during shuffle operations
- Datasets use a specialized encoder for serialization

© 2018 MapR Technologies

When a large amount of data is transferred over the network during shuffle operations, Spark serializes objects into binary formats using a specialized encoder. This can sometimes cause a bottleneck.

Tuning Options: Serialization Format

Serialization Format

- Increase network bandwidth
- Decrease data movement between nodes

© 2018 MapR Technologies

There is not much that can be done specifically for this, but you can try to improve your overall network bandwidth and/or reduce the data movement between nodes. The MapR distribution does this automatically.

Common Performance Issues: Memory Management

Memory used in different ways in Spark

Memory Management

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L5-59

Memory can be used in different ways in Spark. Tuning Spark's memory use can help optimize your application.

By default, Spark will use:

- 60% of space for RDD storage
- 20% for shuffle, and
- 20% for user programs

Tuning Memory Usage

Memory Management

```
cache()
persist()
persist(MEMORY_ONLY)
persist(MEMORY_ONLY_SER)
```


© 2018 MapR Technologies

You can tune the memory usage by adjusting the memory regions used for RDD storage, shuffle, and user programs.

In most normal situations you will use `Cache` or `Persist`, which by default is the same as `Persist(MEMORY_ONLY)`. If there is not enough space to cache new RDD partitions, old ones are deleted and recomputed from the beginning when needed.

Using `MEMORY_ONLY_SER` will cut down on garbage collection. Caching serialized objects may be slower than caching raw objects. However, it does decrease the time spent on garbage collection.

Tuning Memory Usage

Memory Management

+

`persist(MEMORY_AND_DISK)`

© 2018 MapR Technologies

If you're seeing performance issues, you can also try using `Persist(MEMORY_AND_DISK)`. This will store the RDD to disk, which can be loaded into memory when needed. This reduces expensive recomputations of any lost RDD.

To learn more about other `Persist` options, check the Apache Spark documentation online.

Detect Performance Problems

Spark Web UI: Jobs, Stages, and Stage Details

Slow Tasks

Read/Write Issues

Skew

© 2018 MapR Technologies

You can use the Spark Web UI to detect and debug some common performance issues, like slow tasks, read and write issues, or skew.

Detect Performance Problems

Slow Tasks

- **Are there any tasks that are significantly slow?**
- **Are tasks running on certain nodes slow?**

Read/Write Issues

- Do some tasks read or write much more data than others?
- How much time tasks spend on each phase: read, compute, write?

Skew

- Is issue because of skew?

© 2018 MapR Technologies

Slow running tasks can be caused by overall shuffle problems. Look at the Spark Web UI to detect any tasks that are significantly slow. From the Stages Details page, you can also determine if tasks are running slow only on specific nodes.

Detect Performance Problems

Slow Tasks

- Are there any tasks that are significantly slow?
- Are tasks running on certain nodes slow?

Read/Write Issues

- **Do some tasks read or write much more data than others?**
- **How much time tasks spend on each phase: read, compute, write?**

Skew

- Is issue because of skew?

R2.2 © 2018 MapR Technologies

You can also find tasks that are spending too much time on reading, computing, and/or writing. In this case, check your code to see if you have any expensive operations that you can optimize. You should also cache your data as, and when, possible. This minimizes reads on-disk and helps improve overall performance.

Detect Performance Problems

Slow Tasks

- Are there any tasks that are significantly slow?
- Are tasks running on certain nodes slow?

Read/Write Issues

- Do some tasks read or write much more data than others?
- How much time tasks spend on each phase: read, compute, write?

Skew

- **Is issue because of skew?**

© 2018 MapR Technologies

A common source of performance problems in data-parallel systems is skew, which occurs when some small tasks take much longer than others.

To see if skew is the problem, look at the Stages Details page and see if there are some tasks running significantly slower than others. You should also drill down to see if there is a small number of tasks that read or write more data than others.

In these cases, check the distribution of your data and tasks across the cluster to be sure it is distributed evenly. You can also check the partitioning of your data and/or the scheduling of jobs on nodes.

Log Files

Spark logging subsystem based on log4j

Deployment Mode	Location
Spark Standalone	`work/` directory of distribution on each worker node
Apache Mesos	`work/` directory of Mesos slave node
Hadoop YARN	Use YARN log collection tool

© 2018 MapR Technologies

The Spark logging subsystem is based on log4j.

The logging or log output level can be customized. An example of the log4j configuration properties is provided in the Spark conf directory, which can be copied and edited.

The location of the Spark log files depends on your deployment mode.

Log Files

Deployment Mode	Location
Spark Standalone	`work/` directory of distribution on each worker node
Apache Mesos	`work/` directory of Mesos slave node
Hadoop YARN	Use YARN log collection tool

© 2018 MapR Technologies

In Spark Standalone mode, the log files are located in the work directory of the Spark distribution on each worker node.

Log Files

Deployment Mode	Location
Spark Standalone	`work/` directory of distribution on each worker node
Apache Mesos	`work/` directory of Mesos slave node
Hadoop YARN	Use YARN log collection tool

© 2018 MapR Technologies

In Apache Mesos, the log files are in the work directory of the Mesos slave node and are accessible from the Mesos master UI.

Log Files

Deployment Mode	Location
Spark Standalone	`work/` directory of distribution on each worker node
Apache Mesos	`work/` directory of Mesos slave node
Hadoop YARN	Use YARN log collection tool

© 2018 MapR Technologies

To access the logs in YARN, use the YARN log collection tool.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L5-70

Knowledge Check

Some ways to improve performance of your Spark application include:

A. Tune the degree of parallelism

B. Avoid shuffling large amounts of data

C. Use `ds.rdd.partitions.size()` or `ds.rdd.getNumPartitions()`

D. Check your log files

© 2018 MapR Technologies

Knowledge Check

Some ways to improve performance of your Spark application include:

A. **Tune the degree of parallelism**

B. **Avoid shuffling large amounts of data**

C. Use `ds.rdd.partitions.size()` or `ds.rdd.getNumPartitions()`

D. Check your log files

© 2018 MapR Technologies

Answer: A and B

Congratulations! You have completed Lesson 6 and this course DEV 361, Build and Monitor Apache Spark Applications. Proceed on to DEV 362 to learn more about Apache Spark Streaming and other advanced topics.

Welcome to DEV 362, Advanced Apache Spark, Lesson 6: Create an Apache Spark Streaming Application.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-2

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- Basic Steps and Save Data to Parquet Tables

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

When you have finished with this lesson, you will be able to describe the Apache Spark Streaming architecture. You will create a Spark streaming application, and apply operations on the application. You will also learn how windowed operations work, and how streaming applications are fault tolerant.

Learning Goals

6.1 **Describe Spark Streaming Architecture**

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- Basic Steps and Save Data to Parquet Tables

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

First, let's describe the overall Spark Streaming architecture.

Stream Processing Architecture: Streaming Data Source

© 2018 MapR Technologies

A stream processing architecture is typically made of the following components. First, the data we want to process must come from somewhere. This can be sensor networks, mobile applications, web clients, logs from a server, or even a “Thing” from the Internet of Things. Anything that produces data can be a source of streaming data.

Stream Processing Architecture: Ingest Data

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-6

The data generated is delivered through a messaging system such as the MapR Event Store for Apache Kafka, Flume, or it can be deposited directly into a file system like MapR XD Distributed File and Object Store.

The data is then processed by a stream processing system like Spark Streaming.

Processed data is stored in a NoSQL database, such as the MapR Database, or the data can be dumped directly into MapR XD as parquet tables or text files. This system must be capable of low latency, fast read and writes.

End applications like dashboards, business intelligence tools, and other applications can then use the processed data.

The output data can also be routed back into the processing system for further processing later.

Knowledge Check
R2.2
ACADEMYPRO
© 2018 MapR Technologies
L6-11

Knowledge Check

Indicate whether the following statements are TRUE or FALSE.

- Ingesting and processing streaming data involves a structured flow, with flexibility and options at each step.
- The output of processing must go a data visualization application.
- Streaming data is data that comes from a remote, cloud-based source.

© 2018 MapR Technologies

Knowledge Check

Indicate whether the following statements are TRUE or FALSE.

- Ingesting and processing streaming data involves a structured flow, with flexibility and options at each step. – **TRUE**
- The output of processing must go a data visualization application. - **FALSE**
- Streaming data is data that comes from a remote, cloud-based source. - **FALSE**

© 2018 MapR Technologies

- Ingesting and processing streaming data involves a structured flow, with flexibility and options at each step.
  TRUE – ingesting and processing streaming data follows a common process flow, but many different applications can be used at any step.

- The output of processing must go a data visualization application.
  FALSE – the output from processing can sent to a visualization application, but it can also be stored, or sent back into further processing.

- Streaming data is data that comes from a remote, cloud-based source.
  FALSE – Streaming data can come from a variety of sources. It can be remote, local, web-based, cloud-based, direct from a sensor, just about anything.

Let's now take a closer look at how streaming data is captured for processing.

As we just discussed, the streaming data can be from any number of sources like sensors and web applications, and this data is ingested by applications like the MapR Event Store for Apache Kafka.

While we think of the streaming data as a continuous wave, to be processed it still needs to be batched. Even processing each data point as it comes in, it is still thought of as a batch of one record.

Structured Streaming

Time

Transactions

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	

© 2018 MapR Technologies

Take, for example, your credit card company. They stream every transaction made into their system. Structured streaming treats a live data stream like this as a table that is continuously being appended.

Structured Streaming

Time

Transactions

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-16

As new transaction data comes in, each record is written as a row to the table. This table is unbounded, and will continue to grow, accepting new records as they are streamed in.

Understanding Structured Streaming

Current Data

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

Spark Query

© 2018 MapR Technologies

While the data is streaming in, at some point we want to process the data. When we do this, Spark will then query the Input Table. This will return the data currently in the table at the time of the query, as our first batch of data to process.

Understanding Structured Streaming

Current Data

id	first	last	amt	city	...
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

Spark Query

Process

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-18

Processing the data then returns a Result Table. In this case, we are processing the credit transactions to determine if the charge appears fraudulent. Spark will compare details of the record, such as amount and location, with past history for the same customer, and produce a score. Any score above a certain threshold will be added to the Result Table. Further action will then be taken on the information in the Result Table, to determine if the charge was, in fact, fraudulent.

Understanding Structured Streaming

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	
44207	Sara	Donner	355.37	Madrid	
30927	Maria	Domingo	556.22	El Paso	

© 2018 MapR Technologies

Meanwhile, as we are processing the first batch, more data continues to stream in. Each record continues to be appended as a new row to the Input Table.

Understanding Structured Streaming

Previous Data

New Data

id	first	last	amt	city	...
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	
55923	Sara	Donner	355.37	Madrid	
49920	Maria	Domingo	556.22	El Paso	
11036	Ed	Davis	12458.92	Longmont	

Incremental Query

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-20

Spark runs another query on the Input Table. This query will return only the new records in the table since the last query. This is called an incremental query.

Understanding Structured Streaming

1st Interval
1 sec

Previous Data

id	first	last	amt	city	...
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

2nd Interval
1 sec

New Data

id	first	last	amt	city	...
55923	Sara	Donner	355.37	Madrid	
49920	Maria	Domingo	556.22	El Paso	
11036	Ed	Davis	12458.92	Longmont	

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7
11036	87.3	amt	58.2

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-21

Thus, for each interval, for example one second, the new rows that were appended to the Input Table during that interval will be processed and update the Result Table.

Save Output to Storage

Save output as:

- Complete
- Append
- Update

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7
11036	87.3	amt	58.2

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-22

As new data is ingested and processed, and the Result Table is updated, we want to save the results to external storage. This saved data is referred to as the Output.

Output can be saved using one of three different modes: Complete, Append, and Update.

Output: Complete Mode

The entire Result Table is written to external storage each interval

(No fraud detected at Time Interval 1)

Time Interval 2

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	55.2

Time Interval 3

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

Result Table

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

new row

updated row

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-23

In Complete Mode, the entire updated Result Table is written to external storage.

Output: Append Mode

Only new rows in the result table are written each interval

(No fraud detected at Time Interval 1)

Time Interval 2

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	55.2

Time Interval 3

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

Result Table

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
00481	65.9	loc	21.7

new row

updated row

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-24

In Append Mode, only new rows appended in the Result Table since the last trigger interval are written to external storage. This mode is best applied where existing rows in the Result Table are not expected to change. Note in this example that the record for ID 16684 changes at time interval 3, but is not included in the output since it is not a new row.

Output: Update Mode

Updated rows in the result table are written each interval (new rows are considered updated)

(No fraud detected at Time Interval 1)

Time Interval 2

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Time Interval 3

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

new row

updated row

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	65.9	amt	55.2

id	fScore	flag	iScore
16684	70.1	loc	60.2
00481	65.9	loc	21.7

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-25

In Update Mode, rows that have been updated in the Result Table since the last trigger interval will be written to external storage. In this mode, new rows are considered updated rows, so new content is written along with any rows that have had their content changed. Therefore, if the query does not contain aggregations to existing rows, Update Mode is equivalent to Append Mode.

Streaming data is constantly being created by producers, and sent to the ingestion engine. Along this path, there can be a variety of problems that can cause data to be blocked or delayed. This can cause data to come into Spark out of its natural order. In many applications, we want to process the data in the correct order, or at least see all events that happened in a particular time frame. To help deal with this issue, we can use the Event-time.

Event-time is the time an event record occurs, and is embedded into the streaming data. Therefore, if a transaction was delayed, and comes into our credit card company out of order, we can use Event-time when processing, rather than the time the event was received, to make sure we are still including this record into our batch.

Handling Late Data

id	first	last	amt	city	...
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	

id	first	last	amt	city	...
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	526.40	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-27

Event-time allows Spark Streaming to handle data that is ingested out of order from when it was generated. As new data is ingested, and the Input Tables are updated, Spark can use the event-time to update older data and data that has arrived out of order. Then, using the Update Ouput Mode, the Results Table will be updated with the late data along with any aggregated data.

Streaming DataFrames and Streaming Datasets

```
val spark =
SparkSession.builder.appName("SensorData").getOrCreate()

val sensorCsvDF = spark.readStream ("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```


© 2018 MapR Technologies

You can use the Spark API to create `DataFrames` and `datasets` on streaming data, just as you can with static data. You can use the `readStream` method in the `SparkSession` interface to create and apply operations on streaming `DataFrames` and `datasets`, as with standard, static `DataFrames` and `datasets`. We'll look at this method in more detail later in this lesson.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-29

Knowledge Check

Which of these is not a mode of defining the output?

A. Append

B. Complete

C. Overwrite

D. Update

© 2018 MapR Technologies

Which of these is not a mode of defining the output?

Knowledge Check

Which of these is not a mode of defining the output?

A. Append

B. Complete

C. Overwrite

D. Update

© 2018 MapR Technologies

Which of these is not a mode of defining the output?

Answer: C – The modes of defining the output include Append, Complete, and Update.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-32

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 **Create a Spark Structured Streaming Application**
- **Define Use Case**
- Basic Steps and Save Data to Parquet Tables

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

In this section we will take a look at a real-world Spark Streaming use case.

Let's look at an application that monitors oil wells. Sensors in the oil rigs generate streaming data, which is processed by Spark, and stored in the MapR Database, for use by various analytical and reporting tools. Every event is processed by Spark streaming as it streams in, and stored in the MapR Database. The application will filter for and store any alarms that are generated, provide any other real-time data that we need, and daily Spark processing will store aggregated summary statistics.

First, data is streamed from the oil pump sensor to the MapR cluster.

Next, Spark Streaming receives and processes the data.

Then, processed data is written to a MapR Database table.

End applications read the MapR Database table data. Real-time monitoring filters for alarms, and any other immediate data that is needed, and calculates daily summary statistics. Summary statistics are written back to the MapR Database table.

Convert Line of CSV Data to Sensor Object

```
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

sensordata.csv

```
ResourceID,Date,Time,HZ,Displace,Flow,SedimentPPM,PressureLbs,ChlorinePPM
COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
```


© 2018 MapR Technologies

The oil pump sensor data comes in as comma separated value files, or CSVs, saved to a directory. Spark Streaming will monitor the directory and process any files created into that directory. Spark Streaming supports many different streaming data sources. For simplicity, in this example we will use the MapR Filesystem.

Unlike other Hadoop distributions that only allow cluster data import, or import as a batch operation, MapR lets you mount the cluster itself via NFS, so that your applications can read and write data directly. MapR allows direct file modification with multiple concurrent reads and writes via POSIX semantics. With an NFS-mounted cluster, you can read and write data directly with standard tools, applications, and scripts.

This means that MapR XD is really easy to use with Spark Streaming by putting data files into a directory.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-40

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- **Basic Steps and Save Data to Parquet Tables**

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

In this section we will discuss the basic steps needed to create a Spark Streaming application, and how to save processed data to Parquet tables.

Basic Steps for Spark Streaming Code

1. Initialize a Spark `StreamingContext` object
2. Using context, create a DStream
   - Represents streaming data from a source
3. Apply transformations and/or output operations to DStreams
4. Receive and process data
   - Use `streamingContext.start()`
5. Wait for the processing to be stopped
   - Use `streamingContext.awaitTermination()`

© 2018 MapR Technologies

Shown here are the basic steps for Spark Streaming code:

First, initialize a Spark `StreamingContext` object.

Using this context, create a Dstream, which represents streaming data from a source. Apply transformations and/or output operations to the Dstreams.

We can then start receiving data and processing it using the `streamingContext.start()` method.

And finally we wait for the processing to be stopped using the `streamingContext.awaitTermination()` method.

Let's walk through these steps using the oil pump use case described in the previous section as our model. Note that Spark Streaming programs are best run as standalone applications built using Maven or SBT. In the lab you will use the shell for simplicity, and build a streaming application.

Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-43

The first step is to create a SparkSession, which is the main entry point for all Spark operations.

Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```

```
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

R2.2 © 2018 MapR Technologies L6-44

Next, create a Spark Streaming `DataFrame`, using the oil pump schema defined earlier.

Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```

```
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

```
val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-45

Now, create a streaming `DataFrame` using the `spark.readstream` method. As new files are read from the directory, new rows will be added to the unbounded `DataFrame`.

Process Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```

```
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

```
val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```

```
// filter sensor data for low psi
sensorCsvDF.filter(_.psi < 5.0)
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-46

Then apply different transformations to the streaming `DataFrame`, just as you would on a static `DataFrame`. Spark handles the streaming data on the backend.

Wait for Termination

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```

```
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

```
val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```

```
// filter sensor data for low psi
sensorCsvDF.filter(_.psi < 5.0)
```

```
// Wait for the computation to terminate
ssc.awaitTermination()
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-47

Finally, add the `awaitTermination` method to instruct the Spark application to wait for the streaming computation to finish.

Streaming Application Output

```
-------------------------------------------
Time: 1452886488000 ms
-------------------------------------------
COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
COHUTTA,3/10/14,1:05,9.56,1.734,883,1.35,99,0.68
COHUTTA,3/10/14,1:06,9.74,1.736,884,1.27,92,0.73
COHUTTA,3/10/14,1:08,10.44,1.737,885,1.34,93,1.54
COHUTTA,3/10/14,1:09,9.83,1.738,885,0.06,76,1.44
COHUTTA,3/10/14,1:11,10.49,1.739,886,1.51,81,1.83
COHUTTA,3/10/14,1:12,9.79,1.739,886,1.74,82,1.91
COHUTTA,3/10/14,1:13,10.02,1.739,886,1.24,86,1.79
...

low pressure alert
Sensor(NANTAHALLA,3/13/14,2:05,0.0,0.0,0.0,1.73,0.0,1.51)
Sensor(NANTAHALLA,3/13/14,2:07,0.0,0.0,0.0,1.21,0.0,1.51)
-------------------------------------------
Time: 1452886490000 ms
-------------------------------------------
```


© 2018 MapR Technologies

The output from our application will show the name of the oil rig, and streamed data.

Save Data to Parquet

```
noAggDF.writeStream
.format("parquet")
.option("checkpointLocation", "/path/to/checkpoint/dir")
.option("path", "/path/to/destination/dir")
.start()
```


© 2018 MapR Technologies

The code shown here demonstrates how Spark Streaming data results can be written to the file system in parquet format. Results data can also be written to a database like MapR Database, however this is a more complex process, and is beyond the scope of this course. For further information, view MapR documentation online.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-50

Knowledge Check

Referring to the Spark Streaming code shown here:

```
val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("maprfs:///tmp/sensorData") // Equivalent to format("csv")
```

- What is the streaming DataFrame object?
- What method is used to create a streaming DataFrame?
- Where is the streaming data being read from?

© 2018 MapR Technologies

Knowledge Check

Referring to the Spark Streaming code shown here:

```
val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("maprfs:///tmp/sensorData") // Equivalent to format("csv")
```

- What is the streaming DataFrame object? **sensorCsvDF**
- What method is used to create a streaming DataFrame? **spark.readStream()**
- Where is the streaming data being read from? **maprfs:///tmp/sensorData**

© 2018 MapR Technologies

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-53

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- Basic Steps and Save Data to Parquet Tables

6.3 **Apply Operations on Streaming DataFrames**

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

In this section we will look at how to apply operations on Streaming DataFrames.

Operations on Streaming DataFrames

- What is the maximum, minimum, and average for sensor attributes?
- What is the pump vendor and maintenance information for sensors with low pressure alerts?

© 2018 MapR Technologies

Using the oil pump use case we just built, we would like to answer some questions such as, “what is the maximum, minimum, and average for sensor attributes?” and “what is the pump vendor and maintenance information for sensors with low pressure alerts?” To answer these questions, we will apply operations on the DStream we created in the previous lesson.

DataFrame and SQL Operations

```
sensorCsvDF.createOrReplaceTempView("sensor")
  val res = spark.sql( "SELECT resid, date,
   max(hz) as maxhz, min(hz) as minhz, avg(hz) as avghz,
   max(disp) as maxdisp, min(disp) as mindisp, avg(disp) as avgdisp,
   max(flo) as maxflo, min(flo) as minflo, avg(flo) as avgflo,
   max(psi) as maxpsi, min(psi) as minpsi, avg(psi) as avgpsi
   FROM sensor GROUP BY resid,date")

   res.show()
```

 © 2018 MapR Technologies

Our first question, “what is the max, min, and average for sensor attributes?,” is answered by using the query shown here.

Streaming Application Output

```
sensor max, min, averages
resid      date     maxhz minhz avghz              maxdisp mindisp avgdisp            maxflo minflo avgflo
LAGNAPPE   3/12/14 10.5  9.5   9.988799582463459  3.235   1.623   2.4714206680584563 1570.0 788.0  1199.1022964509395
CHER       3/13/14 10.5  9.5   10.005271398747391 3.552   1.857   2.732156576200417  1670.0 873.0  1284.660751565762
BBKING     3/13/14 10.5  9.5   9.996033402922755  1.58    0.902   1.26099164926931   1822.0 1041.0 1454.0240083507306
MOJO       3/12/14 10.5  9.5   10.006482254697294 3.589   2.143   2.888004175365341  1899.0 1134.0 1528.2724425887266
CARGO      3/10/14 10.5  9.5   9.998507306889353  3.752   1.903   2.8525417536534423 1533.0 778.0  1165.5302713987473
LAGNAPPE   3/11/14 10.5  9.5   10.009540709812102 3.092   1.454   2.307491649269313  1500.0 706.0  1119.5647181628392
CHER       3/12/14 10.5  9.5   10.008256784968692 3.443   1.728   2.6184937369519825 1619.0 812.0  1231.230688935282
BBKING     3/12/14 10.5  9.5   10.006795407098123 1.556   0.862   1.2138110647181624 1794.0 994.0  1399.608559498956
MOJO       3/11/14 10.5  9.5   10.0029331941545   3.513   1.979   2.8009843423799587 1859.0 1047.0 1482.2160751565762
LAGNAPPE   3/10/14 10.5  9.5   10.00329853862213  3.116   1.453   2.349840292275576  1512.0 705.0  1140.1022964509395
CHER       3/11/14 10.5  9.5   9.990396659707717  3.325   1.691   2.545035490605431  1564.0 795.0  1196.6837160751566
BBKING     3/11/14 10.5  9.5   9.997348643006282  1.49    0.821   1.1701722338204592 1718.0 947.0  1349.2849686847599
MOJO       3/10/14 10.5  9.5   9.999457202505194  3.345   1.828   2.6188089770354868 1770.0 967.0  1385.8131524008352
CHER       3/10/14 10.5  9.5   9.998726513569954  3.172   1.653   2.4233079331941516 1492.0 777.0  1139.4488517745303
BBKING     3/10/14 10.5  9.5   10.001409185803748 1.502   0.821   1.18567223382046   1732.0 947.0  1367.1711899791233
THERMALITO 3/14/14 10.5  9.5   9.983862212943635  3.784   2.135   3.0065960334029254 1680.0 948.0  1335.1002087682673
THERMALITO 3/13/14 10.5  9.5   9.999311064718169  3.896   2.116   3.069195198329852  1730.0 940.0  1362.910229645094
ANDOUILLE  3/14/14 10.5  9.5   10.011388308977041 2.146   1.139   1.6609373695198306 1592.0 845.0  1232.2296450939457
THERMALITO 3/12/14 10.5  9.5   10.007526096033398 3.823   1.986   2.9294561586638808 1697.0 882.0  1300.8622129436326
ANDOUILLE  3/13/14 10.5  9.5   9.99043841336118   2.174   1.104   1.656331941544886  1613.0 819.0  1228.8371607515658
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-57

Here is the output of the query for maximum, minimum, and average output readings from the oil pump sensors.

DataFrame and SQL Operations

```
// put pump vendor and maintenance data in temp table
spark.sparkContext.textFile("vendor.csv").map(parsePump.toDF().createOrRepl
aceTempView("pump")
spark.sparkContext("maint.csv").map(parseMaint).toDF().createOrReplaceTempV
iew("maint")
//
sensorCsvDF.filter(_.psi < 5.0). ().createOrReplaceTempView("alert")

val alertpumpmaint = sqlContext.sql("select s.resid, s.date, s.psi,
     p.pumpType, p.vendor, m.eventDate, m.technician from alert s join
     pump p on s.resid = p.resid join maint m on p.resid=m.resid")

alertpumpmaint.show()
```


© 2018 MapR Technologies

To answer our second question, “what is the pump vendor and maintenance information for sensors with low pressure alerts?,” we join the filtered alert data with pump vendor and maintenance information, which was read in and cached before streaming. The streaming `DataFrame` is registered as a temporary view, and then queried using SQL. The output of this join provides a list of sensor alerts, along with the vendor. This query is an example of complex data analysis.

Streaming Application Output

```
cmcdonald -
alert pump maintenance data
resid    date    psi pumpType  purchaseDate serviceDate vendor  eventDate technician description
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 10/2/09   T. LaBou   Install
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 10/5/09   W.Stevens  Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 11/24/09  W.Stevens  Tighten Mounts
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 6/10/10   T. LaBou   Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 1/7/11    T. LaBou   Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 9/30/11   W.Stevens  Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 10/3/11   T. LaBou   Bearing Seal
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 11/5/11   D.Pitre    Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 5/22/12   D.Pitre    Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 12/15/12  W.Stevens  Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 6/18/13   T. LaBou   Vane clearance ad...
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 7/11/13   W.Stevens  Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 2/5/14    D.Pitre    Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 3/14/14   D.Pitre    Shutdown Main Fee...
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 10/2/09   T. LaBou   Install
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 10/5/09   W.Stevens  Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 11/24/09  W.Stevens  Tighten Mounts
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 6/10/10   T. LaBou   Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 1/7/11    T. LaBou   Inspect
LAGNAPPE 3/14/14 0.0 HYDROPUMP 5/25/08      9/26/09     GENPUMP 9/30/11   W.Stevens  Inspect
-------------------------------------------
Time: 1452887522000 ms
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-59

The output of our second question query, shown here, displays sensor alerts, the vendor, and other related data.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-60

Knowledge Check

Indicate whether the statements below are TRUE or FALSE.

A. Transformations on a streaming DataFrame return another streaming DataFrame

B. Transformations trigger computations

C. Streaming DataFrames can only be made from streaming sources

© 2018 MapR Technologies

Indicate whether the statements below are TRUE or FALSE.

Knowledge Check

Indicate whether the statements below are TRUE or FALSE:

A. Transformations on a streaming DataFrame return another streaming DataFrame - **TRUE**

B. Transformations trigger computations - **FALSE**

C. Streaming DataFrames can only be made from streaming sources - **FALSE**

© 2018 MapR Technologies

Indicate whether the statements below are TRUE or FALSE.

A: TRUE

B: FALSE – Output actions trigger actions. A transformation returns another streaming DataFrame

C: FALSE – They can be made from file-based sources and network-based sources, such as socket-based sources.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-63

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- Basic Steps and Save Data to Parquet Tables

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

In this section we will look at how to define Spark Streaming windowed operations.

Sliding Windows

- Can compute results across longer time period
- Example – want oil pressure data across last 60 seconds 30 seconds
  - Create a sliding window of the last 60 seconds
  - Computed every 30 seconds

© 2018 MapR Technologies

Aggregations over a sliding event-time window are straightforward with Structured Streaming, and are similar to grouped aggregations. In a grouped aggregation, aggregate values, such as counts, are maintained for each unique value in the user-specified grouping column. In the case of window-based aggregations, aggregate values are maintained for each window the event-time of a row falls into. For example, we want to compute the oil pressure at a specific station every 30 seconds over the last 60 seconds. In this case we create a sliding window of the last 60 seconds, and compute every 30 seconds.

In this illustration, the original stream is coming in at one second intervals. The length of the sliding window is specified by the window length, in this case 3 units.

The window slides over the source `DStream` at the specified sliding interval, in this case 2 units.

Both the window length and the sliding interval must be multiples of the batch interval of the source `DStream`, which in this case is 1 second. When the window slides over the source `DStream`, all the events that fall in that window are combined. The operation is applied to the combined events resulting in batches in the windowed stream.

All window operations require two parameters:
The window length, is the duration of the window. In this example, the window length is 3 units.
And the sliding interval, which is the interval at which the operation window is performed. In this example, the sliding interval is 2 units.

Again, both of these parameters must be multiples of the batch interval of the original `DStream`.

Window-based Transformations

```
val windowedWordCounts = pairs.reduceByKeyAndWindow(
    (a:Int,b:Int) => (a + b), Seconds(12), Seconds(4))
```

sliding window operation

window length

sliding interval

© 2018 MapR Technologies

For example, we want to generate word counts every 4 seconds, over the last 6 seconds of data. To do this, we apply the `reduceByKey` operation on the `DStream` of paired 2-tuples, which consists of a word and the number 1, over the last 6 seconds of data using the operation `reduceByKeyAndWindow`.

Window Operations

Output Operation	Description
`window(windowLength,slideInterval)`	Returns new DStream computed based on windowed batches of source DStream.
`countByWindow(windowLength,slideInterval)`	Returns a sliding window count of elements in the stream.
`reduceByWindow(func,windowLength,slideInterval)`	Returns a new single-element stream created by aggregating elements over sliding interval using `func`.
`reduceByKeyAndWindow(func, windowLength,slideInterval,[numTasks])`	Returns a new DStream of (K,V) pairs from DStream of (K,V) pairs; aggregates using given reduce function `func` over batches of sliding window.
`countByValueAndWindow(windowLength,slideInterval,[numTasks])`	Returns new DStream of (K,V) pairs where value of each key is its frequency within a sliding window; it acts on DStreams of (K,V) pairs.

© 2018 MapR Technologies

Output operations are used to push the data in `DStreams` to an external system such as a database or file system. Some commonly used output operations are listed here.

Window Operations on DStreams

With a windowed stream:

- What is the count of sensor events by pump ID?
- What is the Max, Min, and Average for PSI?

© 2018 MapR Technologies

In our oil pump scenario, with a windowed stream of 10 minutes of data, updating every 5 minutes, we need to know the total count of sensor events by pump ID, and also what the maximum, minimum, and average pump PSI is. To answer these questions, we will use operations on a windowed stream.

Remember that each event from a device is a row in the Input Table. The event-time is a column value in each row.

This allows window-based aggregations, such as the number of events each minute, to be just a special type of grouping and aggregation on the event-time column. Each time window is a group, and each row can belong to multiple windows or groups. Therefore, such event-time-window-based aggregation queries can be defined consistently on both a static dataset, such as from collected device event logs, as well as on a data stream.

DataFrame and SQL Operations

```
//window of 10 minutes of data every 5 minutes.

val res = sensorCsvDF.groupBy(window($"timestamp", "10 minutes", "5
minute"), $"resid", $"date").agg(count("resid").alias("total"))

res.show
```


© 2018 MapR Technologies

The code shown here performs operations on 10 minutes worth of data, on a 5 minute window.

Streaming Application Output

```
cmcdonald — u
sensor count
resid       date     total
LAGNAPPE    3/12/14 958
CHER        3/13/14 958
BBKING      3/13/14 958
MOJO        3/12/14 958
CARGO       3/10/14 958
LAGNAPPE    3/11/14 958
CHER        3/12/14 958
BBKING      3/12/14 958
MOJO        3/11/14 958
LAGNAPPE    3/10/14 958
CHER        3/11/14 958
BBKING      3/11/14 958
MOJO        3/10/14 958
CHER        3/10/14 958
BBKING      3/10/14 958
THERMALITO 3/14/14 958
THERMALITO 3/13/14 958
ANDOUILLE  3/14/14 958
THERMALITO 3/12/14 958
ANDOUILLE  3/13/14 958
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-73

The total count of sensor events by pump ID is shown here.

DataFrame and SQL Operations

```
//window of 10 minutes of data every 5 minutes.
val res = sensorCsvDF.groupBy(window($"timestamp", "10 minutes", "5
minute"), $"resid", $"date").agg(max("psi").alias("maxpsi"),
min("psi").alias("minpsi"), avg("psi").alias("avgpsi"))

res.show
```


© 2018 MapR Technologies

We next need to gather the maximum, minimum, and average pump PSI. The code shown here uses the same data window, 10 minutes of data every 5 minutes, and collects the PSI data for each sensor.

Streaming Application Output

cmcdonald — user0

```
sensor max, min, averages
resid       date     maxpsi minpsi avgpsi
LAGNAPPE    3/12/14 100.0  75.0   87.30793319415449
CHER        3/13/14 100.0  75.0   88.16597077244259
BBKING      3/13/14 100.0  75.0   87.73903966597078
MOJO        3/12/14 100.0  75.0   87.55219206680584
CARGO       3/10/14 100.0  75.0   87.39352818371607
LAGNAPPE    3/11/14 100.0  75.0   87.37473903966597
CHER        3/12/14 100.0  75.0   87.13883089770354
BBKING      3/12/14 100.0  75.0   87.79749478079331
MOJO        3/11/14 100.0  75.0   87.74739039665971
LAGNAPPE    3/10/14 100.0  75.0   87.60647181628393
CHER        3/11/14 100.0  75.0   87.74008350730689
BBKING      3/11/14 100.0  75.0   87.71398747390397
MOJO        3/10/14 100.0  75.0   87.20876826722338
CHER        3/10/14 100.0  75.0   87.44885177453027
BBKING      3/10/14 100.0  75.0   87.20146137787056
THERMALITO  3/14/14 100.0  75.0   87.48225469728601
THERMALITO  3/13/14 100.0  75.0   87.45093945720251
ANDOUILLE   3/14/14 100.0  75.0   87.7223382045929
THERMALITO  3/12/14 100.0  75.0   87.39770354906054
ANDOUILLE   3/13/14 100.0  75.0   87.78496868475992
```

R2.2 MAPR ACADEMYPRO © 2018 MapR Technologies L6-75

The output from our application displays the requested PSI data for each pump.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2018 MapR Technologies
L6-76

Learning Goals

6.1 Describe Spark Streaming Architecture

6.2 Create a Spark Structured Streaming Application
- Define Use Case
- Basic Steps and Save Data to Parquet Tables

6.3 Apply Operations on Streaming DataFrames

6.4 Define Windowed Operations

6.5 Describe How Streaming Applications are Fault Tolerant

© 2018 MapR Technologies

At the end of this section, you will be able to describe how streaming applications are fault tolerant.

Write-Ahead Logs: Review

- All modifications are written to a log before being applied
- Redo and undo information is stored in the log

R2.2 © 2018 MapR Technologies

Let's review how write-ahead logs work. With write-ahead logs, all modifications are written to a log before being applied. Typically, both redo and undo information is stored in the log.

Fault Tolerance in Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors

Enable Write-Ahead Logs (WAL) + Receiver acknowledges receipt of data after data written to WAL = Zero data loss

© 2018 MapR Technologies

When write-ahead logs is enabled, all of the received data is saved to log files in a fault tolerant system. This makes the received data durable in the event of any failure in Spark Streaming.

Fault Tolerance in Spark Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors

R2.2 © 2018 MapR Technologies

Furthermore, if the receiver acknowledges the receipt of the data after it has been written to the WAL, then in the event of a failure, the data that is buffered but not saved can be resent by the source once the driver is restarted.

Fault Tolerance in Spark Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors

© 2018 MapR Technologies

These two factors provide zero data loss.

Refer to the Spark documentation on how to enable and configure write-ahead logs for Spark Streaming.

Checkpointing

- Provides fault tolerance for driver
- Periodically saves data to fault tolerant system
- Two types of checkpointing:
  - Metadata – for recovery from driver failures
  - Data checkpointing – for basic functioning if using stateful transformations
- Enable checkpointing
  - `ssc.checkpoint("hdfs://...")`

© 2018 MapR Technologies

For a streaming application to be available 24/7, and be resilient to failures, Spark Streaming needs to checkpoint enough information to a fault-tolerant storage system for recovery.

There are two types of checkpointing- metadata checkpointing and data checkpointing:
Metadata checkpointing is for recovery from driver failures.
Data checkpointing is needed for basic functioning if using stateful transformations.

You can enable checkpointing by calling the checkpoint method on the StreamingContext.

If you use sliding windows, you need to enable checkpointing.

Knowledge Check
R2.2
ACADEMYPRO
© 2018 MapR Technologies
L6-83

Knowledge Check

Spark Streaming uses several techniques to achieve fault tolerance on streaming data. Match the techniques listed below with the way in which they help ensure fault tolerance.

A. Data Replication

B. Write-Ahead Logs

C. Checkpointing

- ❑ Writes data to a fault tolerant system, and sends acknowledgement of receipt, to protect against receiver failure
- ❑ Save data to a fault tolerant system for recovery from a driver failure
- ❑ Saves data on multiple nodes for recovery should a single node fail

© 2018 MapR Technologies

Data Replication - Saves data on multiple nodes for recovery should a single node fail

Write-Ahead Logs - Writes data to a fault tolerant system, and sends acknowledgement of receipt, to protect against receiver failure

Checkpointing - Save data to a fault tolerant system for recovery from a driver failure

Knowledge Check

Spark Streaming uses several techniques to achieve fault tolerance on streaming data. Match the techniques listed below with the way in which they help ensure fault tolerance.

A. Data Replication

B. Write-Ahead Logs

C. Checkpointing

B. Writes data to a fault tolerant system, and sends acknowledgement of receipt, to protect against receiver failure

C. Save data to a fault tolerant system for recovery from a driver failure

A. Saves data on multiple nodes for recovery should a single node fail

© 2018 MapR Technologies

Data Replication - Saves data on multiple nodes for recovery should a single node fail

Write-Ahead Logs - Writes data to a fault tolerant system, and sends acknowledgement of receipt, to protect against receiver failure

Checkpointing - Save data to a fault tolerant system for recovery from a driver failure

In these activities, you will create a Spark Streaming application. There is also a follow up lab where you save Spark Streaming data to an HBase table.

Lab 6.5: Build a Streaming Application

- Estimated time to complete: **1 hour, 40 minutes**

- In this multi-part lab, you will:
  - Load data using the Spark shell
  - Use Spark Streaming to process a CSV file and display the contents
  - Build a Spark Streaming application using various methods

© 2018 MapR Technologies

Congratulations! You have completed Lesson 7.

Welcome to DEV 362: Advanced Apache Spark, Lesson 7, Use Apache Spark GraphFrames.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-2

Learning Goals

7.1 Describe GraphFrame

7.2 Define Regular, Directed, and Property Graphs

7.3 Create a Property Graph

7.4 Perform Operations on Graphs

© 2017 MapR Technologies

When you have finished this lesson, you will be able to describe GraphFrames, and define regular, directed, and property graphs. In addition, you will create a property graph, and perform operations on graphs.

Learning Goals

7.1 Describe GraphFrame

7.2 Define Regular, Directed, and Property Graphs

7.3 Create a Property Graph

7.4 Perform Operations on Graphs

© 2017 MapR Technologies

The first section introduces Apache Spark GraphFrames.

What is a Graph?

Represent a set of vertices that may be connected by edges

© 2017 MapR Technologies

A graph is a way of representing a set of vertices, that may be connected by edges.

What is a Graph?

Represent a set of vertices that may be connected by edges

R2.2 © 2017 MapR Technologies

A common example of graph processing is to find the shortest route between two points, where the points are the vertices and the route is edges.

What is GraphFrame?

- The Apache Spark component for graphs and graph-parallel computations.
- A distributed graph processing framework that sits on top of Spark core

Components	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrame	• Graph processing • Graph parallel computations
Spark Core	• Task scheduling • Memory management • Fault recovery • Interacting with storage system

© 2017 MapR Technologies

GraphFrame is the Apache Spark component for graphs and graph-parallel computations. It is a distributed graph processing framework that sits on top of Spark core.

Apache Spark GraphFrame

- Combines data parallel and graph parallel processing in single API
- View data as graphs and as collections (DataFrames)
  - No duplication or movement of data
- Operations for graph computation
- Provides graph algorithms and builders

Components	Function
Spark SQL	• Structure Data • Querying with SQL/HQL
Spark Streaming	• Processing of live streams • Micro-batching
MLlib	• Machine Learning • Multiple types of ML algorithms
GraphFrame	• Graph processing • Graph parallel computations
Spark Core	• Task scheduling • Memory management • Fault recovery • Interacting with storage system

© 2017 MapR Technologies

GraphFrame combines data parallel and graph parallel processing into a single API. We can view data as graphs, and as collections, without the need for duplication or movement of data. GraphFrame has a number of operators that can be used for graph computations, and also provides graph algorithms and builders for graph analytics.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-9

Knowledge Check

GraphFrame is the Apache Spark component for graphs and graph parallel computations. Which of the characteristics listed below are true of GraphFrame?

A. GraphFrame sits on top of Spark Core

B. GraphFrame provides distinct APIs for data parallel and graph parallel processing

C. GraphFrame provides multiple operators

D. When using GraphFrame, we can view data as graphs and as collections without the need for duplication or movement of data

© 2017 MapR Technologies

Knowledge Check

GraphFrame is the Apache Spark component for graphs and graph parallel computations. Which of the characteristics listed below are true of GraphFrame?

A. **GraphFrame sits on top of Spark Core**

B. GraphFrame provides distinct APIs for data parallel and graph parallel processing

C. **GraphFrame provides multiple operators**

D. **When using GraphFrame, we can view data as graphs and as collections without the need for duplication or movement of data**

© 2017 MapR Technologies

Answer: A, C, D

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-12

Learning Goals

7.1 Describe GraphFrame

7.2 Define Regular, Directed, and Property Graphs

7.3 Create a Property Graph

7.4 Perform Operations on Graphs

© 2017 MapR Technologies

In this section, we will define regular, directed, and property graphs.

Regular Graphs

Each vertex has the same number of edges

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L7-14

A regular graph is one where each vertex has the same number of edges.

Regular Graphs: Example

Edges run in both directions

© 2017 MapR Technologies

In a regular graph, edges run in both directions. For example, think of users on social media as vertices. If Bob is a friend of Sara, then Sara is also a friend of Bob.

Directed Graphs

Edges run in one direction

© 2017 MapR Technologies

A directed graph, or digraph, is one in which edges run in one direction, such as from vertex 5 to vertex 4.

Directed Graphs: Social Media Example

- User Fan follows user Team
- User Team does not follow user Fan

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L7-17

An example of a directed graph is a social media follower. User Fan can follow user Team without implying that user Team follows user Fan.

Property Graphs

Primary abstraction of Spark GraphFrame

- Immutable
- Distributed
- Fault-tolerant

© 2017 MapR Technologies

A property graph is the primary abstraction of Spark GraphFrame. Property graphs are immutable, distributed, and fault-tolerant.

Property Graphs

- Directed multi-graph
- Multiple edges in parallel
  - Unique, user-defined properties for every edge and vertex

R2.2 © 2017 MapR Technologies

A property graph is a directed multi-graph, which can have multiple edges in parallel. Every edge, and vertex, has unique, user-defined properties associated with it.

Property Graphs: Example

- Airport and flight map
  - Airports are vertices
  - Flights are separate, directed edges

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L7-20

For example, let's look at a property graph mapping airports and flights. Flights 123 and 1002 are separate, directed edges between airports on vertices LAX and SJC.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-21

Knowledge Check

The primary abstraction in Spark GraphFrame is the property graph. Which characteristics below are true of a property graph?

A. Property graphs are bidirectional

B. Edges and vertices have user-defined properties associated with them

C. Property graphs are directional

D. Every edge and vertex is unique

E. Property graphs are immutable

© 2017 MapR Technologies

Knowledge Check

The primary abstraction in Spark GraphFrame is the property graph. Which characteristics below are true of a property graph?

A. Property graphs are bidirectional

B. Edges and vertices have user-defined properties associated with them

C. Property graphs are directional

D. Every edge and vertex is unique

E. Property graphs are immutable

© 2017 MapR Technologies

Answer: B, C, D, E

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-24

Learning Goals

7.1 Describe GraphFrame

7.2 Define Regular, Directed, and Property Graphs

7.3 Create a Property Graph

7.4 Perform Operations on Graphs

© 2017 MapR Technologies

In this section, we will create a property graph.

To create a property graph, follow the steps listed here. We will describe each step in more detail next, using flight data.

Sample Flight Data

Airports (Vertices)

Vertex ID	Property (V)
id	name

Routes (Edges)

Source ID	Dest ID	Property (E)
source	dest	distance

© 2017 MapR Technologies

Let's walk through these steps using the sample flight data shown here, which assigns vertices to represent airports, and edges to represent routes.

Step 1: Import Required Classes

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

To create a property graph, import the GraphFrame related classes first. When importing GraphFrame classes in the Spark shell, use the command shown here.

Step 2: Create Vertex Data

```
//First create data for vertices (Airports)
val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))
```

id	name (Property)
1	SFO
2	ORD
3	DFW

© 2017 MapR Technologies

Next, we create data values for the vertices, which represent airports. Each vertex includes the “id” and “name” of the airport, and the name of the airport is the property.

Step 2: Create Vertex Data

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

The code to add the data values for the vertices, which represents airports, is highlighted here.

Step 3: Create Vertex DataFrame

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

We then create a DataFrame for the vertices by calling the spark.createDataFrame method, and passing in the vertices list data. Column names for this DataFrame are defined as “id” and “name”.

Step 4: Create Edge Data

```
//Create data for edges (Routes)
val edges = List((1,2,1800),(2,3,800),(3,1,1400))
```

source	dest	distance (Property)
1	2	1800
2	3	800
3	1	1400

R2.2 © 2017 MapR Technologies

Next, we need to create data values for edges, just as we did with the verticies. Edges use the values for the source airport, the destination airport, and the distance between them. The distance is the property of edges.

Step 4: Create Edge Data

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

The code to create data values for the edges is highlighted here.

Step 5: Create Edge DataFrame

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

Next, we create a separate DataFrame using the edge list data, again by calling the spark.createDataFrame method. The column names for this DataFrame are defined as `source`, `dest` – for destination, and `distance`. The edge DataFrame must contain `source` and `dest` as the first and second columns. `Source` specifies the ID of the source vertex for the edge, and `dest` specifies the ID of the destination vertex. Additional columns can be added as edge properties.

Step 6: Create Property Graph

```
//import org.graphframes._
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages
graphframes:graphframes:0.5.0-spark2.1-s_2.11

val vertices = List((1, "SFO"),(2, "ORD"),(3,"DFW"))

val verticesDF = spark.createDataFrame(vertices).toDF("id","name")

val edges = List((1,2,1800),(2,3,800),(3,1,1400))

val edgesDF =
spark.createDataFrame(edges).toDF("source","dest","distance")

val graph = GraphFrame(verticesDF, edgesDF)
```


© 2017 MapR Technologies

Once the verticesDF and edgesDF DataFrames have been defined, we create the property graph by using the GraphFrame method, passing in verticesDF and edgesDF as parameters.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-36

Knowledge Check

The steps to create a Property graph are listed below. List these steps in the correct order.

- ❑ Create Vertex DataFrame
- ❑ Create Graph
- ❑ Import Required Classes
- ❑ Create Edge DataFrame

© 2017 MapR Technologies

Knowledge Check

The steps to create a Property graph are listed below. List these steps in the correct order.

1) **Import Required Classes**
2) **Create Vertex DataFrame**
3) **Create Edge DataFrame**
4) **Create Graph**

© 2017 MapR Technologies

1. Import required classes
2. Create vertex DataFrame
3. Create edge DataFrame
4. Create Graph

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-39

Learning Goals

7.1 Describe GraphFrame

7.2 Define Regular, Directed, and Property Graphs

7.3 Create a Property Graph

7.4 Perform Operations on Graphs

© 2017 MapR Technologies

In this section, we will look at the different types of operators on property graphs.

Graph Operators: Collection and Structural

Operator	Description
`vertices`	Returns a DataFrame containing the vertices and associated attributes.
`edges`	Returns a DataFrame containing the edges and associated attributes.
`find(pattern)`	Searches the graph for structural patterns (Motif finding) and returns a DataFrame which consists of all instances of Motif.
`triplets`	Returns a DataFrame with 3 columns: `source`, `edge` and `dest`. Schema for columns `source` and `dest` match `GraphFrame.vertices` and the schema for the column `edge` matches `GraphFrame.edges`.

© 2017 MapR Technologies

Here you can see some of the operators that we can use to find more information about the graph, such as checking vertices and edge data in a graph.

Graph Operators: Example

```
//Check vertices and edges of Graph
graph.vertices.show
graph.edges.show
```

```
scala > graph.vertices.show
+---+----+
| id|name|
+---+----+
|  1| SFO|
|  2| ORD|
|  3| DFW|
+---+----+
```

```
scala > graph.edges.show
+------+----|--------+
|source|dest|distance|
+------+----|--------+
|     1|   2|    1800|
|     2|   3|     800|
|     3|   1|    1400|
+------+----+--------+
```


© 2017 MapR Technologies

We use the GraphFrame vertices method to access and display data about verticesDF, and the edges method to access and display data about edgesDF.

Graph Operators: Graph Information

Operator	Description
`vertexColumns`	Names of columns in vertices DataFrame.
`edgeColumns`	Names of columns in edges DataFrame.
`inDegrees`	The in-degree of each vertex.
`outDegrees`	The out-degree of each vertex.
`Degrees`	The degree of each vertex.

© 2017 MapR Technologies

Here are some operators, and descriptions, to find more information about the graph.

Graph Operators: Vertex Degrees

```
//Check degrees of vertices of Graph
graph.degrees.show

//Check in degree of vertices
graph.inDegrees.show
```

```
scala > graph.degrees.show
+---+------+
| id|degree|
+---+------+
|  1|     2|
|  2|     2|
|  3|     2|
+---+------+
```

```
scala > graph.inDegrees.show
+---+--------+
| id|inDegree|
+---+--------+
|  1|       1|
|  3|       1|
|  2|       1|
+---+--------+
```

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L7-44

In the case of a directed graph, you can specify indegrees and outdegrees.

Graph Operators: Caching Graphs

Operator	Description
`cache()`	Caches the vertices and edges; default level is `MEMORY_ONLY`.
`persist(newLevel)`	Caches the vertices and edges at specified storage level; returns a reference to this graph.
`unpersist()`	Uncaches both vertices and edges of this graph from memory and disk.
`unpersist(blocking)`	Similar to `unpersist()` but accepts a boolean value whether to block until all blocks are removed from memory and disk.

© 2017 MapR Technologies

The operators shown here can be used for caching graphs.

Graph Operators: Standard Graph Algorithms

Operator	Description
`bfs`	Breadth-First-Search (BFS) algorithm for traversing a graph.
`pageRank`	Algorithm that measures the importance of vertices in a graph.
`shortestPaths`	Finds the shortest path between vertices.
`connectedComponents`	Finds the connected components of a graph.
`stronglyConnectedComponents`	Finds strongly connected components of a directed graph.

© 2017 MapR Technologies

Various standard graph algorithms are shown here.

Class Discussion
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L7-47

Class Discussion

1. How many airports are there?
   - In our graph, what represents airports?
   - Which operator could you use to find the number of airports?

2. How many routes are there?
   - In our graph, what represents routes?
   - Which operator could you use to find the number of routes?

© 2017 MapR Technologies

How Many Airports are There?

How many airports are there?

- In our graph, what represents airports?

  Vertices

- Which operator could you use to find the number of airports?

  `graph.vertices.count`

© 2017 MapR Technologies

Vertices represent airports. Use the `graph.vertices.count` operator to calculate the number of airports.

How Many Routes are There?

How many routes are there?

- In our graph, what represents routes?

  Edges

- Which operator could you use to find the number of routes?

  `graph.edges.count`

© 2017 MapR Technologies

Edges represent routes. Use the `graph.edges.count` operator to calculate the number of routes.

Use Cases

- Monitor air traffic at airports
- Monitor delays
- Analyze airport and routes overall
- Analyze airport and routes by airline

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L7-51

Graphs can be used to represent various air traffic use cases.

To monitor data in real time, graph processing can be added to a streaming application, with the output sent to a visualization tool.

If you want to do a longer term analysis, you can save the data over a period such as a month or quarter, and then do the analysis by airline, by route, by week, or whatever variable is needed.

In this activity, you will define a graph and apply graph operators to it. In the first part of the activity, you will create the property graph and apply operators to it in the Interactive shell. In the second part, we analyze real flight data with GraphFrames.

Lab 7.4: Analyze Data with GraphFrame

- Estimated time to complete: **40 minutes**
- In this lab, you will use GraphFrame to analyze flight data and retrieve various flight statistics.

© 2017 MapR Technologies

Congratulations! You have completed Lesson 7: Use Apache Spark GraphFrames. Continue on to lesson 8 to learn about the Apache Spark Machine Learning Library.

Welcome to DEV 362, Advanced Apache Spark, Lesson 8: Use Apache Spark MLlib, the machine learning library.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-2

Learning Goals

8.1 Describe Apache Spark MLlib Machine Learning Algorithms

8.2 Use Collaborative Filtering to Predict User Choice

R2.2 © 2017 MapR Technologies L8-3

At the end of this lesson, you will be able to define the Apache Spark machine learning library, describe three popular machine learning techniques, and use collaborative filtering to predict what a user will like.

Learning Goals

8.1 **Describe Apache Spark MLlib Machine Learning Algorithms**

8.2 Use Collaborative Filtering to Predict User Choice

R2.2 © 2017 MapR Technologies L8-4

Let's begin by learning about the algorithms in the Apache Spark machine learning library.

What is MLlib?

- Only includes machine learning algorithms designed to run on clusters
- Most suitable for running on a single large data set

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-5

MLlib is the Apache Spark library of machine learning functions. MLlib includes machine learning algorithms designed to run on clusters. Non-parallel processing algorithms are not included. Apache Spark machine learning library algorithms are most suitable for running on a single, large data set.

Examples of ML Algorithms

Supervised

- Classification
- Regression Analysis
- Collaborative Filtering

Unsupervised

- Clustering
- Dimensionality Reduction

© 2017 MapR Technologies

In general, machine learning may be broken down into two classes of algorithms: supervised and unsupervised.

Examples of ML Algorithms: Supervised

Supervised

- Classification
- Regression Analysis
- Collaborative Filtering

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-7

Supervised algorithms use data that include data label descriptions. These labels define what is in the data, such as what images may appear in a picture, what topics are in written text, or even a description about the data, such as is this email is spam, or not spam?

These labels are used to help train the algorithm to perform more efficiently, as it starts with a correct definition of the data.

Examples of ML Algorithms: Unsupervised

Unknown Data

Unsupervised

- Clustering
- Dimensionality Reduction

© 2017 MapR Technologies

Unsupervised algorithms, on the other hand, do not use predefined labeled data. These algorithms are left to make sense of unknown data on their own without any help.

Let's take a look at these different algorithms more closely.

Classification: Definition

Form of ML that:

- Identifies which category an item belongs to
- Uses supervised learning algorithms
  - Data is labeled

© 2017 MapR Technologies

Starting with the Supervised algorithms, classification is a family of machine learning algorithms that designate input as belonging to one of several pre-defined classes. Some common use cases for classification include credit card fraud detection and email spam detection, both of which are binary classification problems. Classification data is labeled, for example, as fraud/non-fraud or spam/non-spam. Machine learning assigns a label, or class, to new data based on similar examples of known data.

Classification: Example

 © 2017 MapR Technologies

Gmail uses the classification machine learning technique to determine whether an email is spam or not, based on the email data: the sender, recipients, subject, and message body.

If it Walks/Swims/Quacks Like a Duck… Then It Must Be a Duck

Features of ML:

- Classify something based on pre-determined features
- Features are the “if questions” that you ask

Walks

Swims

Quacks

© 2017 MapR Technologies

You can classify something based on pre-determined features. Features are the “if questions” that you ask. The label is the answer to those questions**.** In this example, if it walks, swims, and quacks like a duck, then the label is "duck". In this simplistic example, the classification is binary, but classification can extend to any number of pre-defined classes.

Building and Deploying a Classifier Model

Spam:
free money now!
Super sale!
free savings $$$

Non-spam:
how are you?
that Spark job
Team meeting

Training Data

Image reference O'Reilly Learning Spark

© 2017 MapR Technologies

To build a classifier model, first extract the features that most contribute to the classification. In our email example, we find features that define an email as spam, or not spam.

Building and Deploying a Classifier Model

Image reference O'Reilly Learning Spark

© 2017 MapR Technologies

The features are transformed and put into Feature Vectors, which is a vector of numbers representing the value for each feature. We rank our features by how strongly they define an email as spam, or not spam.

Building and Deploying a Classifier Model

Image reference O'Reilly Learning Spark

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-14

We train our model by making associations between the input features and the labeled output associated with those features.

Building and Deploying a Classifier Model

Image reference O'Reilly Learning Spark

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-15

Then at runtime, we deploy our model by extracting the same set of features, and ask the model to classify that set of features as one of the pre-defined set of labeled classes.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-16

Knowledge Check

Put the steps for creating and training a Classification model into the correct order:

Define the classification features
Create feature vectors by ranking how strongly each feature classifies the data
Associate the features with the label to train the model
Extract the set of features from new data
Classify the new data with one of the labels

© 2017 MapR Technologies

Knowledge Check

Put the steps for creating and training a Classification model into the correct order:

1. Define the classification features
2. Extract the set of features from new data
3. Create feature vectors by ranking how strongly each feature classifies the data
4. Associate the features with the label to train the model
5. Classify the new data with one of the labels

© 2017 MapR Technologies

Regression Analysis

© 2017 MapR Technologies

The next supervised algorithm we'll look at is regression analysis. Regression analysis determines the direction and strength of relationships between different data variables, helping to define trends, and predict potential results.

Regression Analysis

R2.2 © 2017 MapR Technologies L8-20

A retail business like the Big Office Supply Company uses regression analysis to determine how profit is affected by different factors of their business, such as marketing spending, warehouse size and location, or promotional pricing.

Regression Analysis

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-21

There are a number of regression analysis techniques, and the most commonly used one is linear regression. Linear regression defines a line, the slope of which is used to determine the relationship between our data points.

Regression Analysis

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-22

Linear regression analysis is susceptible to outliers, which can affect the slope of the line, making the correlation between the two variables seem stronger or weaker.

Regression Analysis

© 2017 MapR Technologies

Linear regression is also affected by multicollinearity, which is when more than one variable directly affects the variable we are looking at. For example, both marketing spending and promotional pricing can directly affect company profits.

Multicollinearity can be a problem with big data, as we often gather huge data sets from a variety of sources, many of which can be important to what we want to know. Think about medical data that measures several aspects of a patient's health, for example, and a number of them can have a direct impact on the patient, such as their blood pressure.

With regression analysis, we can see the overall trend of our data, but with a mixed data set it can be difficult to get a good measure of how each variable individually affects the outcome we are looking for.

Machine Learning: Collaborative Filtering

- Based on user preferences data
  - Collaborative
- Recommend items
  - Filtering

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-24

Collaborative filtering algorithms recommend items based on preference information from many users. The collaborative filtering approach is based on similarity; people who liked similar items in the past will like similar items in the future.

In the example shown, Ted likes movies A, B, and C. Carol likes movies B and C. Bob likes movie B. To recommend a movie to Bob, we calculate that users who liked B also liked C, so C is a possible recommendation for Bob.

Machine Learning: Collaborative Filtering

Collaborative Filtering (Recommendation)

© 2017 MapR Technologies

For example, Amazon uses this machine learning technique of collaborative filtering, also commonly referred to as recommendation, to determine products users will like based on their history and similarity to other users.

Clustering

- Classifies inputs into categories by analyzing similarities between input examples

Marco Toledo Bastos, and Gabriela Zago SAGE
Copyright © by a Creative Commons Attribution License, unless otherwise noted.

Clustering is an unsupervised algorithm that classifies unknown data into categories. It analyzes the data set and looks for similarities between input examples.

Clustering uses unsupervised algorithms, which do not have the outputs in advance. No known classes are used as a reference, as with a supervised algorithm like classification.

Clustering: Definition

- Unsupervised learning task
- Groups objects into **clusters of high similarity**
  - **Search results** grouping
  - **Grouping of customers**
  - **Anomaly** detection
  - Text categorization

© 2017 MapR Technologies

Clustering can be used for many purposes, for example, grouping search results.

Other examples include:
- partitioning data into groups, such as grouping similar customers
- anomaly detection, such as fraud detection
- and text categorization, such as sorting books into genres

Machine Learning: Clustering

 © 2017 MapR Technologies

Google News uses a technique called clustering to group news articles into different categories, based on title and content.

Clustering algorithms discover groupings that occur in collections of data.

Clustering: Example

- Group similar objects

© 2017 MapR Technologies

In this example, we are given a set of raw data points. The objective is to create some number of clusters that group these data points with those that are most similar, or closest.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-30

Clustering using the K-means algorithm begins by initializing all the coordinates to centroids. There are various ways you can initialize the points, including randomly.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid

© 2017 MapR Technologies

With each pass of the algorithm, every point is assigned to its nearest centroid based on some distance metric, usually Euclidean distance.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid
3. Update centroids to center of points

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-32

The centroids are then updated to be the “centers” of all the points assigned to it in that pass.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid
3. Update centroids to center of points
4. Repeat until conditions met

© 2017 MapR Technologies

The algorithm stops according to various programmable conditions, such as:

- A minimum change of median from the last iteration
- A sufficiently small intra-cluster distance
- A sufficiently large inter-cluster distance

Dimensionality Reduction

- Reduce dimensions in large dataset
- Process only vital data
- More accurate and efficient

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-34

Dimensionality reduction is a group of machine learning algorithms that are used to reduce a very large set of data, so that the refined data can be used with other algorithms, like regression or classification, more accurately and efficiently.

Dimensionality reduction is often used to pre-process an extremely large or complex data set, to refine it to the key data needed for final processing.

Dimensionality reduction techniques can be divided into two groups: Feature Selection and Feature Extraction.

Dimensionality Reduction: Feature Selection

© 2017 MapR Technologies

Remember from earlier in this lesson, we talked about how regression analysis cannot accurately differentiate multiple variables, also known as dimensions or features, of your data that affect the aspect you are looking at, such as both promotional pricing and marketing spending affecting overall profit. Having more than one data dimension affecting the target variable is called multicollinearity.

Dimensionality Reduction: Feature Selection

© 2017 MapR Technologies

Feature selection techniques are used to pull out just the features that you want to process, ignoring unwanted dimensions in your refined dataset. When processing just the reduced set of desired features, other machine learning algorithms, like regression analysis and classification, are much more efficient and accurate.

Dimensionality Reduction: Feature Selection

ID	Longitude	Latitude	Elevation	Oil Pressure	Wind Speed	Fuel
180931080809	121.9552W	37.3541N	152.64	31.0	4.2ESE	16.2
180931080810	121.9563W	37.3765N	152.33	31.1	4.2ESE	16.2
180931080811	121.9575W	38.4561N	151.22	31.0	4.1ESE	16.2
180931080812	122.0152W	38.9562N	150.01	31.0	4.1ESE	16.2

© 2017 MapR Technologies

Feature selection is very useful for removing redundant features, as seen in the profit analysis example. It can also be used to remove known, irrelevant features to reduce the amount of processing needed to analyze the desired features. For example, say you want to analyze the location data from sensors on a self driving car. You can use feature selection to isolate that data, and not spend resources processing unwanted features like oil pressure, wind speed, and fuel levels.

Dimensionality Reduction: Feature Extraction

© 2017 MapR Technologies

Feature extraction is used to reduce the number of dimensions by combining similar features into a single feature that still represents the data accurately.

Feature extraction is used to reduce the amount of resources needed to process a data set.

Dimensionality Reduction: Feature Extraction

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-39

In our profit margin analysis problem, we can use feature extraction to combine multiple features that affect the profit margin together, to make a single feature that is much easier to process and visualize.

Dimensionality Reduction: Feature Extraction

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-40

Feature extraction is often used with image processing and optical character recognition. Feature extraction algorithms detect desired shapes in the image or video stream, and then define them as the features to be processed.

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-41

Knowledge Check

Match the scenarios listed here with the machine learning technique that would provide the best results: Classification, Clustering, or Collaborative Filtering

A. You want to determine what restaurant someone may like, based on restaurants they have previously liked.

B. You want to detect fraudulent attempts to log into your website.

C. You need to list which students have passed, and which have failed an exam.

D. You want to organize your music collection based on genre metadata.

© 2017 MapR Technologies

Knowledge Check

Match the scenarios listed here with the machine learning technique that would provide the best results: Classification, Clustering, or Collaborative Filtering

A. You want to determine what restaurant someone may like, based on restaurants they have previously liked. **Collaborative Filtering**

B. You want to detect fraudulent attempts to log into your website. **Clustering**

C. You need to list which students have passed, and which have failed an exam. **Classification**

D. You want to organize your music collection based on genre metadata. **Clustering**

© 2017 MapR Technologies

Answers:
A. Restaurant recommendations::Collaborative Filtering. Suggesting restaurants that people liked, who also like the restaurants that our user liked.
B. Login fraud detection::Clustering. Looking for anomalies in failed login attempts.
C. List passed and failed students::Classification
D. Organizing music::Clustering.

Class Discussion
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-44

Class Discussion

- What is the difference between supervised and unsupervised machine learning algorithms?
- Why is classification considered supervised, while clustering is considered unsupervised?

© 2017 MapR Technologies

Supervised algorithms require known information about the results, to which to compare the sample data.

Unsupervised algorithms have no known information, and must determine all of their results from the sample data.

Classification has a known set of definitions that the sample data is compared against. The sample data is determined to either be the same as, or different from the known definitions.

Clustering organizes a set of data into groups, based only on the data itself. It does not require any known prior information about the data.

Learning Goals
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-46

Learning Goals

8.1 Describe Apache Spark MLlib Machine Learning Algorithms

8.2 Use Collaborative Filtering to Predict User Choice

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-47

Now that you are familiar with the machine learning algorithms, let's look at how we can use collaborative filtering to predict what a user will like.

Train a Model to Make Predictions

Ted and Carol like movies B and C

Training Data → Algorithm → Model

Bob likes movie B, what might he like?

New Data → Model → Predictions

Bob likes movie B, **predict C**

User Item Rating Matrix

	A	B	C
Ted	4	5	5
Carol		5	5
Bob		5	?

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-48

Let's take a closer look at our movie recommendation example. To quickly review, the goal of a collaborative filtering algorithm is to take preferences data from users, and to create a model that can be used for recommendations or predictions.

Ted likes movies A, B, and C. Carol likes movies B and C. We take this data and run it through an algorithm to build a model. As new data is received, for example Bob likes movie B, we use the model to predict that C is a possible recommendation for Bob.

Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

- User and Item factor matrices

Item

User

User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

?

T	1.4	.9
C	1.2	1
B	1.5	.9
A	1.2	.8

User

*

A	B	C	D
1.4	1.3	.9	1.2
.8	11	2	.8

Item

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-49

Alternating Least Squares, or ALS, approximates a sparse user item rating matrix of dimension K, as the product of two dense matrices. We start with User and Item factor matrices of size U×K and I×K. The factor matrices are also called latent feature models, and represent hidden features which the algorithm tries to discover.

Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

- User and Item factor matrices
- Tries to learn the hidden features of each user and item

Item

User

User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

?

T	1.4	.9
C	1.2	1
B	1.5	.9
A	1.2	.8

User

*

A	B	C	D
1.4	1.3	.9	1.2
.8	11	2	.8

Item

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-50

One matrix tries to describe the latent or hidden features of each user…

Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

- User and Item factor matrices
- Tries to learn the hidden features of each user and item

Item

User

User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

?

T	1.4	.9
C	1.2	1
B	1.5	.9
A	1.2	.8

User

*

A	B	C	D
1.4	1.3	.9	1.2
.8	11	2	.8

Item

© 2017 MapR Technologies

…and another tries to describe latent properties of each item.

Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

- User and Item factor matrices
- Tries to learn the hidden features of each user and item
- Algorithm alternatively fixes one factor matrix and solves for the other

Item

User

User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

?

T	1.4	.9
C	1.2	1
B	1.5	.9
A	1.2	.8

User

*

A	B	C	D
1.4	1.3	.9	1.2
.8	11	2	.8

Item

© 2017 MapR Technologies

ALS is an iterative algorithm. During each iteration, the algorithm alternatively fixes one factor matrix and solves for the other. This process continues until it converges. The practice of alternating between which matrix to optimize is why the process is called Alternating Least Squares.

A typical machine learning workflow is shown here. To make our predictions, we will perform the following steps:

Load the sample data, and parse the data into the input format for the ALS algorithm.

Split the data into two parts, one for building the model and one for testing the model.

We then run the ALS algorithm to build and train a user product matrix model. We make predictions with the training data, and observe the results. Then, test the model with the test data.

Parse Input Lines into Ratings Objects

```
// Import ml recommendation data types
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS

//Define case class
case class Rating(userId: Int, movieId: Int, rating: Float)

// parse string: UserID::MovieID::Rating
def parseRating(str: String): Rating = {
val fields = str.split("::")
assert(fields.size == 4)
Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat)
}
```


© 2017 MapR Technologies

To build a model, first, we import the required ML libraries. Next, we define a Rating case class with columns, userId, movieId, and rating. The parseRating function parses a line from the ratings data file into the Rating class. We will use this as input for the ALS prediction. In this case we are clearly defining the schema for the data, so we will use a dataset going forward.

Parse Input

```
// load ratings data into a Dataset
val ratingsDS =
spark.read.textFile("/spark/lab8/ratings.dat")
.map(parseRating)
```


© 2017 MapR Technologies

We then load ratings data into the ratingsDS dataset. We call the parseRating function to each element in the data, which returns a new dataset. We are defining this new dataeet as ratingsDS.

Build Model

Split `ratingsDS` into:

- Training data DS (80%)
- Test data DS (20%)

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-59

Next we split the data into two parts, one for building the model and one for testing the model.

Build Model

```
// Randomly split ratings DS into training data DS (80%) and
test data DS (20%)
val splits = ratingsDS.randomSplit(Array(0.8, 0.2), 0L)

val trainingRatingsDS = splits(0).cache()
val testRatingsDS = splits(1).cache()
```


© 2017 MapR Technologies

In the code to split the data, we randomly apply a 80-20 split to the data in ratingsDS. 80% of split data is stored in the trainingRatingsDS dataset, and 20% of split data is stored in testRatingsDS dataset. Both datasets are cached.

Build Model

Build a user product matrix model

© 2017 MapR Technologies

We then run the ALS algorithm to build and train a user product matrix model.

Build Model

```
// build a ALS user product matrix model with rank=20,
iterations=10
val model = new ALS()
.setMaxIter(10).setRank(20).setRegParam(0.01)
.setUserCol("userId").setItemCol("movieId")
.setRatingCol("rating").fit(trainingRatingsDS)
```


© 2017 MapR Technologies

Use the code shown here to build the ALS user product matrix model, with rank=20 and max iterations=10.

Get Predictions

© 2017 MapR Technologies

Now that we have trained our model, we want to get predicted movie ratings for the test data.

Get Predictions

```
// call model.transform with test Userid, MovieId input data
val predictionsDF = model.transform(testRatingsDS)
```


© 2017 MapR Technologies

To get predicted ratings for each test User ID and Movie ID pair, we call the `model.transform` method, with the `testRatingsDS`. Because the input is unknown, this algorithm always returns a DataFrame.

Compare Predictions to Tests

Test Ratings

(user, movie, test rating)

Predicted Ratings

(user, movie, predicted rating)

Joined Ratings for Comparison

(user, movie, test rating, predicted rating)

© 2017 MapR Technologies

Next, we compare test ratings to the predicted ratings. We register datasets `testRatingsDS and predictionsDF as views, and then compare the ratings using a query.`

Compare Predictions to Tests

```
// Register Dataset testRatingsDS as testratings view
testRatingsDS.createTempView("testratings")

// Register DataFrame predictionsDF as predictions view
predictionsDF.createTempView("predictions")

// Join tables and compare the results using query
val compareDS = spark.sql("select t.userid, t.movieid, t.rating,
p.prediction from testratings t, predictions p where t.userid =
p.userid and t.movieid = p.movieid")
```


© 2017 MapR Technologies

Using the commands shown here, register the dataset `testRatingsDS` as the `testratings` view, and `predictionsDF` as the `predictions` view. We can then retrieve the results by executing the compare query.

Compare Predictions to Tests

```
// Check results of compareDS
compareDS.show(5)
```

```
scala> compareDS.show(5)
+------+-------+------+----------+
|userid|movieid|rating|prediction|
+------+-------+------+----------+
|    53|    148|   5.0|  4.388935|
|  1605|    148|   2.0| 2.3322415|
|  2507|    148|   4.0|  3.632678|
|  3650|    463|   2.0|  2.505344|
|  5306|    463|   2.0|  3.174282|
+------+-------+------+----------+
only showing top 5 rows
```

R2.2 MAPR ACADEMYPRO © 2017 MapR Technologies L8-67

We can compare the ratings and predictions by viewing the `compareDS` data, using the `show` method.

Compare Predictions to Tests

Find false positives where:

test rating <= 1 and predicted rating >= 4

compareDS

```
(user, movie, test rating, predicted rating)
```

```
// Register Dataset compareDS as compare view
compareDS.createTempView("compare")
```


© 2017 MapR Technologies

To identify false positives, compare ratings predictions to the actual rating, and look for a highly predicted rating when the actual rating was low. We register the compareDS Dataset as the view “compare”, which we can then query with a tool like Drill.

Test Model

```
spark.sql("select userid, movieid, rating, prediction from
compare where rating<=1 and prediction>=4").show(5)
```

```
+------+-------+------+----------+
|userid|movieid|rating|prediction|
+------+-------+------+----------+
|  1645|   1088|   1.0| 5.8135347|
|  1808|   1088|   1.0|   4.23835|
|  4011|   1342|   1.0| 4.4337964|
|  4342|   1959|   1.0|  4.641789|
|  1017|   2659|   1.0| 4.3406224|
+------+-------+------+----------+
only showing top 5 rows
```


© 2017 MapR Technologies

The code shown here compares actual ratings to predicted ratings by filtering on ratings where the test rating is less than 1 (<=1), and the predicted rating is greater than 4 (>=4).

Knowledge Check
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-70

Knowledge Check

Place the steps below in the order to correctly describe a supervised learning workflow.

- ❑ Use the validated model with new data
- ❑ Build, test and tune the model
- ❑ Load sample data
- ❑ Split the data into training and data sets
- ❑ Extract features

© 2017 MapR Technologies

1. Load sample data
2. Extract features
3. Split the data into training and test sets
4. Build, test, and tune the model
5. Use the validated model with new data

Knowledge Check

Place the steps below in the order to correctly describe a supervised learning workflow.

1) **Load sample data**
2) **Extract features**
3) **Split the data into training and data sets**
4) **Build, test and tune the model**
5) **Use the validated model with new data**

© 2017 MapR Technologies

1. Load sample data
2. Extract features
3. Split the data into training and test sets
4. Build, test, and tune the model
5. Use the validated model with new data

Lab
R2.2
MAPR ACADEMYPRO
© 2017 MapR Technologies
L8-73

Lab 8.2 – Use Apache Spark MLlib

- Estimated time to complete: **60 minutes**
- In this lab, you will use the Spark shell to load and inspect data, make movie recommendations, and analyze a simple flight example using decision trees.

© 2017 MapR Technologies

Congratulations! You have completed DEV 362 Advanced Apache Spark.

Visit the MapR Academy to find more courses about big data processing and analysis.