

# Apache Spark™ and Scala

Certification Examination Preparation Guide



**Global Leader in Professional Certification Training**

[www.simplilearn.com](http://www.simplilearn.com)



[support@simplilearn.com](mailto:support@simplilearn.com)



Live Chat 24/7 on [simplilearn.com](http://simplilearn.com)

## Before you start!

Your Certification is important to us. Please take a moment to read this page and use all of Simplilearn's tools and resources available for thorough preparation.



### 90-Day Online Training Access

With your classroom enrollment, you are entitled to 90 days of free online training access.

- Free 90 Days E-learning access with high quality audio-video chapters
- Start/Pause/Resume your learning
- Multiple Simulation Exams & Quizzes for self-evaluation
- Downloadable eBook for anytime, anywhere reference

### Accessing Simplilearn's Learning Management System

To access our LMS, please visit [www.simplilearn.com](http://www.simplilearn.com), navigate to the Login tab and enter your credentials, For LMS login credentials please write to [support@simplilearn.com](mailto:support@simplilearn.com)



### Obtaining Your PDU Certificate

As a PMI® accredited Authorized Training Organization, Simplilearn provides Professional Development Unit (PDU) certificates for all professionals completing our certification training.

Softcopy of the PDU certificate will be issued via registered email address within seven working days after completion of training. For any PDU certificate related queries, please write to [support@simplilearn.com](mailto:support@simplilearn.com).

### Other Resources

Simplilearn's official blog, The Turning Point and Knowledge Bank section has lots of useful certification-oriented articles and links to free webinars to aid your preparation for 100+ professional certification courses available with Simplilearn.



### Support

Simplilearn's expert Support Team is available 24/7 via Email, Live Chat and Phone to answer all your queries. For any clarification, please visit [www.simplilearn.com](http://www.simplilearn.com) and use any of these channels to get in touch and we'll be more than happy to help.

Over 100,000 professionals have used our course, including mock tests and other resources for their certification exam preparation and we hope you'll find the experience just as satisfying and rewarding.

**Happy Learning!**

## Apache Spark™ and Scala Certification Training



Authored & published by

Simplilearn

Version 1.0

### Notice

This document contains proprietary information, which is protected by copyright. All rights reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior consent of Simplilearn.

## What's in store

### In Classroom Learning



In-depth training by highly experienced and certified trainers



Prepare and practice through Apache SparkTM and Scala Simulation Exam

### Post-Classroom Assistance



90 days access to the online course



15 hours of high quality interactive e-learning content with demos



28 end of lesson quizzes



Real world examples from various industries



Intensive industry specific project



Access to Apache SparkTM and Scala Simulation Exam Papers



1 Downloadable eBook

## How to register Online

1

Go to [www.simplilearn.com](http://www.simplilearn.com). If you have previously registered, please log in using the "Login" option at the top-right area on the screen. Please enter your registered email ID & password to access online or self-study material at any time.

2

If you haven't registered already, please register using the 'Register' button at the bottom of the "Login screen".

3

When you register for the first time, you will need to enter basic profile details. This information will be saved for future use and can be edited. Once you have entered all the necessary information, click "Signup" at the bottom of the page.

4

Once registered, you can edit your profile, access self-study material, take the exam, see your exam results & register for a new course by entering your login information using the "Login" option at the top-right area of the screen.

Thank you for registering. Have a great class and all the best for your exam!

## Icons Used in This Book

The following icons are used across the book to highlight specific points.



### eLearning

Refers to the eLearning course available on the Simplilearn.com site for a detailed explanation of the concepts. This icon will be present on each page of the handbook indicating the need to study the eLearning course content along with the classroom content.



### Real-Life Examples

Highlights examples relevant to specific content. This icon will be present when a concept is explained with the help of a real life example.



### Quiz

Refers to the questions at the end of every lesson. This icon will indicate the start of Quiz questions.



### Tips

Indicates some important advice or point mentioned on the page that one should remember.



### Notes

Use this section to add any key takeaways from the class for your future reference. This icon will be present on each page of the handbook.

## IT Service Management—Best Practices

IT Service Management is the implementation and management of quality IT services that meet the needs of a business. It includes best practices followed by many organisations. The sources of best practices are:

- Existing public standards published by the International Standards Organisation or ISO;
- Industry practices that are shared among industry practitioners;
- Academic research; and
- Internal experiences or an organisation's past experiences in providing similar services.



## Table of Contents

<u>LESSON 0—COURSE OVERVIEW .....</u>	<u>2</u>
<u>LESSON 1—INTRODUCTION TO SPARK.....</u>	<u>2</u>
<u>LESSON 2—INTRODUCTION TO PROGRAMMING IN SCALA.....</u>	<u>2</u>
<u>LESSON 3—USING RDD FOR CREATING APPLICATIONS IN SPARK .....</u>	<u>2</u>
<u>LESSON 4—INTRODUCTION TO CASSANDRA .....</u>	<u>2</u>
<u>LESSON 5—SPARK STREAMING .....</u>	<u>2</u>
<u>LESSON 6—SPARK ML PROGRAMMING .....</u>	<u>2</u>
<u>LESSON 7—SPARK GRAPHX PROGRAMMING.....</u>	<u>2</u>

## Lesson 0—Course Overview





## Before you start!

Your Certification is important to us. Please take a moment to read this page and use all of Simplilearn's tools and resources available for thorough preparation.



### 90-Day Online Training Access

With your classroom enrolment, you are entitled to 90 days of free online training access.

- Free 90 Days E-learning access with high quality audio-video chapters
  - Start/Pause/ Resume your learning
  - Multiple Simulation Exams & Quizzes for self-evaluation
  - Downloadable eBook for anytime, anywhere reference



## Obtaining Your PDU Certificate

As a PMI® accredited Authorized Training Organization, Simplilearn provides PDU certificates for all professionals completing our certification training. Softcopy of the PDU certificate will be issued via registered email address within seven working days after completion of training. For any PDU certificate related queries, please write to [support@simplilearn.com](mailto:support@simplilearn.com).



Accessing Simplilearn's Learning Management System

To access our LMS, please visit [www.simplilearn.com](http://www.simplilearn.com), navigate to the Login tab and enter your credentials. For LMS login credentials please write to [support@simplilearn.com](mailto:support@simplilearn.com)



Support

Simplilearn's expert Support Team is available 24/7 via Email, Live Chat and Phone to answer all your queries. For any clarification, please visit [www.simplilearn.com](http://www.simplilearn.com) and use any of these channels to get in touch and we'll be more than happy to help.



## Other Resources

Simplilearn's official blog, The Turning Point and Knowledge Bank section has lots of useful certification-oriented articles and links to free webinars to aid your preparation for 100+ professional certification courses available with Simplilearn.

Over 100,000 professionals have used this book, included mock tests and other resources for their certification exam preparation and we hope you'll find the experience just as satisfying and rewarding.

Happy Learning!

© Copyright 2015, Simplilearn. All rights reserved.

2

## Key Features

Before you start, please spend some time and go through the key features of Simplilearn offerings displayed here.





## Course Objectives

After completing this course, you will be able to:



- Explain the process to install Spark
- Describe the features of Scala
- Discuss how to use RDD for creating applications in Spark
- Explain how to run SQL queries using SparkSQL
- Discuss the features of Spark Streaming
- Explain the features of Spark ML Programming
- Describe the features of GraphX Programming

© Copyright 2015, Simplilearn. All rights reserved.

3

## Course Objectives

After completing this course, you will be able to:

- Explain the process to install Spark
- Describe the features of Scala
- Discuss how to use RDD for creating applications in Spark
- Explain how to run SQL queries using SparkSQL
- Discuss the features of Spark Streaming
- Explain the features of Spark ML Programming, and
- Describe the features of GraphX Programming

## Course Overview



The Apache Spark and Scala training course offered by Simplilearn provides an overview of the following:

- Fundamentals of real-time analytics and need of distributed computing platform
- Scala and its features
- Architecture of Apache Spark
- Installation and running applications using Apache Spark
- Performing SQL, streaming, and batch processing
- Machine Learning and Graph analytics on the Hadoop data

## Course Overview

The Apache Spark and Scala training course offered by Simplilearn provides details on the fundamentals of real-time analytics and need of distributed computing platform. It will also explain Scala and its features. Further, it will enhance your knowledge on the architecture of Apache Spark. The course will also explain the process of installation and running applications using Apache Spark. Further, it will enhance your knowledge on performing SQL, streaming, and batch processing. Finally, it will explain Machine Learning and Graph analytics on the Hadoop data.

## Target Audience



The target audience for this course are:

- Professionals aspiring for a career in growing and demanding fields of real-time big data analytics
- Analytics professionals, research professionals, IT developers, testers, data analysts, data scientists, BI and reporting professionals, and project managers
- Other aspirants and students, who wish to gain a thorough understanding of Apache Spark

## Target Audience

The course is aimed at professionals aspiring for a career in growing and demanding fields of real-time big data analytics. Analytics professionals, research professionals, IT developers, testers, data analysts, data scientists, BI and reporting professionals, and project managers are the key beneficiaries of this course. Other aspirants and students, who wish to gain a thorough understanding of Apache Spark can also benefit from this course.

## Course Prerequisites



The prerequisites for the Apache Spark and Scala course are:

- Fundamental knowledge of any programming language
- Basic understanding of any database, SQL, and query language for databases
- Working knowledge of Linux- or Unix-based systems (not mandatory)

## Course Prerequisites

Fundamental knowledge of any programming language is a prerequisite for the course. Participants are expected to have basic understanding of any database, SQL, and query language for databases. Working knowledge of Linux or Unix based systems is an added advantage for this course. Although, it is not mandatory.

## Value to the Professionals

Consider the case study below:



© Copyright 2015, Simplilearn. All rights reserved.

7

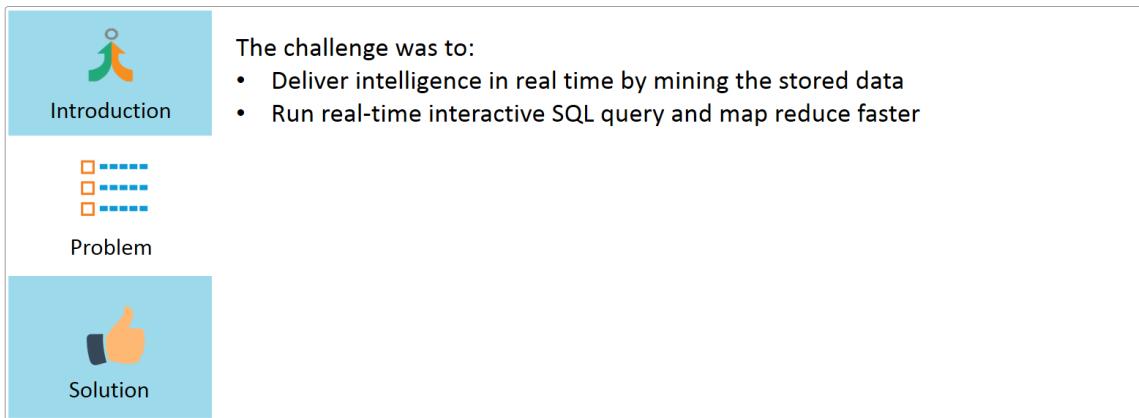
## Value to Professionals

Let's understand the value of Apache Spark and Scala to the professionals with the help of this case study of Ooyala, which is a video services company that is based in San Francisco. It offers services to various media organizations including ESPN, Bloomberg, Yahoo, Japan, and Telegraph Media Group. It offers “actionable analytics” that allows them analysing the manner in which their video content is being consumed in great detail. It is also enabling them optimizing their delivery standards for maximizing their revenues. The analytics engine processes of the organization achieve more than 2 billion analytics events every day. These analytics are obtained from about 200 million viewers across the world watching videos on an Ooyala-powered player.

## Value to the Professionals (contd.)



Consider the case study below:



© Copyright 2015, Simplilearn. All rights reserved.

8

## Value to Professionals (contd.)

Now, the challenge for Ooyala was to deliver intelligence in real time by mining the stored data. This data included personalized content recommendations to clients and viewing patterns. In addition, the organization needed to run real-time interactive SQL query and map reduce faste.

## Value to the Professionals (contd.)

Consider the case study below:

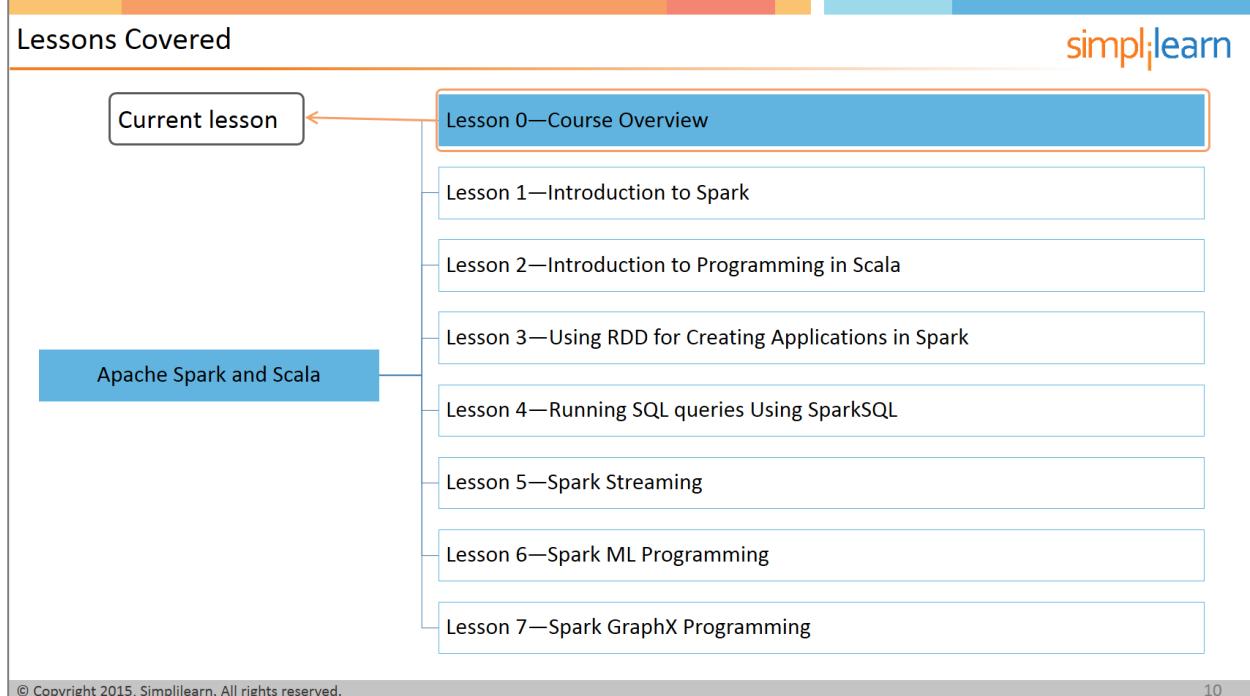


This issue has been resolved by using Spark. Ooyala can now:

- Perform real-time analytics using it
- Expand rapidly and offer real-time intelligence based on huge data sets

### Value to Professionals (contd.)

This issue has been resolved by using Spark. Ooyala can now perform real-time analytics using it. It has made Ooyala a leading organization of big data technology. The organization now has the power to expand rapidly and offer real-time intelligence that are based on huge data sets.



© Copyright 2015, Simplilearn. All rights reserved.

10

## Lessons Covered

There are 7 core lessons in this course, apart from the current lesson, Course Overview. Take a look at the course map displayed on the screen.

## Lessons Covered



Current lesson

Lesson 0—Course Overview

Lesson 1—Introduction to Spark

Lesson 2—Introduction to Programming in Scala

Lesson 3—Using RDD for Creating Applications in Spark

Lesson 4—Running SQL queries Using SparkSQL

Lesson 5—Spark Streaming

Lesson 6—Spark ML Programming

Lesson 7—Spark GraphX Programming

Apache Sp



In addition to the lessons, there are demos available in this course.

© Copyright 2015, Simplilearn. All rights reserved.

10

## Lessons Covered

In addition to the lessons, there are some demos provided in the course to facilitate a better understanding of the concepts.

This concludes 'Course Overview.'

The next lesson is 'Introduction to Spark.'

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

This concludes the course overview. The next lesson is Introduction to Spark.

## Lesson 1—Introduction to Spark



## Objectives



After completing this lesson, you will be able to:



- Describe the limitations of MapReduce in Hadoop
- Compare batch vs. real-time analytics
- Describe the application of stream processing and in-memory processing
- Explain the features and benefits of Spark
- Explain how to install Spark as a standalone user
- Compare Spark vs. Hadoop Eco-system

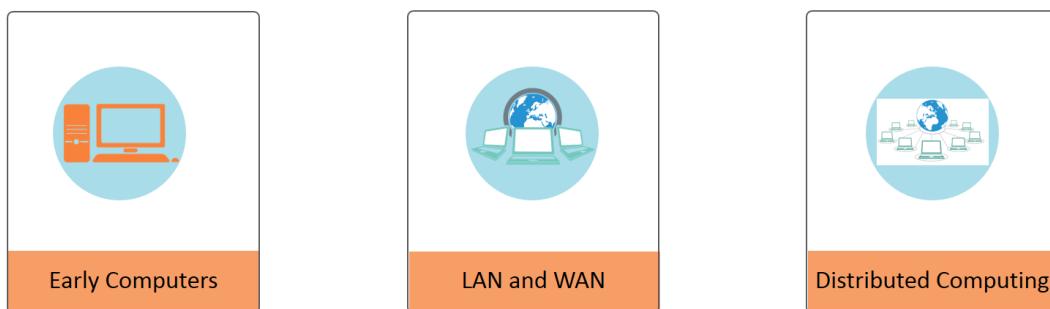
## Objectives

After completing this lesson, you will be able to:

- Describe the limitations of MapReduce in Hadoop
- Compare batch vs. real-time analytics
- Describe the application of stream processing and in-memory processing.
- Explain the features and benefits of Spark.
- Explain how to install Spark as a standalone user, and
- Compare Spark vs. Hadoop Eco-system.

## Evolution of Distributed Systems

The evolution of distributed systems went through the stages below:



Computers were quite expensive and heavy, accessible only in research labs of industries and universities, and to professional users. Then, new concepts were introduced to increase the power and utilization of CPU.

© Copyright 2015, Simplilearn. All rights reserved.

3

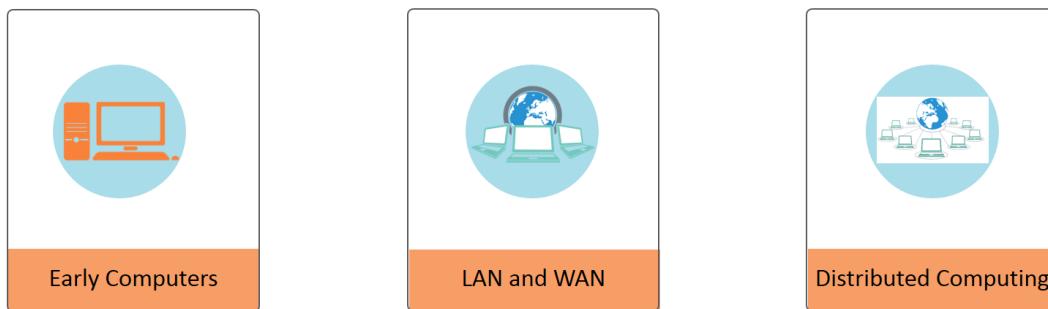
## Evolution of Distributed Systems

The evolution of distributed systems went through the stages mentioned on the screen.

In early days, computers were quite expensive and heavy. Those were accessible only in research labs of industries and universities, and to professional users. In addition, they used to have long job execution time. Various new concepts were introduced to increase the power and utilization of CPU, which helped in multiprogramming, automatic jobs, and long jobs processing.

## Evolution of Distributed Systems (contd.)

The evolution of distributed systems went through the stages below:



LAN allowed connecting local computers to exchange information at the rate of about 10, 100, and 1000Mbps. WAN allowed connecting far-located computers to exchange information at the rate of about 56Kbps/2, 34, 155, and 620Mbps.

### Evolution of Distributed Systems (contd.)

After that, LAN and WAN were introduced. While LAN allowed connecting local computers to exchange information located in a campus or building at the rate of about 10, 100, and 1000 Mbps, WAN allowed connecting far-located computers to exchange information at the rate of about 56Kbps, 2, 34,155, and 620 Mbps.

## Evolution of Distributed Systems (contd.)

The evolution of distributed systems went through the stages below:



Early Computers



LAN and WAN



Distributed Computing

Distributed computing allowed geographically-spread computers to network as they were in a single environment. It has different implementations. In some, computing entities only pass messages to each other.

© Copyright 2015, Simplilearn. All rights reserved.

5

## Evolution of Distributed Systems

Then came distributed computing that allowed geographically-spread computers to network as they were in a single environment. You may find it in different implementations. In some of them, computing entities only pass messages to each other.

## Need of New Generation Distributed Systems

Irrespective of decreasing reliability on centralized computing, there are still many organizations that keep a tight hold on their internal data center. Data centralization is not prevalent nowadays because of reasons such as:

Variety of Client Devices

The number and variety of client devices is increasing each year, leading to a complex array of end points to be served.

Social, Mobile, and Embedded Technology

The amount and variety of the collected data is increasing exponentially.

Landscape Transformation and Latency Reduction

With a few exceptions like High Frequency Trading (HFT), leveraging the distributed computing technology with parallel processing techniques transform the landscape and reduce latency.

## Need for New Generation Distributed Systems

We need a new generation of distributed systems. Let's understand why. Nowadays, it is very rare that an organization exists that depends only on its centralized computing. Irrespective of the fact, there are still many organizations that keep a tight hold on their internal data center and avoid any absolutely required data distribution. This sometimes happens because of their heavy investments in the infrastructure. Data centralization is becoming less popular because of various reasons.

One of those reasons is variety of client devices. The number and variety of these devices is increasing each year, leading to a complex array of end points to be served.

Another reason is Social, Mobile, and Embedded Technology, as the amount and variety of the collected data is increasing exponentially.

Also, Landscape Transformation and Latency Reduction is causing data centralization to decrease. With a few exceptions like High Frequency Trading or HFT, in which physically locating servers in a single location can lower latency, leveraging the distributed computing technology with parallel processing techniques transform the landscape and reduce latency.

## Limitations of MapReduce in Hadoop

The limitations of MapReduce in Hadoop are listed below:

### Unsuitable in real-time processing

Being batch oriented, it takes minutes to execute jobs depending upon the amount of data and number of nodes in the cluster.

### Unsuitable for trivial operations

For operations like Filter and Joins, you might need to rewrite the jobs, which becomes complex because of the key-value pattern.

### Unfit for large data on network

However, it works on the data locality principle, it cannot process a lot of data requiring shuffling over the network well.

## Limitations of MapReduce in Hadoop

MapReduce used in Hadoop is not suitable for many reasons, such as it is not a good choice when it comes to real-time processing. It is batch oriented, because of which it is executed as periodic jobs that take time to process the data and provide results. It takes minutes to complete a job, which mainly depends on the data amount and number of nodes in the cluster.

MapReduce is also not suitable for writing trivial operations such as Filter and Joins. To write such operations, you might need to rewrite the jobs using the MapReduce framework, which becomes complex because of the key-value pattern. This pattern is required to be followed in reducer and mapper codes.

In addition, MapReduce doesn't work well with large data on a network. It works on the data locality principle and hence works well on the node where the data actually resides. However, it is not a good option when you need to process a lot of data requiring shuffling over the network. The reason is that it will take a lot of time to copy the data, which may cause bandwidth issues.

## Limitations of MapReduce in Hadoop (contd.)

A few more limitations are:

### Unsuitable with OLTP

OLTP requires a large number of short transactions, as it works on the batch-oriented framework.

### Problems with Namenode

Namenode tracks the metadata, 600 bytes per file, in your distributed file system. This can be a problem in case of too many files.

### Unfit for processing graphs

The Apache Giraph library processes graphs, which adds additional complexity on top of MapReduce.

### Unfit for iterative execution

Being a state-less execution, MapReduce doesn't fit with use cases like Kmeans that need iterative execution.

## Limitations of MapReduce in Hadoop (contd.)

MapReduce is also unsuitable with OLTP that includes a large number of short transactions. Since it works on the batch-oriented framework, it lacks latency of seconds or sub seconds.

Another limitation exists with Namenode that tracks the metadata of about 600 bytes per file, as estimated by Yahoo. This means that in case of too many files, there can be a problem with Namenode.

Additionally, MapReduce is unfit for processing graphs. Graphs represent the structures to explore relationships between various points, for example, finding common friends in Social Media like Facebook. Hadoop has Apache Giraph library for such cases; however on top of MapReduce, it adds to complexity.

Another important limitation is its unsuitability for iterative execution of programs. Some use cases, like Kmeans, need such execution where data needs to be processed again and again for refining results. MapReduce, being a state-less execution, runs from the start every time.

## Batch vs. Real-Time Processing

The features below show a comparison of batch and real-time analytics in the enterprise use cases:

**Batch Processing**

- Large group of data/transactions is processed in a single run.
- Jobs run without any manual intervention.
- The entire data is pre-selected and fed using command-line parameters and scripts.
- It is used to execute multiple operations, handle heavy data load, reporting, and offline data workflow.

**Example:** Regular reports requiring decision making

**Real-Time Processing**

- Data processing takes place upon data entry or command receipt instantaneously.
- It must execute on response time within stringent constraints.

**Example:** Fraud detection

© Copyright 2015, Simplilearn. All rights reserved.

9

## Batch vs. Real-Time Processing

The features listed on the screen show a comparison of batch and real-time processing in case of the enterprise use cases.

In case of batch processing, a large amount of data or transactions is processed in a single run over a time period. The associated jobs generally run entirely without any manual intervention. Additionally, the entire data is pre-selected and fed using command-line parameters and scripts. In typical cases, it is used to execute multiple operations, handle heavy data load, reporting, and offline data workflow. An example is to generate daily or hourly reports for the purpose of decision making.

On the other hand, real-time processing takes place upon data entry or command receipt instantaneously. It needs to execute on response time within stringent constraints. An example is fraud detection.

Note that Hadoop has different sub systems like Pregel, GraphX, S4, and Drill for different business use cases. It would be better to have just one processing framework to solve all these use cases.

## Application of Stream Processing

Stream processing fits well for applications showing the following three characteristics:

Compute Intensity	Defined as the number of arithmetic operations per global memory or Input/Output reference
Data Parallelism	Exists in a kernel when a function is applied to an input stream's records and multiple records can be processed simultaneously
Data Locality	Is a particular type of temporal locality that is general in media and signal processing applications; intermediate streams can capture this locality directly

## Application of Stream Processing

Stream processing fits well for applications showing three characteristics.

Let's first talk about computer intensity, which is defined as the number of arithmetic operations per global memory or Input/Output reference. Today, in various signal processing applications, this intensity is well above 50:1. Also, it is increasing with the complexity of algorithms.

The next feature is data parallelism that exists in a kernel when a function is applied to an input stream's records and multiple records can be processed simultaneously. This should happen without results waiting from the previous records.

Data locality is the third feature that is a particular type of temporal locality and is general in media and signal processing applications, in which data is produced and read once or twice, and then never again read. Intermediate streams can capture this locality directly. These are the streams that are passed between kernels and the data within kernel functions and they do it using the model of stream processing programming.

## Application of In-Memory Processing

With column-centric databases coming, the similar information can be stored together. The working of in-memory processing can be explained as below:

- The entire information is loaded into memory, eliminating the need for indexes, aggregates, optimized databases, star schemas, and cubes.
- Compression algorithms are used by most of the in-memory tools, thereby reducing the in-memory size.
- Users querying the data loaded into the memory is different from caching.
- With in-memory tools, the analysis of data can be flexible in size and can be accessed within seconds by concurrent users with an excellent analytics potential.
- It is possible to access visually rich dashboards and existing data sources.

## Application of In-Memory Processing

With column-centric databases coming, the similar information can be stored together and hence data can be stored with more compression and efficiency. It also permitted to store large data amounts in the same space, which thereby reduced the memory amount required for performing a query and also increased the speed of processing. In an in-memory database, the entire information is loaded into memory, eliminating the need for indexes, aggregates, optimized databases, star schemas, and cubes.

With the use of in-memory tools, compression algorithms can be implemented that thereby decrease the in-memory size, even beyond what is required for hard disks.

Users querying the data loaded into the memory is different from caching. This also helps to avoid performance bottlenecks and slow database access. Caching is a popular method for speeding up the performance of a query, where caches are subsets of very particular organized data that are defined already.

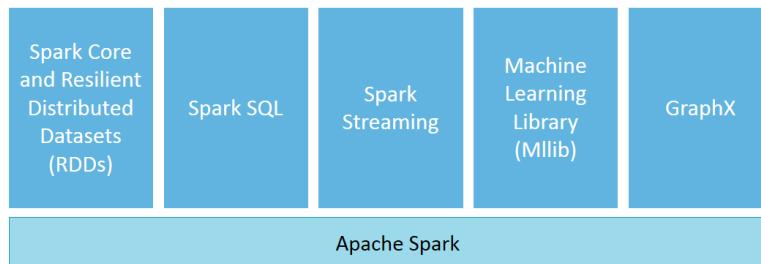
With in-memory tools, the analysis of data can be flexible in size and can be accessed within seconds by concurrent users with an excellent analytics potential. This is possible as data lies completely in memory. In theoretical terms, this leads to data access improvement that is 10,000 to 1,000,000 times faster, as compared to a disk. In addition, it also reduces the performance tuning need by the IT folks and hence provides faster data access for end users.

With in-memory processing, it is also possible to access visually rich dashboards and existing data sources. This ability is provided by several vendors. In turn, this allows end users and business analytics to create customized queries and reports without any need of extensive expertise or training.

## Introduction to Apache Spark

It:

- Is an open-source cluster computing framework
- Provides up to 100 times faster performance for a few applications with in-memory primitives, as compared to the two-stage disk-based MapReduce paradigm of Hadoop
- Is suitable for machine learning algorithms, as it allows programs to load and query data repeatedly



© Copyright 2015, Simplilearn. All rights reserved.

12

## Introduction to Apache Spark

Apache Spark is an open-source cluster computing framework that was initially developed at UC Berkeley in the AMPLab.

As compared to the disk-based, two-stage MapReduce of Hadoop, Spark provides up to 100 times faster performance for a few applications with in-memory primitives.

This makes it suitable for machine learning algorithms, as it allows programs to load data into the memory of a cluster and query the data constantly.

As shown on the screen, a Spark project contains various components such as Spark Core and Resilient Distributed Datasets or RDDs, Spark SQL, Spark Streaming, Machine Learning Library or MLlib, and GraphX.

## Components of a Spark Project

The components of a Spark project are explained below:

Spark Core and RDDs	As the foundation, it provides basic I/O, distributed task dispatching, and scheduling. RDDs can be created by applying coarse-grained transformations or referencing external datasets.
Spark SQL	As a component lying on the top of Spark Core, it introduces SchemaRDD, which can be manipulated. It supports SQL with ODBC/JDBC server and command-line interfaces.
Spark Streaming	It leverages the fast scheduling capability of Spark Core, ingests data in small batches, and performs RDD transformations on them.
MLLib	As a distributed machine learning framework on top of Spark, it is nine times faster than the Hadoop disk-based version of Apache Mahout.
GraphX	Being a distributed graph processing framework on top of Spark, it gives an API and provides an optimized runtime for the Pregel abstraction.

## Components of a Spark Project

The first component is Spark Core and RDDs, which is the foundation of the entire project. It provides basic Input/Output functionalities, distributed task dispatching, and scheduling. RDDs is the basic programming abstraction and is a collection of data that is partitioned across machines logically. These can be created by applying coarse-grained transformations on the existing RDDs or by referencing external datasets. The examples of these transformations are reduce, join, filter, and map. The abstraction of RDDs is exposed similarly as in-process and local collections through a language-integrated API in Python, Java, and Scala. As a result, the complexity of programming is simplified, as the manner in which applications change RDDs is similar to changing local data collections.

Spark SQL is a component lying on the top of Spark Core. It introduces SchemaRDD, which is a new data abstraction and supports semi-structured and structured data. This abstraction can be manipulated in Java, Scala, and Python by the Spark SQL provided a domain-specific language. In addition, Spark SQL supports SQL with ODBC/JDBC server and command-line interfaces.

The next component Spark Streaming leverages the fast scheduling capability of Spark Core for streaming analytics, ingests data in small batches, and performs RDD transformations on them. With this design, the same application code set that is written for batch analytics can be used on a single engine for streaming analytics.

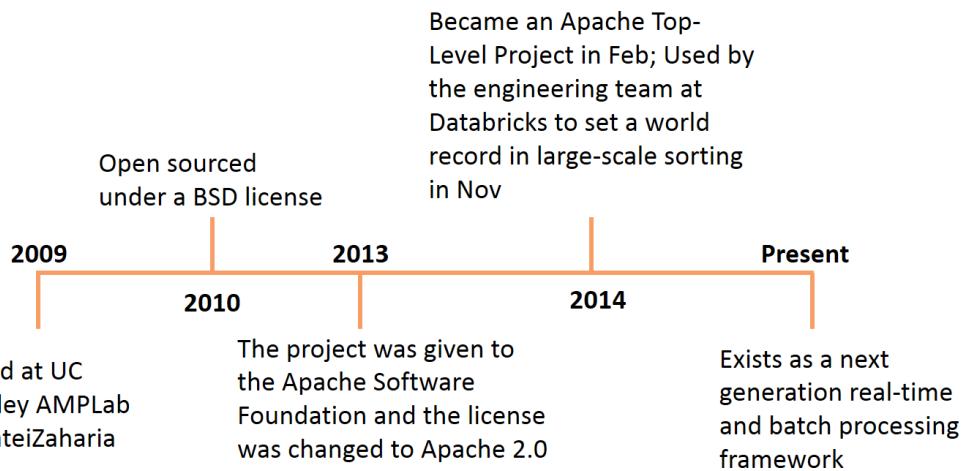
Machine Learning Library lies on the top of Spark and is a distributed machine learning framework. With its memory-based architecture, it is nine times faster than the Apache Mahout's Hadoop disk-based

version. In addition, the library performs even better than Vowpal Wabbit. In addition, it applies various common statistical and machine learning algorithms.

The last component, GraphX also lies on the top of Spark and is a distributed graph processing framework. For the computation of graphs, it provides an API and an optimized runtime for the Pregel abstraction. The API can also model this abstraction.

## History of Spark

The history of Spark is explained below:



© Copyright 2015, Simplilearn. All rights reserved.

14

## History of Spark

As discussed, Spark was started at UC Berkeley AMPLab by MateiZaharia in the year 2009. It was in 2010 when it was open sourced under a BSD license. The project was then donated to the Apache Software Foundation and the license was changed to Apache 2.0 in the year 2013. In the month of February 2014, Spark became an Apache Top-Level Project. Then in November of the same year, it was used by the engineering team at Databricks to set a world record in large-scale sorting. Now, Databricks provides commercial support and they provide certification for it. At present, Spark exists as a next generation real-time and batch processing framework.

## Language Flexibility in Spark

Spark is popular for its performance benefits over MapReduce. Another important benefit is language flexibility, as explained below:

Support to various development languages

Spark supports popular development languages like Java, Scala, and Python and will likely support R.

Capability to define functions in-line

With the temporary exception of Java, a common element in these languages is that they provide methods to express operations using lambda functions and closures.

## Language Flexibility in Spark

We have already discussed that Spark provides performance, which in turn provides developers an experience that they won't forget easily. Spark is considered over MapReduce, mainly for its performance advantages and versatility. Apart from this, another critical advantage is its development experience. Language flexibility is another important benefit that we will discuss here.

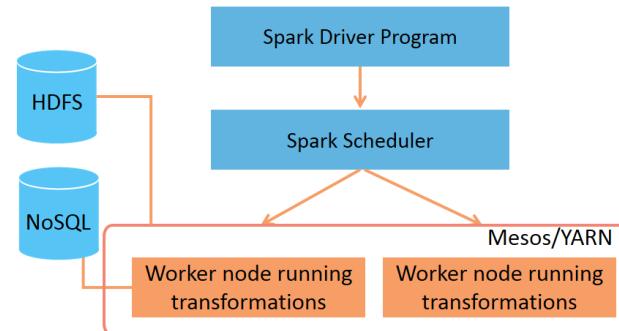
Spark provides support to various development languages like Java, Scala, and Python and will likely support R.

In addition, it has the capability to define functions in-line. With the temporary exception of Java, a common element in these languages is that they provide methods for expressing operations using lambda functions and closures. Using closures, you can use the application core logic to define the functions in-line, which helps to create easy-to-comprehend code and preserve application flow.

## Spark Execution Architecture

The components of the Spark execution architecture are explained below:

- **Spark-submit script:** Used to launch applications on a cluster; can use all cluster managers through a uniform interface
- **Spark applications:** Run as independent sets of processes on a cluster and are coordinated by the SparkContext object in the driver program
- **Cluster managers:** Supported cluster managers are: Standalone, Apache Mesos, and Hadoop YARN
- **Spark's EC2 launch scripts:** Make launching a standalone cluster easy on Amazon EC2



© Copyright 2015, Simplilearn. All rights reserved.

16

## Spark Execution Architecture

The components of Spark Execution Architecture are listed on the screen. It has Spark-submit script that is used to launch applications on a Spark cluster. It can use all cluster managers, supported by Spark, using an even interface. Due to this, it is not required to configure your application for each one particularly.

The next component is Spark applications. These applications run as sets of processes independently on a Spark cluster.

These are coordinated by the SparkContext object in the driver program, which is your main program.

SparkContext can connect to different cluster managers, which are of three types, Standalone, Apache Mesos, and Hadoop YARN. A standalone cluster manager is a simple one that makes setting up a cluster easy. Apache Mesos is a general cluster manager that is also capable of running service applications and MapReduce. On the other hand, Hadoop YARN is the resource manager in Hadoop 2.

The last component, EC2 launch scripts, makes launching a standalone cluster easy on Amazon EC2.

The interaction of these components is shown in the diagram displayed on the screen.

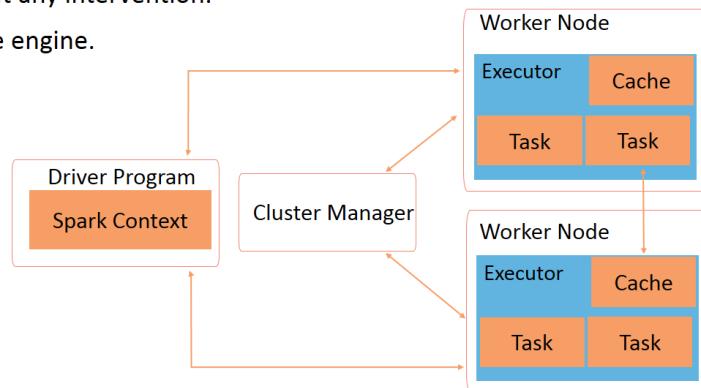
## Automatic Parallelization of Complex Flows

It is explained through the points below:

- It is your task to make the sequence of MapReduce jobs parallel in case of a complex pipeline. Here, a scheduler tool like Apache Oozie is generally required.
- The series of individual tasks is expressed as a single program flow, which allows to parallelize the flow of operators automatically without any intervention.
- This allows certain optimizations to the engine.

**Example:**

```
1 rdd1.map(splitlines).filter("ERROR")
2 rdd2.map(splitlines).groupByKey()
3 rdd2.join(rdd1, key).take(10)
```



© Copyright 2015, Simplilearn. All rights reserved.

17

## Automatic Parallelization of Complex Flows

Let's now talk about the next feature of Spark, automatic parallelization of complex flows.

It is your task to make the sequence of MapReduce jobs parallel in case of a complex pipeline. Here, a scheduler tool like Apache Oozie is generally required for constructing this sequence carefully.

Using Spark, the series of individual tasks is expressed in terms of a single program flow. To give a whole picture of the execution graph to the system, this flow is lazily evaluated. Using this approach, the core scheduler can map the dependencies lying between various application stages correctly. This allows parallelizing the operators flow automatically without any intervention.

With this capability, you can also achieve a few optimizations to the engine with less burden.

An example of such a job is given on the screen. The screen also shows how this parallelization works through the given diagram.

## Automatic Parallelization of Complex Flows—Important Points

Some important points to be noted about this architecture are:

- Every application has its own executor processes, which run tasks in multiple threads and stay till the duration of the entire application. While it benefits in terms of scheduling and executor sides, it implies that without writing to an external storage system, data cannot be shared across applications.
- Till Spark can obtain executor processes, and these can connect, it is comparatively relaxed to run it even on a cluster manager supporting other applications.
- The driver program needs to listen and accept connections coming from its executors all the time.
- The driver schedules tasks on the cluster. Therefore, it should run in proximity to the worker nodes. To send remote requests to the cluster, open an RPC to the driver and let it submit operations.

## Automatic Parallelization of Complex Flows—Important Points

An important point about this structure is that every application has its own executor processes, which run tasks in various threads and stay till the duration of the entire application. While it benefits in terms of scheduling and executor sides by separating applications, it also implies that without writing to an external storage system, you cannot share data across applications of Spark.

Another feature is the agnostic behavior of Spark to the cluster manager underlying. Till Spark can obtain executor processes, and these can connect, it is comparatively relaxed to run it even on a cluster manager supporting other applications such as YARN and Mesos.

Note that the driver program needs to listen and accept connections coming from its executors all the time. In other words, the driver program must be accessible to the network to be addressed by the worker nodes.

The driver schedules the tasks on the cluster. Therefore, it should run in proximity to the worker nodes, if possible on the same local network. To send remote requests to the cluster, you should open an RPC to the driver and let it submit operations from neighborhood. This is better than running a driver far through the worker nodes.

## APIs That Match User Goals



Some developers turn to the higher-level APIs for writing their MapReduce jobs, as there are no built-in features to simplify the process.

### Example API Operators:

*Filter, Collect, Count, countByValue, Distinct, Filter, flatMap, groupByKey, Join, Map, mapPartitions, Reduce, reduceByKey, sortByKey, Union, sample, and so on...*



When scripting frameworks like Apache Pig, many high-level operators are also available. Spark allows you to access them as in typical programming environment.

## APIs that Match User Goals

As a developer, when you work with MapReduce, you generally get forced to combine basic operations to make them customer Mapper/Reducer jobs. This happens because there is no built-in feature that could streamline this process. Therefore, some developers turn to the higher-level APIs for writing their MapReduce jobs. These APIs are provided by frameworks such as Cascading and Apache Crunch.

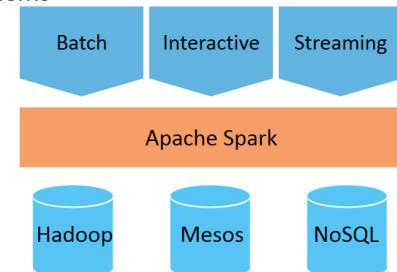
On the other hand, Spark provides a powerful and ever-growing operators library. These APIs contain functions for the operators listed on the screen. These are just a few examples. There are above 80 operators available in Spark. While a few of them provide you operations that are equivalent to MapReduce operations, the others are high level and allow you to write much more precisely.

Note that when scripting frameworks such as Apache Pig, many high-level operators are also available. Spark lets you access them in the full programming language context. As a result, you can use functions, classes, and control statements, as in a typical environment of programming.

## Apache Spark—A Unified Platform of Big Data Apps

The different advantages of Spark are:

- **Speed:** Extending the MapReduce model to support computations like stream processing and interactive queries
- **Combination:** Covering various workloads that used to require different distributed systems, which makes combining different processing types and allows easy tools management
- **Hadoop Support:** Allowing to create distributed datasets from any file stored in the Hadoop Distributed File System (HDFS) or any other supported storage systems



#### Why does unification matter?

- Developers need to learn only one Platform
- Users can take their apps to everywhere

## Apache Spark—A Unified Platform of Big Data Apps

When it comes to speed, Spark has extended the MapReduce model to support computations like stream processing and interactive queries. The feature of speed is critical to process large datasets, as this implies the difference of waiting for hours or minutes and exploring the data interactively. Spark supports running computations in memory. Also, the related system is more effective as compared to MapReduce when it comes to running complex applications on a disk. These features add to the speed capability of Spark.

Spark covers various workloads that used to require different distributed systems such as streaming, iterative algorithms, and batch applications. As these workloads are supported on the same engine, combining different processing types is easy. It is normally required in production data analysis pipelines. The combination feature also allows easy management of separate tools.

Spark is capable of creating distributed datasets from any file that is stored in the Hadoop Distributed File System or HDFS or any other supported storage systems. You should note that Spark does not need Hadoop. It just supports the storage systems that implement the APIs of Hadoop. It supports SequenceFiles, Parquet, Avro, text files, and all other Input/Output formats of Hadoop.

Now the question is why unification matters. Unification not only provides developers the ease of learning only one platform, but also allows users to take their apps everywhere.

The graphic shows the apps and systems that can be combined in Spark.

## More Benefits of Apache Spark



Some more benefits of Spark are:

- Contains various closely-integrated components for distributing, scheduling, and monitoring applications with many computational tasks
- Empowers various higher-level components specialized for different workloads like machine learning or SQL
- Integrates tightly allowing to create applications that easily combine different processing models; for example, ability to write an application using machine learning to categorize data in real time as it is ingested from sources of streaming
- Allows to access the same data through the Python shell for ad-hoc analysis and in standalone batch applications

## More Benefits of Apache Spark

A Spark project includes various closely-integrated components for distributing, scheduling, and monitoring applications with many computational tasks across a computing cluster or various worker machines.

The Spark's core engine is general purpose and fast. As a result, it empowers various higher-level components that are specialized for different workloads like machine learning or SQL. These components can interoperate closely.

Another important benefit is that it Integrates tightly, allowing to create applications that easily combine different processing models; for example, ability to write an application using machine learning to categorize data in real time as it is ingested from sources of streaming.

Additionally, it allows analysts to query the data thus resulted through SQL. Moreover, data scientists and engineers can access the same data through the Python shell for ad-hoc analysis and in standalone batch applications. For all this, the IT team needs to maintain just one system.

## Running Spark in Different Modes

The different deployment modes of Spark are explained below:

### Spark as Standalone

Can be launched manually, by using launch scripts, or starting a master and workers; used for development and testing

### Spark on Mesos

Has advantages like scalable partitioning among different Spark instances and dynamic partitioning between Spark and other frameworks

### Spark on YARN

Has all parallel processing and all benefits of the Hadoop cluster

### Spark on EC2

Has key-value pair benefits of Amazon

## Running Spark in Different Modes

The different deployment modes of Spark are explained on the screen. The standalone mode is a simple one that can be launched manually, by using launch scripts, or starting a master and workers. This mode is usually used for development and testing.

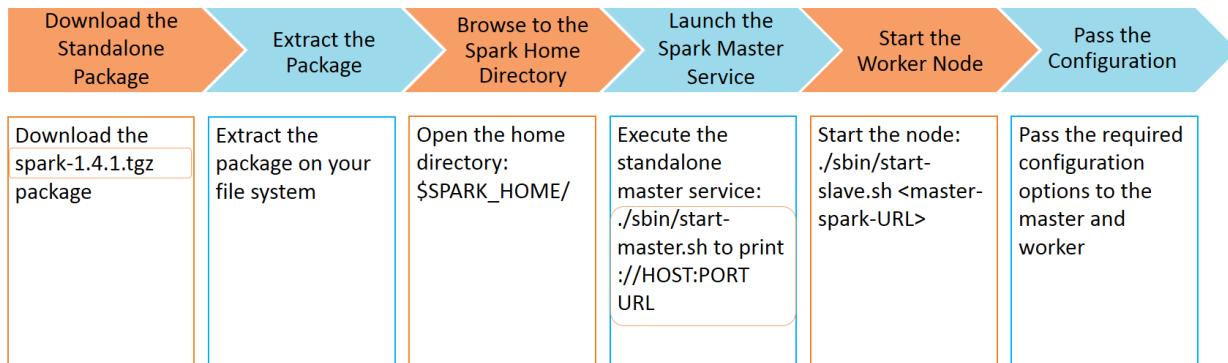
Spark can also be run on hardware clusters that are managed by Mesos. Running Spark in this mode has advantages like scalable partitioning among different Spark instances and dynamic partitioning between Spark and other frameworks.

Running Spark on YARN has all parallel processing and all benefits of the Hadoop cluster.

By running Spark on EC2, you have key value pair benefits of Amazon.

## Installing Spark as a Standalone Cluster—Configurations

Place a complied Spark version on each cluster node. The steps to install Spark as a standalone cluster are mentioned below:



## Installing Spark as a Standalone Cluster—Configurations

To install Spark as a standalone cluster, you just need to place a complied Spark version on each cluster node. To do so, you can either get a pre-built Spark version or create it by yourself. As the first step, download the latest package on your system. The latest version is mentioned on the screen.

Next, you need to extract the package anywhere on your file system.

Once the extraction is done, you need to open the Home directory of Spark.

After this, you need to launch the Spark master service, which is mentioned on the screen. Once it is started, the master will print out the given service. This can be used to connect workers or pass as the “master” argument to SparkContext.

Next, you need to start one or more workers and connect them to the master using the given service.

Once a worker has started, look at the web UI of the master. The new node must be listed there with the information like CPU numbers and memory.

As the final step, you must pass the required configuration options to the master and worker.

## Installing Spark as a Standalone Cluster—Configurations

The configuration options that can be passed to the master and worker are listed in the table below:

Argument	Explanation
-h HOST, --host HOST	Hostname to listen on
-i HOST, --ip HOST	Hostname to listen on (deprecated, use -h or --host)
-p PORT, --port PORT	Port for service to listen on (default: random for worker, 7077 for master)
--webui-port PORT	Port for web UI (default: 8081 for worker, : 8080 for master)
-c CORES, --cores CORES	Total CPU cores allowing Spark applications to use on the machine (default: all available); only on worker
-m MEM, --memory MEM	Total amount of memory allowing Spark applications to use on the machine, in a format like 1000M or 2G (default: the total RAM on machine minus 1 GB); only on worker
-d DIR, --work-dir DIR	Directory to use for job output logs and scratch space (default: SPARK_HOME/work); only on worker
--properties-file FILE	Path to a custom Spark properties file to load (default: conf/spark-defaults.conf)

## Installing Spark as a Standalone Cluster—Configurations

The configuration options that can be passed to the master and worker are listed in the given table on the screen.

This demo will show the steps to install Apache Spark on a Linux machine.

© Copyright 2015, Simplilearn. All rights reserved.

25

## Demo—Install Apache Spark

In this demo, you will learn to install Apache Spark on a Linux machine.

As the first step, download the latest spark tar file.

Once the spark binary is downloaded, move the tgz file to the home directory. Now, untar it by executing the given command.

Now, you need to add the spark installation path in the ,bashrc file so that you can run Spark from anywhere in Linux . Add the given lines in the ,bashrc file by executing the given command. Note that your .bashrc file location might be different based on your username and profile.

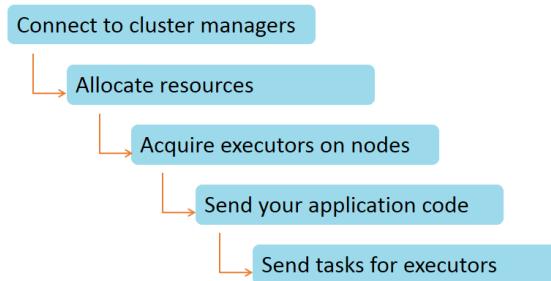
Let's now reload the environment variable by executing the given command.

Type “echo \$SPARK\_HOME” to verify whether the SPARK\_HOME environment variable has been correctly defined or not.

Go to the sbin directory of the SPARK installed directory and type the given command to start master and worker daemons process.

## Overview of Spark on a Cluster

The below steps depict how Spark applications (run independently as different processes sets on a cluster) run on a cluster:



## Overview of Spark on a Cluster

The steps listed on the screen depict how Spark applications run on a cluster.

These applications run independently as different processes sets on a cluster. The `SparkContext` object that lies in your main program coordinates the process. For running on a cluster, this object can connect to different types of cluster managers. We have already discussed different types of cluster managers. This allocates resources across applications. Once the system is connected, Spark acquires executors on nodes in the cluster. Executors are the processes that store data for applications and run computations. Then, it sends the application code to executors. At the end, `SparkContext` sends tasks for executors to run.

## Tasks of Spark on a Cluster

A few tasks performed on a Spark cluster are listed below:

### Submitting Applications

Submit applications to a cluster using the spark-submit script.

### Monitoring

Access the `http://<driver-node>:4040` URL to access the web UI of driver programs.

### Scheduling Jobs

Control resource allocation both across and within applications.

## Tasks of Spark on a Cluster

One of the tasks that is performed on a Spark cluster is submitting applications of any type to a cluster. This is done using the spark-submit script.

Another task is monitoring. Every driver program has a web-based UI that usually lies on port 4040. It shows information about the storage usage, executors, and running tasks. To access the same, go to the given URL.

You can also schedule jobs on a cluster as Spark provides control over resource allocation both across and within applications.

## Companies Using Spark—Use Cases



Companies like NTTDATA, Yahoo, Groupon, NASA, Nokia, and more are using Spark for creating applications for different use cases such as:

- Stream processing of network machine data
- Performing Big Data analytics for subscriber personalization and profile in the telecommunications domain
- Executing the Big Content platform, which is a B2B content asset management service that provides an aggregated and searchable source of public domain media, live news feeds, and archives of content
- Building data intelligence and eCommerce solutions in the retail industry
- Analyzing and visualizing patterns in large-scale recordings of brain activities

## Companies Using Spark—Use Cases

Companies like NTTDATA, Yahoo, Groupon, NASA, Nokia, and more are using Spark for creating applications for different use cases.

These use cases are stream processing of network machine data, Performing Big Data analytics for subscriber personalization and profile in the telecommunications domain, and executing the Big Content platform, which is a B2B content asset management service that provides an aggregated and searchable source of public domain media, live news feeds, and archives of content.

A few more use cases are Building data intelligence and eCommerce solutions in the retail industry and analyzing and visualizing patterns in large-scale recordings of brain activities.

## Hadoop Ecosystem vs. Apache Spark

Let's now compare the main features of the Hadoop Ecosystem and Apache Spark.

**Hadoop Ecosystem**

- Uses MapReduce for batch analytics
- Supports third-party plugins/tools such as Talend
- Supports pig or hive queries

**Apache Spark**

- Supports both real-time and batch processing

## Hadoop Ecosystem vs. Apache Spark

The Hadoop Ecosystem allows storing large files on various machines. It uses MapReduce for batch analytics that is easy as it is distributed in nature. In Hadoop, third-party support is also available, for example by using ETL talend tools, various batch-oriented workflows can be designed. In addition, it supports pig or hive queries that non-Java developers can use and prepare batch workflows using SQL scripts.

On the other hand, Apache supports both real-time and batch processing.

## Hadoop Ecosystem vs. Apache Spark (contd.)

You can perform every type of data processing using Spark that you execute in Hadoop.

- **Batch Processing:** Spark batch can be used over Hadoop MapReduce.
- **Structured Data Analysis:** Spark SQL can be used using SQL.
- **Machine Learning Analysis:** MLlib can be used for clustering, recommendation, and classification.
- **Interactive SQL Analysis:** Spark SQL can be used over Stringer/Tez/Impala.
- **Real-time Streaming Data Analysis:** Spark streaming can be used over specialized library like Storm.

Hadoop	Spark
Hive	SparkSQL
Apache Graph	Graphax
Impala	SparkSQL
Apache Storm	Spark Streaming
Apache Mahout	MLlib

## Hadoop Ecosystem vs. Apache Spark (contd.)

You can perform every type of data processing using Spark that you execute in Hadoop.

For batch processing, Spark batch can be used over Hadoop MapReduce.

For Structured Data Analysis, Spark SQL can be used using SQL.

For Machine Learning Analysis, Machine Learning Library can be used for clustering, recommendation, and classification.

For Interactive SQL Analysis, Spark SQL can be used over Stringer, Tez, and Impala.

In addition, for real-time Streaming Data Analysis, Spark streaming can be used over specialized library like Storm.

**QUIZ  
1**

Which of the following is true about Spark? *Select all that apply.*

- a. It can be used for batch analytics.
- b. It can be used for real-time analytics.
- c. It can be used for streaming analytics.
- d. It supports the key-value pattern.

**QUIZ  
2**

Which of the following is a distributed machine learning framework on the top of Spark?

- a. Spark SQL
- b. R-hadoop
- c. M-Lib
- d. GraphX



**QUIZ  
3**

Which of the following language can be used in Spark for doing data processing? *Select all that apply.*

- a. Java
- b. Scala
- c. Python
- d. C#

**QUIZ  
4**

Which of the following are supported by Spark? *Select all that apply.*

- a. HDFS data for processing
- b. NoSQL data for processing
- c. Apache Giraph library processes
- d. In-memory data for processing



**ANSWERS:**

S.No.	Question	Answer & Explanation
1	Which of the following is true about Spark?	a., b., and c. Spark can be used for batch analytics, real-time analytics, and streaming analytics. The key-value pattern is the feature of MapReduce.
2	Which of the following is a distributed machine learning framework on the top of Spark?	c. M-Lib is a distributed machine learning framework on the top of Spark. It is nine times faster than the Hadoop disk-based version of Apache Mahout.
3	Which of the following language can be used in Spark for doing data processing?	a., b., and c. Scala supports Java, Scala, and Python as data processing languages.
4	Which of the following are supported by Spark?	a., b., and d. Using Spark, you can perform processing for data stored in HDFS, NoSQL, or in memory. Apache Giraph library processes is the feature of MapReduce..

## Summary

Let us summarize the topics covered in this lesson:



- Data centralization is becoming less popular because of various reasons such as variety of client devices.
- MapReduce in Hadoop has many limitations such as it is unsuitable for real-time processing.
- Apache Spark is an open-source cluster computing framework.
- The components of a Spark project are Spark Core and RDDs, Spark SQL, Spark Streaming, MLlib, and GraphX.
- Spark is popular for its performance benefits over MapReduce. Another important benefit is language flexibility.

## Summary

Let us summarize the topics covered in this lesson:

- Data centralization is becoming less popular because of various reasons such as variety of client devices.
- MapReduce in Hadoop has many limitations such as it is unsuitable for real-time processing.
- Apache Spark is an open-source cluster computing framework.
- The components of a Spark project are Spark Core and RDDs, Spark SQL, Spark Streaming, MLlib, and GraphX.
- Spark is popular for its performance benefits over MapReduce. Another important benefit is language flexibility.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- The components of the Spark execution architecture are Spark-submit script, Spark applications, SparkContext, cluster managers, and EC2 launch scripts.
- The different advantages of Spark are speed, combination, unification, and Hadoop support.
- The different deployment modes of Spark are standalone, on Mesos, on YARN, and on EC2.
- Companies like NTTDATA, Yahoo, GROUPON, NASA, Nokia, and more are using Spark for creating applications for different use cases.
- You can perform every type of data processing using Spark that you execute in Hadoop.

## Summary (contd.)

- The components of the Spark execution architecture are Spark-submit script, Spark applications, SparkContext, cluster managers, EC2 launch scripts.
- The different advantages of Spark are speed, combination, unification, and Hadoop support.
- The different deployment modes of Spark are standalone, on Mesos, on YARN, and on EC2.
- Companies like NTTDATA, Yahoo, GROUPON, NASA, Nokia, and more are using Spark for creating applications for different use cases.
- You can perform every type of data processing using Spark that you execute in Hadoop.

simplilearn

This concludes ‘Introduction to Spark.’

The next lesson is ‘Introduction to Programming in Scala.’

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 1 “Introduction to Spark” of the Apache Spark and Scala course. The next lesson is Introduction to Programming in Scala.

## Lesson 2—Introduction to Programming in Scala



## Objectives



After completing this lesson, you will be able to:

- Explain the features of Scala
- List the basic data types and literals used in Scala
- List the operators and methods used in Scala
- Discuss a few concepts of Scala



© Copyright 2015, Simplilearn. All rights reserved.

2

## Objectives

After completing this lesson, you will be able to:

- Explain the features of Scala
- List the basic data types and literals used in Scala
- List the operators and methods used in Scala, and
- Discuss a few concepts of Scala

Scala is:

- A modern and multi-paradigm programming language for expressing common programming patterns
- A pure object-oriented language, as every value in it is an object
- A functional language, as every function in it is a value
- Statically typed; equipped with an expressive type system; supports features like annotations, classes, views, polymorphic methods, compound types, explicitly typed self references, and upper and lower type bounds
- Extensible; provides an exceptional combination of language mechanisms

## Introduction to Scala

Scala is a modern and multi-paradigm programming language. It has been designed for expressing general programming patterns in an elegant, precise, and type-safe way. One of the prime features is that it integrates the features of both object-oriented and functional languages smoothly.

It is a pure object-oriented language, as every value in it is an object. The objects' behavior and types are explained through traits and classes.

It is also a functional language, as every function in it is a value. By providing a lightweight syntax to define anonymous functions, it provides support for higher-order functions. In addition, the language also allows functions to be nested and provides support for currying. It also has features like case classes and pattern matching model algebraic types support.

In addition, Scala is statically typed, being empowered with an expressive type system. The system enforces the use of abstractions in a coherent and safe way. To be particular, this system supports various features like annotations, classes, views, polymorphic methods, compound types, explicitly typed self-references, and upper and lower type bounds.

When it comes to developing domain-specific applications, it generally needs domain-specific language extensions. Scala, being extensible, provides an exceptional combination of language mechanisms. Due to this, it becomes easy to add new language constructs as libraries.

## Features of Scala

The features of Scala are listed below:

General Purpose Programming Language	Operates in nearly any type of use cases in which other languages such as Java or C# are used
Multi-Paradigm Programming Language	Offers Object Oriented and functional programming constructs that can be combined to cut out the domain model
Expressive Type System with Type Inferencing	Supports statically checked duck typing, expressive syntax, and powerful constructs
Powerful Parser Combinator Library	Helps designing external DSLs by constructing monadic parsers
Rich Concurrency	Supports the Erlang model of concurrency and powerful frameworks like Akka ( <a href="http://akka.io">http://akka.io</a> )
Apache ActiveMQ and Camel Based System	Allows to create a data collection, processing, and analysis system
Interoperability with Java and .NET	Works well with Java 2 Runtime Environment (JRE) and .NET Framework (CLR)

© Copyright 2015, Simplilearn. All rights reserved.

4

## Features of Scala

The basic features of Scala are listed on the screen.

As discussed, it is a general purpose programming language. Therefore, you can use it in nearly any type of use cases in which other languages such as Java or C# are used. Like in all other languages, in Scala too, you can create your application more precisely than others.

We have also discussed that it is a multi-paradigm programming language. This means that it offers Object Oriented and functional programming constructs that can be combined to cut out the domain model.

In addition, Scala has an expressive type system with type inferencing. Therefore, till some point, you can get benefitted with both the features. Scala supports statically checked duck typing. Also, it has an expressive syntax and provides powerful constructs, using which you can create powerful abstractions and internal DSLs.

Scala incorporates a very powerful parser combinator library, using which you can design external DSLs by constructing monadic parsers. You can combine these parsers using the combinators.

Another important feature is its rich concurrency that primarily comes through actors. Scala supports the Erlang model of concurrency. With Scala, you get powerful frameworks such as Akka, in which the features of actors can be combined with Software Transactional Memory or STM. You can access this framework using the URL given on the screen. Scala therefore proves to be a powerful tool if your use case includes a lot of stuff dealing with concurrent programming.

Scala lets you create a data collection, processing, and analysis system that is based on apache activemq and camel.

Also, Scala provides interoperability with Java and .NET. The language has been designed to work well with Java 2 Runtime Environment or JRE. It interacts smoothly with the mainstream object-oriented Java language. Scala works on the same compilation model of Java and lets you access multiple existing high-quality libraries. In addition, Scala also supports .NET Framework, also called CLR.

## Basic Data Types

Scala supports the same data types as Java does. The table below lists and explains the Scala data types:

Data Type	Explanation
Byte	8-bit signed value; Range: -128 to 127
Short	16-bit signed value; Range: -128 to 127
Int	32-bit signed value; Range: -128 to 127
Long	64-bit signed value; Range: -128 to 127
Float	Single-precision 32-bit IEEE 754 value
Double	Double-precision 64-bit IEEE 754 value
Char	16-bit unsigned Unicode character; Range: U+0000 to U+FFFF
String	Chars sequence
Unit	No value
Null	Empty or null reference
Boolean	Literal true or false
Any	Super type of any type
AnyRef	Super type of any reference type
Nothing	Sub type of every other type; No value contained

© Copyright 2015, Simplilearn. All rights reserved.

5

## Basic Data Types

Scala supports the same data types as Java does, with the same precision and memory footprint. The table on the screen lists and explains all the basic data types used in Scala. These are Byte, Short, Int, Long, Float, Double, Char, String, Unit, Null, Boolean, Any, AnyRef, and Nothing.

## Basic Literals

The basic literals used in Scala are listed below with examples:

### Integer Literals

- Types: Int, Long, Short, and Byte
- Forms: Decimal and Hexadecimal

**Example:**

```
scala> val hex = 0x6; output - hex: Int = 6
```

### Floating Type Literals

- Contain decimals, optionally a decimal point, followed by an E or e and an exponent

**Examples:**

- `scala>val big = 1.2345; output - big: Double = 1.2345`
- `scala>val little = 1.2345F; output - little: Float = 1.2345`

## Basic Literals

The rules used by Scala for literals are easy and intuitive. Let us first discuss the integer and floating type literals.

Integer literals are of Int, Long, Short, and Byte types. They are available in two forms Decimal and Hexadecimal. An example is given on the screen.

On the other hand, floating point literals contain decimal digits. They may optionally have a decimal point and may optionally be followed by an E and an exponent. A few examples for defining such literals are given on the screen.

## Basic Literals (contd.)

Other types of literals are:

### Character Literals

- Include any Unicode character between single quotes
- Are special character literal escape sequences

**Example:**

```
scala> val a = 'A'
```

### String Literals

- Contain characters surrounded by double quotes

**Examples:**

- ```
scala> val hello = "hello"; output - hello:  
java.lang.String = hello
```
- ```
println("""Welcome to Ultamix 3000 Type  
"HELP" for help.""")  
Welcome to Ultamix 3000  
Type "HELP" for help
```

© Copyright 2015, Simplilearn. All rights reserved.

7

## Basic Literals (contd.)

Other types of literals are Character literals and String literals.

Character literals contain any Unicode character between single quotes. They are special character literal escape sequences. An example is given on the screen.

On the contrary, string literals contain characters surrounded by double quotes. A few examples to define these literals are given on the screen.

## Basic Literals (contd.)

A few more literals are:

**Boolean Literals**

- Has two literals: True and False

**Examples:**

- `scala> val bool = true; output - bool:  
Boolean = true`
- `scala> val fool = false; output - fool:  
Boolean = false`

**Symbol Literals**

- Are written with idents, where an ident can be any alphanumeric identifier

**Example:**

```
scala> updateRecordByName('favoriteBook,  
"Spark in Action")
```

## Basic Literals (contd.)

Another type of literals is the Boolean type that has literals, true and false. A few examples for defining these literals are given on the screen.

One more type is symbol literals, which are written with idents. An ident can be any alphanumeric identifier. An example of it is given on the screen.

## Introduction to Operators

An operator:

- Is a symbol that allows performing specific logical or mathematical manipulations
- Provides a syntax for general method calls



**Example:**  $2 + 1$  actually implies  $(2).+(1)$

The class Int includes a method called + that takes an Int and provides an Int as a result. To invoke the + method, you need to add two Ints: `scala> val sum = 2 + 1 // Scala invokes (2).+(1)`

© Copyright 2015, Simplilearn. All rights reserved.

9

## Introduction to Operators

An operator is a symbol that conveys to the compiler that it needs to perform specific logical or mathematical manipulations. Actually, these operators provide a nice syntax for general method calls.

For instance, consider the example given on the screen. In this example, the class Int includes a method called + that takes an Int and provides an Int as a result. To invoke the + method, you need to add two Ints, as depicted on the screen.

## Types of Operators

Scala provides a rich set of built-in operators, which are:

### Arithmetic Operators

- Allow invoking arithmetic methods using infix operator notation for +, -, \*, /, and %, on a numeric type
- Example:** `scala> 1.2 + 2.3 // output - res6: Double = 3.5`

### Relational and Logical Operators

- Allow comparing numeric types using relational methods such as >, <,  $\geq$ , and  $\leq$ , and inverting a Boolean value using the ! operator
- Example:** `scala> 1.2 + 2.3 // output - res6: Double = 3.5`

### Bitwise Operators

- Allow performing operations on individual bits of integer types using bitwise methods such as &, |, and ^, and inverting each bit in its operand using the ~ operator
- Example:** `scala> 1 & 2 // output - res24: Int = 0`

### Object Equality Operators

- Allow comparing two objects for equality using either == or !=
- Example:** `scala> 1 == 2 // output - res31: Boolean = false`

© Copyright 2015, Simplilearn. All rights reserved.

10

## Types of Operators

Scala provides a rich set of built-in operators.

The first type of operators is Arithmetic Operators, which allow you to invoke arithmetic methods using infix operator notation for addition, subtraction, multiplication, division, and remainder on a numeric type. An example is given.

The next type is Relational and Logical Operators. Using these operators, you can compare numeric types using relational methods such as greater than, less than, greater than or equal to, and less than or equal to. All these methods provide a Boolean result. Additionally, you can use the unary operator for inverting a Boolean value. An example is given on the screen.

Bitwise operators allow you to perform operations on individual bits of integer types using bitwise methods such as bitwise and, bitwise or, and bitwise XOR.

You can also invert each bit in its operand using any unary bitwise complement operator. An example to use this operator is given on the screen.

The object equality operators allow you to compare two objects for equality using either the double equal operator or inverse. An example to use this operator is given on the screen

## Demo—Use Basic Literals and the Arithmetic Operator



This demo will show the steps to declare and use basic literals and the arithmetic operator in Scala.

## Demo—Use the Logical Operator



This demo will show the steps to declare and use the logical operator in Scala.

## Introduction to Type Inference

Type Inference is a built-in mechanism that allows to **omit**:

- Certain type annotations; for example, specifying the type of a variable is not required generally
- Return types of methods

**Example:**

```
object InferenceTest1 extends Application {  
    val x = 1 + 2 * 3 // the type of x is Int  
    val y = x.toString() // the type of y is String  
    def succ(x: Int) = x + 1 // method succ returns Int values  
}
```

## Introduction to Type Interface

Scala provides a built-in mechanism called type inference, using which you can omit certain type annotations. Therefore, for example, it is generally not required to specify the variable type, as the compiler can deduce it from the initialization expression of the variable.

In addition, you can also omit return types of methods, as they correspond to the body type and can be inferred by the compiler.

An example to define type inference is given on the screen.

## Type Inference for Recursive Methods

For such methods, the compiler cannot infer a result type.

**Example:**

```
object InferenceTest2 {  
    def fac(n: Int) = if (n == 0) 1 else n * fac(n - 1)  
}
```

## Type Inference for Recursive Methods

For recursive methods, the compiler cannot infer a result type. The program example shown on the screen will fail the compiler for the same reason.

## Type Inference for Polymorphic Methods and Generic Classes

When polymorphic methods are called or generic classes are instantiated, it is not required to specify type parameters.

**Example:**

```
case class MyPair[A, B](x: A, y: B);
object InferenceTest3 extends App{
  def id[T](x: T) = x
  val p = new MyPair(1, "scala") // type: MyPair[Int, String]
  val q = id(1) // type: Int
}
```

Here, the last two lines are equivalent to:

```
val x: MyPair[Int, String] = new MyPair[Int, String](1, "scala")
val y: Int = id[Int](1)
```

## Type Inference for Polymorphic and Generic Classes

When polymorphic methods are called or generic classes are instantiated, it is not required to specify type parameters. The reason is that the compiler of Scala will infer these missing type parameters from the actual method or constructor parameters and the context.

The example given on the screen illustrates the same concept. In this program, the last two lines are equivalent to the code given on the screen, where all inferred types have been made explicit.

## Unreliability on Type Inference Mechanism

In the program below, the type inferred for variable obj is Null, which is the only value of that type. Therefore, you cannot make this variable refer to another value.

**Example:**

```
object InferenceTest4 {  
    var obj = null  
    obj = new Object()  
}
```

## Unreliability on Type Inference Mechanism

In some scenarios, using type inference can become dangerous. For example, consider the program given on the screen. This program does not compile, as the type inferred for variable objects is Null. Because the only value of that type is null, you cannot make this variable refer to another value.

## Mutable Collection vs. Immutable Collection

A comparison between a mutable collection and an immutable collection is given in the table below:

Feature	Mutual	Immutable
Possible to Modify the Collection Object	Yes (Can append elements and change the collection's references with +=)	No (Need to build new collection objects with operations like + or ++)
Persistent Data Structures	Partially	Fully
Possible to Reassign a Val	No	Yes; if it is assigned to a var, that var can be assigned to a collection built from it using operations like +

## Mutable Collection vs. Immutable Collection

To understand the concepts of a mutable and immutable collection, let's compare them through the table given on the screen.

Mutable implies that it is possible to modify the collection in concern. Therefore, to change a collection "a" and every other reference to that collection, you can append elements using the given operator. On the other hand, immutable implies that the given collection object never changes. You would need to build new collection objects using the given operators. This helps in concurrent algorithms, because it doesn't need locking to add anything to a collection. However, this feature can prove to be very useful, it may come at the cost of some overhead.

As compared to mutable collections, immutable collections are fully persistent data structures.

The difference between these collections is very similar to what exists between variable and value. A mutable collection that is bound to a value can be altered; however, you cannot reassign the value. On the other hand, an immutable collection cannot be altered; however, if it is assigned to a variable, that var can be assigned to a collection built from it using operations like +.

## Functions

A function:

- Is a “first-class” value
- May be returned as a result or passed as a parameter



**Example:**

```
def sumInts(a: Int, b: Int): Int = {if (a > b) 0  
else  
  a + sumInts(a + 1, b)}
```



Higher-order functions are the ones that take other functions as parameters or return them as results.

© Copyright 2015, Simplilearn. All rights reserved.

18

## Functions

In Scala, a function is a first-class value. Therefore, like other values, these can be returned as a result or passed as a parameter. An example to define such functions is given on the screen.

Note that higher-order functions are the ones that take other functions as parameters or return them as results.

## Anonymous Functions

An anonymous function:

- Is an alternative of named function definitions for small argument functions that get created by parameterization by functions
- An expression that evaluates to a function
- Is defined without giving it a name

**Example:**

$(y: Int) \Rightarrow y * y$

## Anonymous Functions

An anonymous function is an alternative of named function definitions for small argument functions that tend to get created by parameterization by functions. It is an expression that evaluates to a function. These are defined without giving them any name. An example of such a function is given on the screen.

## Objects

An object is a named instance with members like methods and fields.

**Example:**

```
object MyObject
val x = MyObject
```

A few of its uses are:

Contain independent methods and fields

**Example:**

```
println("Square root of 4: " + math.sqrt(4))
//Prints "Square root of 4: 2.0"
```

Create instances of classes

**Example:**

```
class ScalaTest {
  def SayHello() = println("Hello world!")
}
```

```
val greeter = new ScalaTest()
greeter.SayHello()
```



Scala has singleton objects instead of static members, which is a class with only one instance and can be created using the keyword object.

© Copyright 2015, Simplilearn. All rights reserved.

20

## Objects

An object is a named instance with members like methods and fields. In Scala, an object and its class have the same name.

For instance, in this example, the first line has the keyword “object”, which is being used to declare a new object. It is being followed by the name of the object. In the second line, a value “x” is being assigned to the object instance. Note that the members of objects are similar to those of classes.

Let's now discuss some of the uses of objects. One of the primary uses is to contain methods and fields that are independent from any environment. An example is “math”. This has the object name in the standard library containing multiple fields and methods depending only on arguments, if any, provided to them.

Objects are also used to create instances of classes. An example is given on the screen.

Note that in Scala, there are singleton objects instead of static members, which is a class with only one instance and can be created using the keyword object. In this way, Scala is more object-oriented than Java.

## Classes

A class:

- Allows defining rational numbers
- Is a blueprint of objects
- Allows creating objects using the keyword “new”



**Example:**

```
class ScalaClass1(a: Int, b: Int) {
    var x: Int = a
    var y: Int = b
    def addidtion(c: Int, d: Int) {
        x = x + c
        y = y + d
        println ("x output : " + x);
        println ("y output : " + y);
    }
}
```

© Copyright 2015, Simplilearn. All rights reserved.

21

## Classes

Scala does not include built-in rational numbers; however, you can define them using a class. A class acts like a blueprint of objects. Once a class is defined, objects can be created from this blueprint using the keyword “new”.

For instance, in this example, two variables, x and y, and a method are being defined. The method is not returning any value. The variables of a class are called fields of the class and methods are termed as class methods. The name of the class acts as a class constructor that can take multiple parameters.

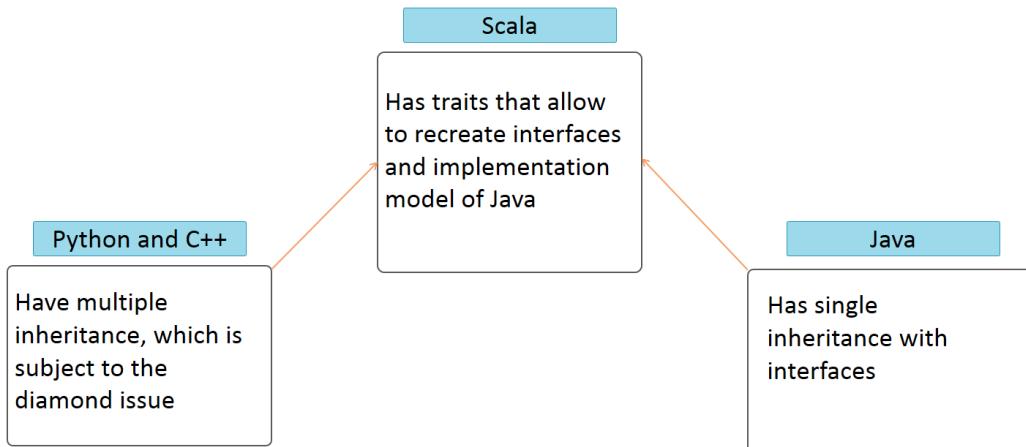
## Demo—Use Type Inference, Functions, Anonymous Function, and Class



This demo will show the steps to define and use Type Inference, Functions, Anonymous Function, and Class in Scala.

## Traits as Interfaces

Before you understand traits, let us first discuss the reason of its origin.



© Copyright 2015, Simplilearn. All rights reserved.

23

## Traits as Interfaces

Before you understand traits, let us first discuss the reason of its origin. In languages like Python and C++, you don't require interface as they have multiple inheritance, which is subject to the diamond issue. What will happen if you define a class inheriting two methods with the same type signature from two different classes? An interface is basically a set of properties and methods that an implementing class needs to have.

For avoiding the issues of multiple inheritance, the Java designers instead decided to have single inheritance with interfaces to make the system a little flexible.

Scala exists half-way between full multiple inheritance and Java's single inheritance with interface model, as it has traits. Traits allow you to recreate interfaces as Java does and also lets you go further. Using traits, you can recreate the interfaces and implementation model of Java.

## Traits—Example

**Example:**

```
trait Emp {  
    var name: String  
    var gender: Gender  
    def passExam(subject: String, body: String): Unit  
}
```

Implement the Student trait as:

```
class Student extends Emp {  
    var name: String  
    var gender: Gender  
    def passExam(subject: String, body: String): Unit = {  
        // some code  
    }  
}
```

## Traits—Example

Consider the given example to create traits in Scala. The code below shows how you can implement the Student trait.

**Collections:**

- Are containers of things
- Can be sequenced as linear sets of items
- May be:
  - One element, bounded to zero, or have an arbitrary number
  - Lazy or strict
  - Mutable or immutable



Mutable and immutable collections work better in different problems. In doubt, start with an immutable collection and change it later.

## Collections

Scala includes a rich set of collection library, which are containers of things. These can be sequenced as linear sets of items. They may be one element, bounded to zero, or have an arbitrary number. They can be lazy or strict too. Lazy collections contain elements that may not require memory until accessed, for example Ranges.

In addition, collections can be mutable or immutable. Immutable collections can contain mutable items.

You should note that mutable and immutable collections work better in different problems. In doubt, you should start with an immutable collection and change it later if required.

## Types of Collections

Different types of collections are listed below:

**Lists**

List[X] is a linked list of type X

**Example:**

```
// Define List of integers.  
val x = List(1,2,3,4)
```

**Sets**

A collection of pairwise various elements of the same type

**Example:**

```
// Define a set.  
var x = Set(1,3,5,7)
```

**Maps**

A collection of key or value pairs

**Example:**

```
// Define a map.  
val x = Map("one" -> 1, "two" -> 2, "three" -> 3)
```

© Copyright 2015, Simplilearn. All rights reserved.

26

## Types of Collections

The most commonly used collections type is lists. List[X] is a linked list of type X. An example is given on the screen.

Another type is sets, which is a collection of pairwise various elements of the same type. An example is displayed.

Scala maps represent a collection of key or value pairs. Consider the given example.

## Types of Collections (contd.)

Other types of collections are listed below:

**Tuples**

Can hold objects with different types

**Example:**

```
// Create a tuple of two elements
val x = (10, "Scala")
```

**Options**

Option[X] gives a container for one or zero element of a given type

**Example:**

```
// Define an option
val x:Option[Int] = Some(5)
```

**Iterator**

A way to access the elements of a collection one by one

**Example:**

```
object Test {def main(args: Array[String]) {
  val it = Iterator("a", "b", "c", "d")
  while (it.hasNext){println(it.next())}}}
```

© Copyright 2015, Simplilearn. All rights reserved.

27

**Types of Collections (contd.)**

Scala Tuples can hold objects with different types like an array or a list. An example of the same is given.

In Scala Options, Option[X] gives a container for one or zero element of a given type. Consider the given example. The most direct method to diagnose all the elements that are returned by an iterator is by using a while loop, as explained through the given example.

## Lists

All operations on lists can be expressed using the following methods:

- **head**: Returns the first element of a list
- **tail**: Returns a list with all elements except the first
- **isEmpty**: Returns true if the list is empty

```
object Test {
  def main(args: Array[String]) {
    val processing = "MapReduce" :: ("Hive" :: ("Spark" :: Nil))
    val query = Nil
    println("Head of processing : " + processing.head)
    println("Some List of person : " + processing.tail)
    println("Check if processing is empty : " +
      processing.isEmpty)
    println("Check if query is empty : " + query.isEmpty)
  }
}
```

Head of *processing* :MapReduce  
 Some List of *processing* : List(Hive, Spark)  
 Check if *processing* value is empty : false  
 Check if *query* value is empty : true

© Copyright 2015, Simplilearn. All rights reserved.

28

## Lists

All operations on lists can be expressed using the given methods. Head is a method that returns the first element of a list. The tail method returns a list with all elements except the first. The third method, isEmpty returns true if the list is empty otherwise false.

The example given on the screen shows the use of these methods. When this code is compiled and executed, it gives the output shown.

## Demo—Perform Operations on Lists



This demo will show the steps to use different operations on the list data structure.

## Maps

Retrieve any value based on the Scala map key, which is unique but not its values. Also called Hash tables, Maps are of two types:

- Immutable (default)
- Mutable

To use mutable maps

Import `scala.collection.mutable.Map` class explicitly

To use both maps

Refer the immutable Map as `Map`, but can refer the mutable set as `mutable.Map`



### Example:

```
// Map – key and value
val courses = Map("course1" -> "Hadoop-Bigdata", "course2" -> "Apache Spark")
```

## Maps

As discussed, maps represent a collection of key or value pairs. You can retrieve any value based on the Scala map key. These keys are unique, however, values need not be unique. Also called Hash tables, Maps are of two types, the immutable and the mutable.

Scala uses the immutable Maps by default. If you need to use the mutable maps, you would need to import `scala.collection.mutable.Map` class explicitly. To use both types of maps in the same map, you can continue referring to the immutable Map as `Map`, but you can refer the mutable set as `mutable.Map`.

The example given on the screen shows how to declare immutable maps.

## Maps—Operations

Maps operations are similar to those on sets and fall into the categories below:

Lookup Operations	Operations to turn maps to limited functions: <code>Contains</code> , <code>isDefinedAt</code> , <code>getOrElse</code> , <code>get</code> , <code>apply</code> ; Fundamental method: <code>def get(key): Option[Value]</code> ; Test operation for associated key: <code>m get key</code>
Transformations	Operations to produce a map through filtering and transforming bindings of a current map: <code>filterKeys</code> and <code>mapValues</code>
Subcollection Procedures	Operations to return keys and values of a map separately in different forms: <code>keys</code> , <code>keysIterator</code> , <code>keySet</code> , <code>values</code> , and <code>valuesIterator</code>
Removals	Operations to remove bindings from a map: <code>-</code> , <code>--</code>
Addition and Updates	Operations to add new or change existing bindings: <code>+</code> , <code>++</code>

## Maps—Operations

The basic operations on maps are similar to those on sets and fall into the given categories.

The lookup operations such as Contains, isDefinedAt, getOrElse, get, and apply, turn maps to limited functions from keys to values. The basic method is def get(key): Option[Value]. To test if the map includes an association for the given key, you can use the m get key operation. It returns the associated value in a Some if the association is included, otherwise, it returns None. The apply method returns the associated value with a given key directly, without wrapping it in an Option. An exception is raised if the key is not defined in the map.

Another maps operations is transformations. They produce a map through filtering and transforming bindings of a current map. These operations are filterKeys and mapValues.

Subcollection procedures return keys and values of a map separately in different forms. These operations are keys, keysIterator, keySet, values, and valuesIterator.

Removals remove bindings from a map.

However, the last type, additions and updates, allow to add new or change the existing bindings.

## Pattern Matching

Pattern matching:

- Allows to process messages
- Lets you match on any sort of data using the first-match policy
- Is the second-most widely used feature


**Example:**

```
object MatchTest1 extends App {
  def matchTest(x: Int): String = x match {
    case 1 => "Numberone"
    case 2 => "Numbertwo"
    case _ => "many"
  }
  println(matchTest(3))
}
```

© Copyright 2015, Simplilearn. All rights reserved.

32

## Pattern Matching

Pattern matching is supported by Scala for processing messages. This feature allows you to match on any sort of data using first-match policy. This built-in mechanism is the second most widely used feature in Scala, after closures and function values.

The example given on the screen depicts how to match against an integer value. The case statements block declares a function mapping integers to strings. The keyword match gives an easy way to apply a function to an object.

## Implicits



A method containing implicit parameters is applicable like normal methods, as the implicit label has no impact. If the method misses arguments, those are provided automatically. The actual arguments can be of two types:

1. All identifiers y accessible at the method call point without any prefix and denoting an implicit definition or parameter
2. All members of companion modules belonging to the implicit type of parameters, labeled implicit

## Implicits

A method containing implicit parameters are applicable to arguments like normal methods. This is because the implicit label has no impact. If the method misses arguments for its implicit parameters, those are provided automatically.

The actual arguments can be of two types.

First, these are all identifiers y that are accessible at the method call point without any prefix and denoting an implicit definition or parameter. And second, these are all members of companion modules belonging to the implicit type of parameters, labeled implicit.

## Implicits (contd.)

**Example:**

```

abstract class MyFirstAbstractClass[A[A] {
def addition(x: A, y: A): A
}
abstract class MySecondAbstractClass[A] extends MyFirstAbstractClass[A] {
def unit: A
}
object ImplicitTest extends App {
implicit object StringClass extends class MySecondAbstractClass[A[String] {
def addition(x: String, y: String): String = x concat y
def unit: String = ""
implicit object IntClass extends MySecondAbstractClass[Int] {
def addition(x: Int, y: Int): Int = x + y
def unit: Int = 0
}
def sum[A](xs: List[A])(implicit m: MySecondAbstractClass[A]): A =
if (xs.isEmpty) m.unit
else m. addition(x(xs.head, sum(xs.tail)))
println(addition(List(1, 2, 3)))
println(addition(List("a", "b", "c")))
}
}

```

**Implicits (contd.)**

In the example given on the screen, a method sum has been defined that is computing the sum of a list of elements through unit and add operations. Note that the implicit value cannot exist at the top. They need to be members of a template.

## Streams

The Stream class executes lazy lists in which elements are evaluated only when required.



### Example:

```
import scala.math.BigInt
object Test2 extends App {
  val fibs: Stream[BigInt] = BigInt(0) #:: BigInt(1) #:: fibs.zip(fibs.tail).map { n => n._1 + n._2 }
  fibs take 5 foreach println
}
// prints
//0
//1
//1
//2
//3
```



A Stream is a list with its tail as a lazy val. A value remains computed and is reused.

© Copyright 2015, Simplilearn. All rights reserved.

35

## Streams

The Stream class executes lazy lists in which elements are evaluated only when required. An example is given on the screen.

Note that in Scala, a Stream is a list with its tail as a lazy value. A value remains computed and is reused. In other words, the values are cached.

## Demo—Use Data Structures



This demo will show the steps to use different types of data structures in Scala.

**QUIZ  
1**

Which of the following is true about Scala?

- a. Multi-paradigm programming language
- b. Expressive type system with type inferencing
- c. Rich concurrency

**QUIZ  
2**

Which of following is an example of a tuple in Scala ?

- a. val x = Map("one" -> 1, "two" -> 2, "three" -> 3)
- b. var x = Set(1,3,5,7)
- c. val x = (10, "Scala")
- d. val x:Option[Int] = Some(5)



**QUIZ  
3**

Which of the following statement is true about the tail method in Scala?

- a. This method returns the first element of a list
- b. This method returns a list consisting of all elements except the first
- c. This method returns true if the list is empty otherwise false

**QUIZ  
4**

Which of the following statement is true about closure in Scala?

- a. A function, whose return value depends on the value of one or more variables declared outside this function
- b. A function, whose return value depends on the value of one or more variables declared inside this function
- c. A function, whose return value depends on the value of one or more variables declared as final

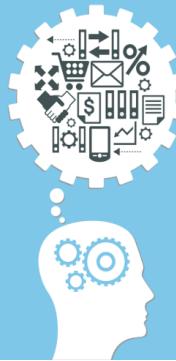


**ANSWERS:**

S.No.	Question	Answer & Explanation
1	Which of the following is true about Scala?	a., b., and c. Scala has all the features mentioned here.
2	Which of following is an example of a tuple in Scala?	c. Option c correctly defines a tuple in Scala.
3	Which of the following statement is true about the tail method in Scala?	b. The tail method returns a list consisting of all elements except the first.
4	Which of the following statement is true about closure in Scala?	a. A closure is a function, whose return value depends on the value of one or more variables declared outside this function.

## Summary

Let us summarize the topics covered in this lesson:



- Scala is a modern and multi-paradigm programming language, supporting the same data types as Java does.
- The basic literals used in Scala are: Integer, Floating Type, Character, String, Boolean, and Symbol.
- Scala provides various operators: Arithmetic, Relational and Logical, Bitwise, and Object Equality.
- Type inference is a mechanism that allows to omit certain annotations and return types of methods.
- A function is a “first-class” value.

© Copyright 2015, Simplilearn. All rights reserved.

46

## Summary

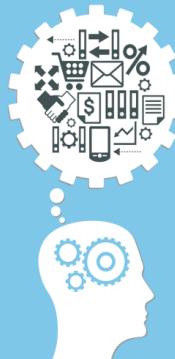
Let us summarize the topics covered in this lesson:

- Scala is a modern and multi-paradigm programming language, supporting the same data types as Java does.
- The basic literals used in Scala are: Integer, Floating Type, Character, String, Boolean, and Symbol.
- Scala provides various operators like Arithmetic, Relational and Logical, Bitwise, and Object Equality.
- Type inference is a mechanism that allows to omit certain annotations and return types of methods.

A function is a “first-class” value.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- A class is a blueprint of objects.
- Scala has traits that allow to recreate interfaces and implementation model of Java.
- Collections are containers of things.
- You can retrieve any value based on the Scala map key, which is unique but not its values.
- Pattern matching allows to process messages.
- A method containing implicit parameters are applicable like normal methods.
- The Stream class executes lazy lists in which elements are evaluated only when required.

© Copyright 2015, Simplilearn. All rights reserved.

47

## Summary (contd.)

- A class is a blueprint of objects.
- Scala has traits that allow to recreate interfaces and implementation model of Java.
- Collections are containers of things.
- You can retrieve any value based on the Scala map key, which is unique but not its values.
- Pattern matching allows to process messages.
- A method containing implicit parameters are applicable like normal methods.
- The Stream class executes lazy lists in which elements are evaluated only when required.

This concludes ‘Introduction to Programming in Scala.’

The next lesson is ‘Using RDD for Creating Applications in Spark.’

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 2 “Introduction to Programming in Scala” of the Apache Spark and Scala course. The next lesson is Using RDD for creating applications in Spark.

## Lesson 3—Using RDD for Creating Applications in Spark



## Objectives



After completing this lesson, you will be able to:



- Explain the features of RDDs
- Explain how to create RDDs
- Describe RDD operations and methods
- Discuss how to run a Spark project with SBT
- Explain RDD functions
- Describe how to write different codes in Scala

© Copyright 2015, Simplilearn. All rights reserved.

2

## Objectives

After completing this lesson, you will be able to:

- Explain the features of RDDs
- Explain how to create RDDs
- Describe RDD operations and methods
- Discuss how to run a Spark project with SBT
- Explain RDD functions, and
- Describe how to write different codes in Scala

An RDD:

- Acts as a handle for a collection of individual data partitions
- Can be reconstructed in case of lost partitions using lineage information
- Derives directly or indirectly from the class RDD
- Does not have restrictions regarding what data can be stored within partitions
- API:
  - Contains many useful operations, but many convenience functions are missing
  - Considers every data item as a single value

## RDDs API

An RDD acts like the workhorse of Spark, as it can be considered as a handle for a collection of individual data partitions. Actually, RDDs are more than that. In case of cluster installations, different data partitions can be on different nodes. RDDs, acting as handles, provide the capability to access all partitions. They also allow you to perform computations and transformations using the contained data.

In case the entire or a part of RDD is lost, they can be reconstructed by using lineage information. Lineage means the sequence of transformations that is used to produce the current RDD.

An RDD derives directly or indirectly from the class RDD. This class contains various methods that perform operations within the associated partitions on the data. It is an abstract class. Using an RDD means that you are actually using a concertized implementation of RDD.

Spark has recently become very popular to process big data. The reason is that it does not have restrictions regarding what data can be stored within partitions.

In addition, the RDD API contains various useful operations; however, various convenience functions are missing. The reason is that the Spark creators needed to keep the core API that could be common enough to handle arbitrary data-types.

Basically, an RDD API considers every data item as a single value. But, you would want to work with key-value pairs, which Spark provides through its extended capability to support PairRDDFunctions.

## Features of RDDs

All Spark tasks are expressed in terms of RDDs. It distributes the data included in RDDs automatically across your cluster and then parallelizes the operations. It has the following features:

### Immutable

- Helps to parallelize
- Allows to cache data for long

### Lazy Evaluated

- Defers evaluation
- Allows separating execution from evaluation
- Allows to recreate data on failure

### Cacheable

- Improves execution engine performance

### Type Inferred

- Allows to determine the type by operation
- Simplifies the representation for many transforms

## Features of RDDs

RDDs are simply distributed collections of elements. All Spark tasks are expressed in terms of RDDs such as creating new RDDs, transforming the existing RDDs, and calling operations on RDDs for computing results. Spark under the hood distributes the data included in RDDs automatically across your cluster. It then parallelizes the operations performed by you. The key features of RDDs are listed on the screen.

RDDs are immutable, which means that once they are created, they never change. This feature helps to parallelize and also allows to cache data spread between different cluster nodes.

RDDs are lazy evaluated too. Note that when you define an RDD, it does not contain any data. It is only when the data is referenced, the computation to create the data in an RDD is done. This feature defers evaluation and allows to separate execution from evaluation. Lazy transformations also allow to recreate data on failure.

In addition, RDDs are cacheable, which improves execution engine performance.

Also, the type inference feature is a part of compiler to determine the type by value. All transformations are free from side effects. Therefore, you can determine the type by operation. Each transformation includes a specific return type. With this feature, you can be less worried about the representation for many transforms.

## Creating RDDs

There are two ways to create RDDs:

**Parallelize an Existing Collection**

- Call the SparkContext's parallelize method on a collection in your driver program (a Scala Seq).

**Example:**

```
val data = Array(1, 2, 3, 4, 5)  
val distData = sc.parallelize(data)
```

**Reference an External Dataset**

- Reference a dataset in an external storage system such as a HDFS, HBase, shared file system, or any data source offering a Hadoop Input Format.

## Creating RDDs

To create RDDs, you can either parallelize an existing collection or reference an external dataset. To create parallelized collections, you would need to call the parallelize method of SparkContext on a collection that exists in your driver program. Consider the given example, which is creating a parallelized collection with numbers 1 to 5.

In addition, one can build these RDDs by referencing any Hadoop supported storage source, which includes a HDFS, HBase, your shared file system, or any data source offering a Hadoop InputFormat.

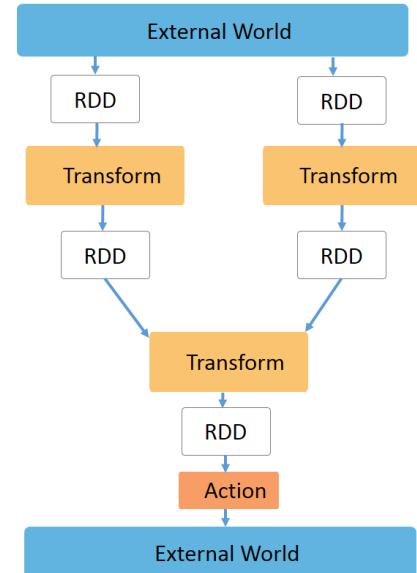
## Creating RDDs—Referencing an External Dataset

Spark supports:

Text Files

Sequence Files

Other Hadoop Input Formats



© Copyright 2015, Simplilearn. All rights reserved.

6

## Creating RDDs—Referencing an External Dataset

Let's talk about creating RDDs by referencing an external dataset in detail. Spark mainly supports text files, SequenceFiles, and other Hadoop InputFormats. The image on the screen shows how RDDs undergo transformations and interact with the external world.

## Referencing an External Dataset—Text Files

To create text file RDDs, use the **SparkContext's textFile** method to take an URI for a local file or `hdfs://`, `s3n://` and read it as lines collection.

**Example:**

```
scala> val distFile = sc.textFile("data.txt")
distFile: RDD[String] = MappedRDD@1d4cee08
```

Optionally, `distFile` can be acted on by dataset operations:

```
distFile.map(s => s.length).reduce((a, b) => a + b)
```



`SparkContext.wholeTextFiles` allows reading a directory with various small text files and returns each as pairs of filename and content.

## Referencing an External Dataset—Text Files

To create text file RDDs, use the `textFile` method of `SparkContext`. First, the `textFile` method considers an URI for a locally residing file or the given file. It then reads the file as lines collection. An example to use the method is given on the screen. Once the text file RDDs are created, dataset operations can act upon them. For instance, the sizes of all the lines can be added using the `reduce` and `map`. An example is displayed.

Note that you can also use the `SparkContext.wholeTextFiles` method, which allows reading a directory with various small text files. This method then returns each file as different pairs of filename and content. This is in contrary to the `textFile` method, which gives one record for each line in each file.

## Referencing an External Dataset—Text Files (contd.)

A few points about reading files with Spark are:

- When using a local path, the file must also be available at the same location on worker nodes.
- All file-based input methods of Spark, support to run on compressed files, directories, and wildcards;  
**Example:** `textFile("/my/directory")`
- To control the number of partitions, the `textFile` method also takes an optional second argument.



You cannot have fewer partitions than blocks.

## Referencing an External Dataset—Text Files (contd.)

You must remember a few points about reading files.

When you use a local path that exists on the file system, the related file needs to be available at the same location on worker nodes. You can use a network-mounted shared file system or copy the file to all workers.

All file-based input methods of Spark, which include `textFile`, support to run on compressed files, directories, and wildcards. An example is given.

To control the number of partitions of the file, the `textFile` method also takes a second argument, which is optional. For each block, Spark by default creates one partition. However, you can ask for more partitions. To do this, you need to pass a larger value.

You must remember that the number of partitions must be higher than blocks.

## Referencing an External Dataset—Sequence Files

For Sequence Files, use the **SparkContext's sequenceFile[K, V]** method.

**Points to Remember:**

- The parameters should be subclasses of the writable interface of Hadoop such as Text and IntWritable.
- You can specify native types for a few common writables;

**Example:** `sequenceFile[Int, String]`

## Referencing an External Dataset—Sequence Files

For SequenceFiles, use the SparkContext's `sequenceFile[K, V]` method. Here, K and V represent the types of key and values in the file respectively. For this method, note that the parameters K and V should be subclasses of the writable interface of Hadoop such as Text and IntWritable. You can also specify native types for a few common writables. In the given example, the method will automatically read Texts and IntWritable.

## Referencing an External Dataset—Other Hadoop Input Formats

For other Hadoop Input Formats, use the **SparkContext.hadoopRDD** method.

**Points to Remember:**

- The method takes an arbitrary JobConf and input format class, value class, and key class.
- `SparkContext.newAPIHadoopRDD` can also be used.



`SparkContext.objectFile` and `RDD.saveAsObjectFile` support to save an RDD in a simple format with serialized Java objects.

## Referencing an External Dataset—Other Hadoop Input Formats

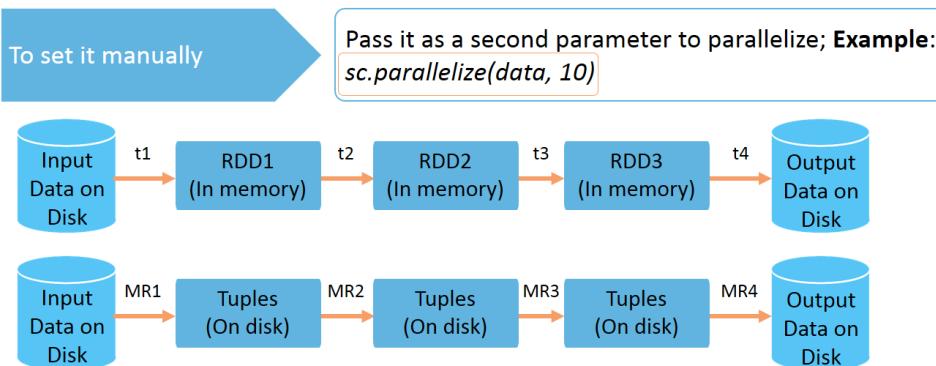
In case of other Hadoop InputFormats, you can use the **SparkContext.hadoopRDD** method. Note that this method considers an arbitrary input format class and JobConf, value class, and key class. You need to set them in the similar manner as done for a Hadoop job. In addition, the `SparkContext.newAPIHadoopRDD` method can also be used. This method is based on the new MapReduce API.

Note that `SparkContext.objectFile` and `RDD.saveAsObjectFile` support to save an RDD in an easy-to-understand format along with Java objects that are serialized. However, these methods are not as effective as others, but they provide an easy method to save an RDD.

## Creating RDDs—Important Points

A few important points to remember are:

- The distributed dataset (distData) can be operated on in parallel.
- Spark runs one task per cluster partition.
- Spark attempts setting the number of partitions automatically depending on your cluster.



© Copyright 2015, Simplilearn. All rights reserved.

11

## Creating RDDs—Important Points

You must know some important points related to the creation of RDDs. Once these are created, the distributed dataset, that is distData, can be operated on in parallel.

For parallel collections, the number of partitions for the dataset is an important parameter. Spark runs one task per cluster partition. However, in typical cases, you need 2-4 of them for each of your CPUs in your cluster.

Generally, Spark attempts setting the number of partitions automatically depending on your cluster. But, they can be set manually.

To do so, pass it as a second parameter to parallelize. An example is depicted on the screen.

The first image on the screen shows that data processing can be performed in Spark by reading data from the disk and loading it as an RDD in memory. After the first transformation, data can be stored on a hard disk that can again be loaded as an RDD to perform a transformation and then store the final result on the hard disk.

The second image shows the output of each transformation that will be stored only on the hard disk.

## RDD Operations

RDDs support two types of operations:

### Transformations

Allow to create a new dataset from an existing one  
**Example:** map

### Actions

Return a value to the driver program  
**Example:** reduce

Transformations	Actions
map(func)	take(N)
flatMap(func)	count()
filter(func)	collect()
groupByKey()	reduce(func)
reduceByKey(func)	takeOrdered(N)
mapValues(func)	top(N)

## RDD Operations

RDDs provide support to two different types of operations, which are transformations and actions.

Transformations allow creating a dataset from a present one. For example, map passes the elements of every dataset using a function and gives an RDD with results.

However, actions allow returning a value to the driver program. Actions return values once a computation on the dataset is run. For example, reduce aggregates all RDD elements using a function. It then provides the final result to the driver program.

The table shown on the screen shows some more transformations and actions.

## RDD Operations—Transformations

All Spark transformations are lazy. Note that:

- Transformations are computed only when an action needs a result to be returned to the driver program.
- Each transformed RDD, by default, may be recomputed every time when an action is run on it. To persist an RDD in memory, use the cache or persist method.

## RDD Operations—Transformations

All Spark transformations are lazy, which means that they do not compute the results immediately. These are computed only in case when an action requires that a result should be returned to the driver program. Due to this feature, Spark is capable of running more proficiently. As an example, you can see that if you create a dataset using the map feature, it is used in a reduce. It returns the result to the driver of only the reduce.

As the default, each transformed RDD may get recomputed in every instance when you run an action on it. For persisting an RDD in memory, you can use the cache or persist method. By doing so, Spark allows faster access at the time of next time query by keeping the elements around on the cluster.

## Features of RDD Persistence

The features of RDD persistence are listed below:

- Every node stores any of its partitions; computed in memory and reused in other actions on that dataset.
- The cache is fault-tolerant; any lost RDD partition will be automatically recomputed using the original transformations.
- To store every persisted RDD, use a different storage level; levels are set by passing a StorageLevel object to persist().

## Features of RDD Persistence

RDD persistence is considered to one of the most important Spark traits. As of its feature, every node stores any of its partitions that is computed within memory. It is reused in all other actions that are on the dataset or any other derived dataset. Due to this capability, future actions are much faster.

Therefore, caching helps in iterative algorithms and interactive use.

The cache is fault-tolerant. This means that any lost RDD partition will be automatically recomputed using the original transformations.

To store every persisted RDD, you can use a different storage level. For instance, for persisting the disk dataset, it can be persisted in memory, however as serialized Java objects for saving space, replicating it among nodes, and storing it off-heap in Tachyon. To set these levels, you can pass a StorageLevel object to the persist() method.

## Storage Levels of RDD Persistence

The table below lists and explains the storage levels of RDD persistence:

Storage Level	Explanation
MEMORY_AND_DISK	Allows to store RDD as deserialized Java objects; stores the partitions not fitting on disk and reads when required
MEMORY_ONLY	Allows to store RDD as serialized Java objects; a few partitions will not be cached and will be recomputed on the go if the RDD does not fit in memory
MEMORY_ONLY_SER	Allows to store RDD as serialized Java objects; permits more space efficiency, especially in case of a fast serializer
MEMORY_AND_DISK_SER	Is similar to MEMORY_ONLY_SER, except spilling partitions not fitting in memory to disk
DISK_ONLY	Allow to store RDD partitions only on disk
MEMORY_ONLY_2, MEMORY_AND_DISK_2, and others	Are similar to above, except replicating every partition on two cluster nodes.
OFF_HEAP (experimental)	Allow to store RDD in serialized format in Tachyon; reduces garbage collection overhead as compared to MEMORY_ONLY_SER; does not lead to losing the in-memory cache

© Copyright 2015, Simplilearn. All rights reserved.

15

## Storage Levels of RDD Persistence

As discussed, using the persist() method, you are allowed to specify the desired storage level. You can use this method for assigning any other storage level other than the default one. The table given on the screen lists and explains all the storage levels of RDD persistence.

## Choosing the Correct RDD Persistence Storage Level

Choose the applicable storage level, as there are trade-offs between memory usage and CPU efficiency:



- |  |  |
|--|--|
| RDDs fit with the default storage level.                           | Leave them that way.   |
| Leaving them is not possible.                                      | Use MEMORY_ONLY_SER and select a fast serialization library. |
| Fast fault recovery is required.                                   | Use the replicated storage levels.                           |
| Environments have high amounts of memory or multiple applications. | Use the experimental OFF_HEAP storage level.                 |

## Choosing the Correct RDD Persistence Storage Level

Storage levels provide trade-offs between CPU efficiency and memory usage. Therefore, you must choose the one that fits to your needs. In case your RDDs are fitting well with the default storage level, it is recommended that you leave them in the same manner. The reason is that it provides the maximum CPU efficiency and hence, allows to run the operations on RDDs as fast as possible.

However, if you do not want to use the default one that leaving them is not possible, try the one given on the screen and select a serialization library that is fast. This will help in making the objects comparatively more efficient in terms of space, however practically fast in accessing. You should not perform the disk spill except if functions that have computed the datasets filter a large amount of the data or are expensive. If you do so, the process of recomputing a partition can become as efficient as if you are reading it from a disk.

In case fast fault recovery is required, you should use the replicated storage levels. Although all storage levels provide fault tolerance, using replicated storage levels allow continuing to run tasks on the RDD while you don't have to wait for recomputing a partition that is lost.

In case environments have various applications or high memory amounts, you can use the experimental OFF\_HEAP storage level. Using it has many advantages like multiple executors sharing the same memory pool in Tachyon and reduced costs of garbage collection. In addition, the data that got cached does not get lost in case separate executors get crashed.

## Invoking the Spark Shell

Spark Shell:

- Provides an easy way to learn the API
- Allows to analyze data interactively
- Is available in Scala or Python

**How to Run?**

1. Go to \$SPARK\_HOME.
2. Run the following in the Spark directory:
  - **Scala:** ./bin/spark-shell
  - **Python:** ./bin/pyspark

© Copyright 2015, Simplilearn. All rights reserved.

17

## Invoking the Spark Shell

The Spark shell provides an easy way to learn the API. In addition, it is a powerful tool that allows to analyze data interactively. It is available in Scala or Python.

To run the same, you need to first access the Spark Home directory and then run the given commands applicable to Scala and Python.

## Importing Spark Classes

Import some Spark classes into your program by executing the codes below:

Scala

```
import org.apache.spark.SparkContext  
import org.apache.spark.SparkConf
```

Java

```
import org.apache.spark.api.java.JavaSparkContext  
import org.apache.spark.api.java.JavaRDD  
import org.apache.spark.SparkConf
```

Python

```
from pyspark import SparkContext, SparkConf
```

## Importing Spark Classes

Once the shell is invoked, you need to create import some Spark classes into your program by executing the codes given on the screen. Note the different codes applicable to Scala, Java, and Python

## Creating the SparkContext

Create a SparkContext instance to interact with Spark and distribute jobs by executing the codes below:

Scala

```
val conf = new SparkConf().setAppName(appName).setMaster(master)  
new SparkContext(conf)
```

Java

```
SparkConf conf = new SparkConf().setAppName(appName).setMaster(master);  
JavaSparkContext sc = new JavaSparkContext(conf)
```

Python

```
conf = SparkConf().setAppName(appName).setMaster(master)  
sc = SparkContext(conf=conf)
```

## Creating the SparkContent

Next, you need to create a SparkContext instance to interact with Spark and distribute jobs. A SparkContext class provides a connection to a Spark cluster and hence offers the entry point to interact with Spark. To create it, you can execute the given codes applicable to Scala, Java, and Python. These codes set the application name and Spark Master details.

## Loading a File in Shell

**Load the README file into Spark:**

```
scala>valtextFile = sc.textFile("README.md")
textFile: spark.RDD[String] = spark.MappedRDD@2ee9b6e3
```

## Loading a File in Shell

Let's now create a new RDD from the README file text in the Spark source directory. For this, you can execute the given code.

## Performing Some Basic Operations on Files in Spark Shell RDDs

Here are a few actions that can be performed on files in Spark shell RDDs:

Get count from a file

**Example:**

```
scala> textFile.count() // Number of items in this RDD  
res0: Long = 126
```

Get the first element from a file

**Example:**

```
scala> textFile.first() // First item in this RDD  
res1: String = # Apache Spark
```

## Performing Some Basic Operations on Files in Spark Shell RDDs

Let's now perform a few actions on files in the Spark shell RDDs. The first action is to get count from a file. Consider the given example to do the same.

The next action is to get the first element from a file. An example to do the same is given on the screen.

## Packaging a Spark Project with SBT

SBT is supported for day-to-day development over Maven (officially recommended) for faster iterative compilation.

1

Create the SBT build.

**Example:**

```
build/sbt -Pyarn -Phadoop-2.3 assembly
```



2

Test with SBT.

**Example:**

```
build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver assembly
```

```
build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver test
```

## Packaging a Spark Project with SBT

To package Spark, Maven is the official recommendation. It is the build of reference. However, for day-to-day development, SBT is supported. The reason is that it provides much faster iterative compilation. It is used by more advanced developers. Its build is derived from the POM files of Maven.

To package a Spark project with SBT, you need to create it first. An example of such code is given on the screen. This will set the same Maven profiles and variables to control the build.

Next, you need to test with SBT. Note that a few tests need Spark to be packaged first. So you should always run the build/sbt assembly at the first time. The example given on the screen shows the same.

## Running a Spark Project with SBT

Use the following codes:

Specific test suite

```
build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver "test-only  
org.apache.spark.repl.ReplSuite"
```

Test suites of a  
specific sub  
project

```
build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver core/test
```



Cache the RDD using the same context and reuse it for other jobs. Use an external caching solutions such as Tachyon.

## Running a Spark Project with SBT

Let's now view how to run a Spark project with SBT. To run only a specific test suite, you can use the code given on the screen. Similarly, to run test suites of a specific sub project, use the given code.

You should cache the RDD using the same context and reuse it for other jobs. In this manner, you only cache once and use it many times. In addition, you should use an external caching solutions such as Tachyon.

## Demo—Build a Scala Project

This demo will show the steps to build a Scala project with the SBT tool.

© Copyright 2015, Simplilearn. All rights reserved.

24

## Demo—Build a Scala Project

In this demo, you will learn how to build a Scala project using the SBT tool.

First, you need to create a directory structure as “src/main/scala” to keep all of your Scala source code files.

Let's now create a file “SampleApp.scala” in this folder. In this file, we will write our first application to read a local file and count the number of lines in which characters “a” and “b” have occurred.

As shown in the code, we have imported SparkContext, SparkConf and rest of the classes from the same package. This class has the main method in which we are reading the “README.md” file using the textFile method of the SparkContext object. After that, we are using the filter method to read each line and check the occurrence of characters a and b. We are caching this count in the memory by using the cache method.

To build a Scala project, we need to create a build file “spark.sbt” to provide dependency details. We will provide all the dependent jar details in the libraryDependencies section.

Once we have the source code and sbt build file written, we can build this project form the command prompt by executing the “sbt package” command. This command will download all the dependent jars files and compile the source files to create a jar file that can be used for running the application.

After successfully compiling the code, our jar file is available in the “target\scala-2.10” directory

## Demo—Build a Spark Java Project



This demo will show the steps to build a Spark Java project with Maven.

© Copyright 2015, Simplilearn. All rights reserved.

25

## Demo—Build a Spark Java Project

In this demo, you will learn how to build a Spark Java project with Maven.

First, you need to create a directory structure as “src/main/java” to keep all your Scala source code files. Let’s then keep some Java source files in this folder.

To build a Java project, we need to create a build file “pom.xml” to provide the dependency details. We will provide all the dependent jar details in the dependency section. Here, in this file we are mentioning dependent jar file as Hadoop, Hive, hbase, kafka, and others

Once we have the source code and pom.xml build file written, we can build this project from the command prompt by executing the “mvn package” command. This command will download all the dependent jars files and compile the source files to create a jar file that can be used for running the application.

After successfully compiling the code, our jar file is available in the “target” directory.

## Shared Variables—Broadcast

Broadcast variables:

- Allow to keep a read-only variable cached on every machine
- Can be used for providing every node a copy of a large input dataset efficiently
- Can be distributed using efficient broadcast algorithms
- Allow to broadcast the common data required by tasks within every stage
- Are created by calling `SparkContext.broadcast(v)`
- Should be used in place of actual data structure in any functions running on the cluster


**Example:**

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)
scala> broadcastVar.value
res0: Array[Int] = Array(1, 2, 3)
```

© Copyright 2015, Simplilearn. All rights reserved.

26

## Shared Variables—Broadcast

Let us now talk about shared broadcast variables, which allow you to save a read-only variable that is cached on every machine. In general situations, its copy is shipped with tasks.

The variables are capable of providing each node a large input dataset copy efficiently.

These variables can be distributed by the use of effective broadcast algorithms, which reduces the cost of communication.

The Spark actions are performed via a stages set that are disconnected by distributed “shuffle” operations. However, broadcast variables allow to broadcast the data that is common and is required by tasks during every stage. In this way, the data that is broadcasted is cached in the serialized form and deserialized before every task is run.

They are created by calling the `SparkContext.broadcast(v)` method from the variable v. They provide a wrapper around v.

In addition, these variables should be used in place of v in all functions that are running on a cluster. This ensures that v doesn't get shipped more than a time to the nodes. This object is recommended not to be changed once it is broadcasted for making sure that all of the nodes receive the same broadcast value.

The example given on the screen shows the creation of broadcast variables.

## Shared Variables—Accumulators

## Accumulators:

- Are added only to through an associative operation
- Can be used to implement counters or sums
- Are numeric types natively
- Are displayed in the UI of Spark if created with a name
- Are created by calling `SparkContext.accumulator(v)`
- Can be read by only the driver program using the `value` method

**Example:**

```
scala> val accum = sc.accumulator(0, AccumulatorExample")
scala> sc.parallelize(Array(5, 5, 5, 5)).foreach(x => accum += x)
scala> accum.value
res2: Int = 20
```

© Copyright 2015, Simplilearn. All rights reserved.

27

## Shared Variables—Accumulators

Another type of shared variables is accumulators, which are added only to through an associative operation. Therefore, they are efficiently supported in parallel. They are usable to implement counters or sums.

Natively, Spark supports numeric types accumulators; however, you can add support for new types. They are displayed in the UI of Spark if created with a name. Due to this, it is useful to understand the running stages progress.

To create an accumulator, you need to use the `SparkContext.accumulator(v)` method. Then, the tasks running on the cluster can add to it using the `add` method or the plus equal to operator. Note that only the driver program can read the values of accumulators using the `value` method.

The code given on the screen shows an accumulator that is being used to add an array elements.

## Writing a Scala Application

**Example:**

```
/* FirstSimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object FirstSimpleApp {
    def main(args: Array[String]) {
        val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
        val conf = new SparkConf().setAppName("Simple Application")
        val sc = new SparkContext(conf)
        val logData = sc.textFile(logFile, 2).cache()
        val numAs = logData.filter(line => line.contains("a")).count()
        val numBs = logData.filter(line => line.contains("b")).count()
        println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))  }}}
```

## Writing a Scala Application

Now that you have learned the basic concepts of RDD and caching levels, let's view a simple Scala application. In this example, the text file README.MD is being loaded and cached in the memory. Post this, the number of lines is being counted containing characters 'a' and 'b'.

## Demo—Run a Scala Application



This demo will show the steps to run a Scala application using a jar file.

© Copyright 2015, Simplilearn. All rights reserved.

29

## Demo—Run a Scala Application

In this demo, you will learn how to run a Scala application using a jar file.

You need to go to the bin directory of your Spark installation path to launch “spark\_submit”. To run the application, type the command as shown on the screen.

From the output displayed on the screen, you can see that there were 60 lines in which character “a” has been found while “b” has been found in 29 lines.

## Demo—Write a Scala Application Reading the Hadoop Data

This demo will show the steps to write a Scala application that reads the Hadoop data.

## Demo—Write a Scala Application Reading the Hadoop Data

In this demo, you will learn how to write a Scala application that reads the Hadoop data.

In this application, we are going to read a file stored in HDFS and count its length and store that length in the memory. After that, we want to iterate half of that count to see how much time it takes to iterate over it.

In the code shown on screen, after importing classes from the Spark package, we are creating the `SparkContext` object and calling the `textFile` method to read the HDFS file. This file is passed as a run time parameter. By using `map()`, we are calculating the length of the file and persisting that value in the memory. Finally, we are iterating half of the length value to see how much time in millisecond it takes.

## Demo—Run a Scala Application Reading the Hadoop Data

This demo will show the steps to run a Scala application that reads the Hadoop data.

## Demo—Run a Scala Application Reading the Hadoop Data

In this demo, you will learn how to run a Scala application that reads the Hadoop data.

First, we will copy the “input.txt” file into HDFS using the Hadoop put command.

We can run this program as we ran our first scala application by using the `spark_submit` script file available in the bin directory of the Spark installation path. We need to type the command as shown in the screen. Note this program expect to pass the HDFS file name as a run time parameter that we will mention after the jar name.

From the output displayed on the screen, you can see it has taken 120 ms for doing 9 iterations.

## Scala RDD Extensions

In Spark, there are four extensions to the RDD API:

DoubleRDDFunctions	Contain methods to aggregate numeric values that become available when a RDD data items can be implicitly converted to the Scala data-type double
PairRDDFunctions	Include methods that become available when the data items have a two-component tuple structure
OrderedRDDFunctions	Include methods that become available if the data items are two-component tuples
SequenceFileRDDFunctions	Include methods that let you to create Hadoop sequences from RDDs; data items must be two component key-value tuples

## Scala RDD Extensions

We discussed that Spark has PairRDDFunctions as its extensions. There are three more extensions, DoubleRDDFunctions, OrderedRDDFunctions, and SequenceFileRDDFunctions.

The DoubleRDDFunctions extension contains methods to aggregate numeric values that become available when a RDD data items can be implicitly converted to the Scala data-type double.

The PairRDDFunctions extension includes methods that become available when the data items have a two-component tuple structure. Spark interprets the first item as the key and the second one as the associated value.

The OrderedRDDFunctions extension includes methods that become available if the data items are two-component tuples. Here, the key is sortable implicitly.

The SequenceFileRDDFunctions extension includes methods that let you to create Hadoop sequences from RDDs. Data items must be two component key-value tuples, which are needed by PairRDDFunctions.

## DoubleRDD Methods

The table below lists and explains DoubleRDD methods:

Method	Explanation
def mean(): Double	Calculates the mean of the RDD's elements
def meanApprox(timeout: Long, confidence: Double = 0.95): PartialResult[BoundedDouble]	Returns the mean within a timeout
def sampleStdev(): Double	Calculates the sample standard deviation of RDD's elements
def stats(): StatCounter	Returns a StatCounter object that captures the mean, variance, and count of the RDD's elements
def stdev(): Double	Calculates the standard deviation of RDD's elements
def sum(): Double	Adds up the elements in an RDD
def sumApprox(timeout: Long, confidence: Double = 0.95): PartialResult[BoundedDouble]	Returns the sum within a timeout
def variance(): Double	Computes the variance of RDD's elements

© Copyright 2015, Simplilearn. All rights reserved.

33

## DoubleRDD Methods

Let's now learn about DoubleRDD functions or methods. These are listed and explained in the given table.

## PairRDD Methods—Join

If you consider the simple join operator, it is an inner join. The methods to join PairRDDFunctions are listed below:

**reduceByKey()**

Can aggregate data separately for each key

**Example:**

```
val lines = sc.textFile("data.txt")
val pairs = lines.map(s => (s, 1))
val counts = pairs.reduceByKey((a, b) => a + b)
```

**join()**

Can merge two RDDs

**Example:**

```
rdd.join(otherRDD)
```

**PairRDD Methods—Join**

If you consider the simple join operator, it is an inner join. The keys that are only present in both pair RDDs come as output. However, in case of various values for the same key, the output RDD has an entry for each possible pair of values. For PairRDD functions, you can use reduceByKey() that can aggregate data separately for each key. An example to use it is given on the screen. This code uses the reduceByKey operation on key-value pairs to count how many times each line of text occurs in a file.

PairRDD also have the join() method that can merge two RDDs by grouping elements with the same key. An example to use the same is displayed.

## PairRDD Methods—Others

The table below lists and explains other PairRDD methods:

Method	Explanation
def cogroup[W1, W2]	For every key a in this or other1 or other2, returns a resultant RDD containing a tuple with the list of values for that key in this, other1 and other2
def collectAsMap(): Map[K, V]	Returns the key-value pairs in this RDD to the master as a map
def combineByKey[C]	Hash-partitions the resulting RDD using the default parallelism level
def countByKey(): Map[K, Long]	Counts the number of elements for each key and returns the result to the master as a map
def groupByKey(): RDD[(K, Seq[V])]	Groups the values for each key in the RDD into a single sequence
def groupByKey(partitioner: Partitioner): RDD[(K, Seq[V])]	Groups the values for each key in the RDD into a single sequence
def reduceByKeyLocally(func: (V, V) ⇒ V): Map[K, V]	Merges the values for every key using an associative reduce function, but returns the results immediately to the master as a map

© Copyright 2015, Simplilearn. All rights reserved.

35

## PairRDD Methods—Others

The table on the screen lists and explains other PairRDD methods.

## Java PairRDD Methods

The table below lists and explains Java PairRDD methods:

Method	Explanation
def saveAsTextFile(path: String): Unit	Saves an RDD as a text file using string representations of elements
def sortByKey(comp: Comparator[K], ascending: Boolean): JavaPairRDD[K, V]	Sorts an RDD by key so that each partition contains a sorted range of the elements; calling collect or save on the resulting RDD returns or outputs an ordered list of records
def sortByKey(comp: Comparator[K]): JavaPairRDD[K, V]	<i>Same as above</i>
def sortByKey(ascending: Boolean): JavaPairRDD[K, V]	<i>Same as above</i>

© Copyright 2015, Simplilearn. All rights reserved.

36

## Java PairRDD Methods

Let's now learn about Java PairRDD functions or methods. These are listed and explained through the given table.

## Java PairRDD Methods (contd.)

A few more Java PairRDD methods are:

Method	Explanation
def sortByKey(): JavaPairRDD[K, V]	Sorts an RDD by key so that each partition contains a sorted range of the elements in ascending order; calling collect or save on the resulting RDD returns or outputs an ordered list of records
def splits: List[Split]	Sets of partitions in an RDD
def take(num: Int): List[(K, V)]	Merges the values for every key using an associative reduce function, but returns the results immediately to the master as a map
def takeSample(withReplacement: Boolean, num: Int, seed: Int): List[(K, V)]	Takes the first num elements of a RDD; currently scans the partitions one by one, so it will be slow if a lot of partitions are required; use collect() to get the whole in this case
def union(other: JavaPairRDD[K, V]): JavaPairRDD[K, V]	Returns the union of two RDDs; all identical elements appear multiple times (use .distinct())

## Java PairRDD Methods (contd.)

A few more Java PairRDD methods are listed and explained too.

## General RDD Methods

The table below lists and explains general RDD methods:

Method	Explanation
<code>def ++(other: RDD[T]): RDD[T]</code>	Returns the union of two RDDs; all identical elements appear multiple times (use <code>.distinct()</code> )
<code>def aggregate[U](zeroValue: U)(seqOp: (U, T) ⇒ U, combOp: (U, U) ⇒ U)(implicit arg0: ClassManifest[U]): U</code>	Aggregates each partition's elements and then the results for all the partitions, using a neutral "zero value" and given combine functions; can return a different result type, U, than the type of this RDD, T; hence, one operation is required for merging a T into a U and one operation is required for merging two U's, as in <code>scala.TraversableOnce</code> ; both of them functions can modify and return their first argument instead of creating a new U to avoid memory allocation
<code>def cache(): RDD[T]</code>	Persists an RDD with the default storage level (MEMORY_ONLY)

## General RDD Methods

The table shows and explains general RDD methods.

## General RDD Methods (contd.)

A few other general RDD methods are:

Method	Explanation
def cartesian[U](other: RDD[U])(implicit arg0: ClassManifest[U]): RDD[(T, U)]	Returns the cartesian product of two RDDs; the another RDD is of all pairs of elements (a, b), where a is in this and b is in other
def collect(): Array[T]	Returns an array containing all the elements in an RDD
def context: SparkContext	Refers to the SparkContext that this RDD was created on
def count(): Long	Returns the number of elements in an RDD
def countApprox(timeout: Long, confidence: Double = 0.95): PartialResult[BoundedDouble]	Returns a potentially incomplete result within a timeout, even if not all tasks have finished

## General RDD Methods (contd.)

A few more general RDD methods are also listed.

## Java RDD Methods

The table below lists and explains Java RDD methods:

Method	Explanation
def aggregate[U](zeroValue: U)(seqOp: Function2[U, T, U], combOp: Function2[U, U, U]): U	Aggregates each partition's elements and then the results for all the partitions, using a neutral "zero value" and given combine functions; can return a different result type, U, than the type of this RDD, T; hence, one operation is required for merging a T into a U and one operation is required for merging two U's, as in scala.TraversableOnce; both of them functions can modify and return their first argument instead of creating a new U to avoid memory allocation
def cache(): JavaRDD[T]	Persists an RDD with the default storage level (MEMORY_ONLY)
def cartesian[U](other: spark.api.java.JavaRDDLike[U, _]): JavaPairRDD[T, U]	Returns the cartesian product of two RDDs; the another RDD is of all pairs of elements (a, b) where a is in this and b is in other

## Java RDD Methods

Let's now talk about the Java RDD methods. These are shown in the table on the screen.

## Java RDD Methods (contd.)

A few more Java RDD methods are:

Method	Explanation
def checkpoint(): Unit	Marks an RDD for checkpointing; is saved to a file inside the checkpoint directory set with SparkContext.setCheckpointDir() and all references to its parent RDDs are removed; must be called before any job has been executed on this RDD; as recommended, this RDD is persisted in memory, otherwise saving it on a file will require recomputation
coalesce(numPartitions: Int): JavaRDD[T]	Returns a new RDD reduced into numPartitions partitions
def collect(): List[T]	Returns an array containing all the elements in an RDD
def context: SparkContext	Refers to the SparkContext that this RDD was created on
def count(): Long	Returns the number of elements in an RDD

© Copyright 2015, Simplilearn. All rights reserved.

41

## Java RDD Methods (contd.)

A few more Java RDD methods are also listed and explained.

## Common Java RDD Methods

The table below lists and explains common Java RDD methods:

Method	Explanation
def distinct(numPartitions: Int): JavaRDD[T]	Returns a new RDD with distinct elements in an RDD
def distinct(): JavaRDD[T]	Returns a new RDD with distinct elements in an RDD
filter(f: Function[T, Boolean]): JavaRDD[T]	Returns a new RDD with only the elements satisfying a predicate
def flatMap[K2, V2](f: PairFlatMapFunction[T, K2, V2]): JavaPairRDD[K2, V2]	Returns a new RDD by first applying a function to all elements of an RDD and then flattening the results
def foreach(f: VoidFunction[T]): Unit	Applies a function f to all the elements of an RDD
def flatMap(f: DoubleFlatMapFunction[T]): JavaDoubleRDD	Returns a new RDD by first applying a function to all elements of an RDD and then flattening the results

## Common Java RDD Methods

Now, we will discuss common Java RDD methods. These are listed and explained through the table displayed on the screen.

## Spark Java Function Classes

Below is a table that lists the function classes (abstract method: call ()) used by the Java API:

Class	Function Type
Function<T, R>	T => R
DoubleFunction<T>	T => Double
PairFunction<T, K, V>	T => Tuple2<K, V>
FlatMapFunction<T, R>	T => Iterable<R>
DoubleFlatMapFunction<T>	T => Iterable<Double>
PairFlatMapFunction<T, K, V>	T => Iterable<Tuple2<K, V>>
Function2<T1, T2, R>	T1, T2 => R (contains two arguments)

## Spark Java Function Classes

The table on the screen lists the function classes that are used by the Java API. Every class has a single abstract method, which is call() and must be implemented.

## Method for Combining JavaPairRDD Functions

The **def aggregate[U](zeroValue: U)(seqOp: Function2[U, (K, V), U], combOp: Function2[U, U, U]): U** method:

- Aggregates the elements of each partition
- Can return a different result type, U, than the type of this RDD, T
- Requires one operation for merging a T into an U and one operation for merging two U's, `scala.TraversableOnce`



## Method for Combining JavaPairRDD Functions

To combine JavaPairRDD functions, you can use the given method that aggregates the elements of each partition. It then aggregates the results for all the partitions using neutral "zero value" and combine functions.

Using this method, you can return a different result type, U, than the type of this RDD, T. Therefore, you would need two operations, one operation for merging a T into an U and one operation for merging two U's. The other one is listed on the screen. Both of them can modify and return their first argument and not create a new U for avoiding the problem of memory allocation.

## Transformations in RDD

Below are the examples of transformations in RDD:

**Sample**

Returns a random sample subset RDD of the input RDD

**Example:**

```
scala> parallel.sample(true,.2).count
```

**Map**

Passes each element through func

**Example:**

```
scala> val rows = babyNames.map(line =>
line.split(","))
```

**Filter**

Creates a new RDD by passing in a func used to filter the results

**Example:**

```
val file = sc.textFile("catalina.out")
val errors = file.filter(line =>
line.contains("ERROR"))
```

**groupByKey**

Returns a dataset of (K, Iterable<V>) pairs

**Example:**

```
scala> val namesToCounties = rows.map(name => (name(1),name(2)))
scala> namesToCounties.groupByKey.collect
```

© Copyright 2015, Simplilearn. All rights reserved.

45

## Transformations in RDD

Transformations in RDD are sample, map, filter, and **groupByKey**.

A Sample Returns a random sample subset RDD of the input RDD. An example is given.

A Map passes each element through function, as depicted on the screen.

A filter on the other hand creates a new RDD by passing in a function used to filter the results. Consider the given example.

However, groupByKey returns a dataset of given pairs. An example is shown on the screen.

## Other Methods

The table below lists and explains some other methods:

Method	Explanation
reduceByKey(func, [numTasks])	Operates on (K,V) pairs of course, where the func must be of type (V,V) => V
sortByKey([ascending], [numTasks])	Sorts the (K,V) pair by K
flatMap(func)	Similar to map, however each input item can be mapped to 0 or more output items
join(otherDataset, [numTasks])	Joins two datasets
def cogroup[W1, W2](other1: RDD[(K, W1)], other2: RDD[(K, W2)], numSplits: Int): RDD[(K, (Seq[V], Seq[W1], Seq[W2]))]	Returns a resulting RDD containing a tuple with the list of values for that key in this, other1 and other2, for each key k in this or other1 or other2
def mapValues[U](f: (V) => U): RDD[(K, U)]	Passes each value in the key-value pair RDD through a map function without changing the keys; also retains the original RDD's partitioning

## Other Methods

Other methods used in RDD are listed and explained through the table on the screen.

## Actions in RDD

The basic actions and their syntax are given in the table below:

Action	Syntax
Collect	wordCounts.collect()
Reduce	textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
Count	val numAs = logData.filter(line => line.contains("a")).count()
Save	accessLogs.saveAsTextFile(outputDirectory)
lookup	lookup(key: K): Seq[V]

## Actions in RDD

Let's now talk about actions in RDD. These are given in the table shown on the screen with their syntax.

## Key-Value Pair RDD in Scala



Some special operations are only available on RDDs of key-value pairs, which are available automatically on RDDs containing Tuple2 objects in the PairRDDFunctions class that automatically wraps around an RDD of tuples.

**Example:** distributed “shuffle” operations



**Example:**

```
val lines = sc.textFile("data.txt")
val pairs = lines.map(s => (s, 1))
val counts = pairs.reduceByKey((a, b) => a + b)
```

## Key-Value Pair RDD in Scala

Some special operations are only available on RDDs of key-value pairs. These operations are available automatically on RDDs with Tuple2 objects in the PairRDDFunctions class that automatically wraps around an RDD of tuples. A common example is shuffle operations like aggregating or grouping the elements by a key.

In the given example, the reduceByKey operation on key-value pairs is being used for counting the number of times each line of text occurs in a file.

## Key-Value Pair RDD in Java

In Java, these are represented using the `scala.Tuple2` class. Call `new Tuple2(a, b)` to create a tuple, and access its fields later with `tuple._1()` and `tuple._2()`. RDDs of key-value pairs are represented by the `JavaPairRDD` class. Construct `JavaPairRDDs` from `JavaRDDs` using special versions of the map operations, like `flatMapToPair` and `mapToPair`.



### Example:

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaPairRDD<String, Integer> pairs = lines.mapToPair(s -> new Tuple2(s, 1));
JavaPairRDD<String, Integer> counts = pairs.reduceByKey((a, b) -> a + b);
```

## Key-Value Pair RDD in Java

In Java, these pairs are represented using the `scala.Tuple2` class that exists in the Scala standard library. For this, you can call `new Tuple2(a, b)` and access its fields later with the given tuples. On the other hand, key-value pairs' RDDs are characterized by the `JavaPairRDD` class. For this, you can build `JavaPairRDDs` from `JavaRDDs` by using special versions of the map operations such as `flatMapToPair` and `mapToPair`.

Consider the given example that is using the `reduceByKey` operation on key-value pairs for counting the number of times each line of text occurs in a file.

## Using MapReduce and Pair RDD Operations

**Example:**

```

new HadoopRDD(sc: SparkContext, conf: JobConf, inputFormatClass: Class[_ <: InputFormat[K,
V]], keyClass: Class[K], valueClass: Class[V], minPartitions: Int)

def saveAsHadoopFile(path: String, keyClass: Class[_], valueClass: Class[_],
outputFormatClass: Class[_ <: org.apache.hadoop.mapred.OutputFormat[_, _]], conf:
JobConf = new JobConf): Unit

def saveAsHadoopFile[F <: OutputFormat[K, V]](path: String)(implicit fm: ClassManifest[F]): Unit
def saveAsNewAPIHadoopFile(path: String, keyClass: Class[_], valueClass: Class[_],
outputFormatClass: Class[_ <: org.apache.hadoop.mapreduce.OutputFormat[_, _]], conf:
Configuration): Unit
def saveAsNewAPIHadoopFile[F <: OutputFormat[K, V]](path: String)(implicit fm:
ClassManifest[F]): Unit

```

**Using MapReduce and Pair RDD Operations**

Consider the given example to use MapReduce and Pair RDD operations.

The highlighted code outputs the RDD to any Hadoop-supported file system as it uses a Hadoop OutputFormat class. The class is supporting the key and value types K and V.

This code outputs the RDD to any Hadoop-supported file system as it uses the new Hadoop API OutputFormat object.

## Reading Text File from HDFS

**Example:**

```
import org.apache.hadoop.io.{LongWritable, Text}  
import org.apache.hadoop.mapred.TextInputFormat  
import org.apache.spark.SparkContext  
object TextFileRead {  
    def main(args: Array[String]) {  
        val sc = new SparkContext(args(0), "sparkapiexamples") //actual textFile api converts to the  
        //following code  
        val dataRDD = sc.hadoopFile(args(1), classOf[TextInputFormat], classOf[LongWritable],  
        classOf[Text],  
        sc.defaultMinPartitions).map(pair => pair._2.toString)  
        println(dataRDD.collect().toList)  
    }  
}
```

## Reading Text File from HDFS

To perform batch analytics, Spark reads files from HDFS.

In the given example, a text file is being read from HDFS using the hadoopFile method of SparkContext and being printed as a list.

## Reading Sequence File from HDFS

**Example:**

```
import org.apache.hadoop.io.{LongWritable, Text}  
import com.spark.hadoopintegration.SalesRecordWritable  
import org.apache.hadoop.io.NullWritable  
import org.apache.spark.SparkContext  
object SequenceFileRead {  
    def main(args: Array[String]) {  
        val sc = new SparkContext(args(0), "apiexamples")  
        val dataRDD =  
            sc.sequenceFile(args(1), classOf[NullWritable], classOf[SalesRecordWritable]).map(_._2)  
            .println(dataRDD.collect().toList)  
    }  
}
```

## Reading Sequence File from HDFS

SequenceFile provides a file format for storing data as serialized key value pairs used in Hadoop. You can read sequence files stored in HDFS and perform transformation of the data stored.

In the given example, a sequence file is being read by using customized Writable class and being printed as a list.

## Writing Text Data to HDFS

**Example:**

```
import org.apache.spark.SparkContext
object TextFileWriteToHDFS {
    def main(args: Array[String]) {
        val sc = new SparkContext(args(0), "apiexamples")
        val dataRDD = sc.textFile(args(1))
        val outputPath = args(2)
        val itemPair = dataRDD.map(row => { val columns = row.split(",") (columns(2), 1) })
        /* itemPair is MappedRDD which is a pair. We can import the following to get more methods */
        import org.apache.spark.SparkContext._
        val result = itemPair.reduceByKey(_ +_) result.saveAsTextFile(outputPath)
    }
}
```

## Writing Text Data to HDFS

You can store the result back into HDFS for persistence once transformation is done or the business logic is applied on the RDD.

For example, here a text file is being written into HDFS using the saveAsTextFile method.

## Writing Sequence File to HDFS

**Example:**

```
com.spark.apandexamples.serialization.SalesRecordParser
import com.spark.hadoopintegration.SalesRecordWritable
import org.apache.hadoop.io.NullWritable
import org.apache.spark.SparkContextimport org.apache.spark.SparkContext.
object SequenceFilePersist { def main(args: Array[String]) {
  val sc = new SparkContext(args(0), "apandexamples")
  val dataRDD = sc.textFile(args(1))  val outputPath = args(2)
  val salesRecordRDD = dataRDD.map(row => {
    val parseResult = SalesRecordParser.parse(row)  parseResult.right.get  })
  val salesRecordWritableRDD = salesRecordRDD.map(salesRecord => { (NullWritable.get(), new
    SalesRecordWritable(salesRecord.transactionId,
    salesRecord.customerId,salesRecord.itemId, salesRecord.itemValue))  })
  salesRecordWritableRDD.saveAsSequenceFile(outputPath) }}
```

## Writing Sequence File to HDFS

In the example given on the screen, customized Writable class salesRecordWritableRDD is being used for writing data as sequence file in HDFS.

## Using GroupBy

**Example:**

```
import java.util.Random
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext._

/** 
 * Usage: GroupByTest [numMappers] [numKVPairs] [KeySize] [numReducers]
 */
object GroupByTest {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("GroupBy Test")
    var numMappers = if (args.length > 0) args(0).toInt else 2
    var numKVPairs = if (args.length > 1) args(1).toInt else 1000
    var valSize = if (args.length > 2) args(2).toInt else 1000
    var numReducers = if (args.length > 3) args(3).toInt else numMappers
    val sc = new SparkContext(sparkConf)
```

## Using GroupBy

In this example, Scala is being used for performing group by operation in Spark.

## Using GroupBy (contd.)

**Example:**

```
val pairs1 = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>
    val ranGen = new Random
    var arr1 = new Array[(Int, Array[Byte])](numKVPairs)
    for (i <- 0 until numKVPairs) {
        val byteArr = new Array[Byte](valSize)
        ranGen.nextBytes(byteArr)
        arr1(i) = (ranGen.nextInt(Int.MaxValue), byteArr)
    }
    arr1
}.cache()
// Enforce that everything has been calculated and in cache
pairs1.count()
println(pairs1.groupByKey(numReducers).count())
sc.stop()
}
```

© Copyright 2015, Simplilearn. All rights reserved.

56

**Using GroupBy (contd.)**

The further code is displayed.

## Demo—Run a Scala Application Performing GroupBy Operation

This demo will show the steps to run a Scala application that performs GroupBy operation.

## Demo—Run a Scala Application Performing GroupBy Operation

In this demo, you will learn how to run a Scala application that performs group by operation.

In this code, we will generate some random numbers and then we will perform group by operation on them to see how many group by operations were performed. In the code, we are using the parallelize method to generate different sets of number across the partitions. The generated random integer data sets are persisted in the memory and used for group by the key operation.

We can run this program as we ran our first Scala application using the spark\_submit script file available in the bin directory of the Spark installation path. We need to type the command as shown on the screen.

From the output displayed on the screen, you can see it has grouped 2000 items.

We can also pass number of mappers, key value pairs, and keys and number of reducer as a run time parameter. We will mention it after the jar name.

From the output displayed on the screen, you can see that it has grouped 1500 items, which is a multiple of number of mappers into number of keys.

## Demo—Run a Scala Application Using the Scala Shell

This demo will show the steps to run a Scala application using the Scala shell.

## Demo—Run a Scala Application Using the Scala Shell

In this demo, you will learn how to run the Scala application using the Scala shell.

Here, we are going to write a simple application to read a local file and count the number of lines in which characters “a” and “b” have occurred.

As shown in the code, we have imported SparkContext, SparkConf and rest of the classes from the same package. This class has the main method in which we are reading the “README.md” file by using the textFile method of the SparkContext object. After that, we are using the filter method to read each line and check the occurrence of characters a and b. We are caching this count in the memory by using the cache method.

After completing our source code, we can run this application by typing “SampleApp.main(null)” as shown on the screen .

You can see that there are 60 lines with the character “a” and 29 lines with the character “b” in the README.md file.

## Demo—Write and Run a Java Application

This demo will show the steps to write and run a Java application.

© Copyright 2015, Simplilearn. All rights reserved.

59

## Demo—Write and Run a Java Application

In this demo, you will learn how to write and run a Java application.

Let's create a file "SimpleApp.java" in this folder. Here, we are going to write our first Spark java application to read a local file and count the number of lines in which characters "a" and "b" have occurred .

As shown in the code, we have imported JavaSparkContext, SparkConf and rest of the classes from the same package. This class has the main method in which we are reading the "README.md" file by using the textFile method of the JavaSparkContext object. After that, we are using the filter method to read each line and check the occurrence of characters a and b. We are caching this count in the memory by using the cache method. In the filter method, we have overridden the call method to return a Boolean variable.

To run this application, we need to type the command as shown on the screen.

**QUIZ  
1**

Which of the following method is a type of an action in RDD?

- a. Filter
- b. Reduce
- c. Sample
- d. Map

**QUIZ  
2**

Which of following statement is true about OrderedRDDFunctions?

- a. Include methods that become available if the data items are two-component tuples
- b. Include methods that become available when the data items have a two-component tuple structure
- c. Include methods that let you to create Hadoop sequences from RDDs; data items must be two component key-value tuples
- d. Include methods to aggregate numeric values that become available when a RDD data items can be implicitly converted to the Scala data-type double



**QUIZ  
3**

Which of the following is the correct syntax to create a broadcast variable?

- a. new broadcast(Array(1, 2, 3))
- b. Spark.broadcast(Array(1, 2, 3))
- c. SparkContext.broadcast()
- d. Context.broadcast(Array(1, 2, 3))

**QUIZ  
4**

What is the recommended when choosing the right RDD persistence storage level when fast fault recovery is required?

- a. Use MEMORY\_ONLY\_SER
- b. Select a fast serialization library
- c. Use the replicated storage levels
- d. Leave them that way



**ANSWERS:**

S.No.	Question	Answer & Explanation
1	Which of the following method is a type of an action in RDD?	b. The reduce method is a type of an action in RDD. Filter, sample, and map are transformations.
2	Which of following statement is true about OrderedRDDFunctions?	a. OrderedRDDFunctions include methods that become available if the data items are two-component tuples. The other statements define DoubleRDDFunctions, PairRDDFunctions, and SequenceFileRDDFunctions.
3	Which of the following is the correct syntax to create a broadcast variable?	c. The option c correctly defines a broadcast variable..
4	What is the recommended when choosing the right RDD persistence storage level when fast fault recovery is required?	c. If fast fault recovery is required, you should use the replicated storage levels.

## Summary



Let us summarize the topics covered in this lesson:



- There are two ways to create RDDs: Parallelize an existing collection and reference an external dataset.
- Spark supports text files, Sequence Files, and other Hadoop Input Formats.
- RDDs support two types of operations: transformations and actions.
- Various features of RDDs are immutable, persistence, lazy evaluated, and more.
- Choose the applicable storage level, as there are trade-offs between memory usage and CPU efficiency.
- To invoke the Spark shell, go to the home directory and run the applicable code.
- A few actions that can be performed on files in Spark shell RDDs are getting count from a file and getting the first element from a file.

© Copyright 2015, Simplilearn. All rights reserved.

69

## Summary

Let us summarize the topics covered in this lesson:

- There are two ways to create RDDs: parallelize an existing collection and reference an external dataset.
- Spark supports text files, SequenceFiles, and other Hadoop InputFormats.
- RDDs support two types of operations: transformations and actions.
- Various features of RDDs are immutable, persistence, lazy evaluated, and more.
- Choose the applicable storage level, as there are trade-offs between memory usage and CPU efficiency.
- To invoke the Spark shell, go to the home directory and run the applicable code.
- A few actions that can be performed on files in Spark shell RDDs are getting count from a file and getting the first element from a file.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- To build a Spark project with SBT, create the build and test it.
- Broadcast variables allow to keep a read-only variable cached on every machine, while accumulators are added only through an associative operation.
- An RDD acts as a handle for a collection of individual data partitions.
- In Spark, there are four extensions to the RDD API: DoubleRDDFunctions, PairRDDFunctions, OrderedRDDFunctions, and SequenceFileRDDFunctions.
- The def aggregate[U](zeroValue: U)(seqOp: Function2[U, (K, V), U], combOp: Function2[U, U, U]): U method aggregates the elements of each partition.
- Transformations in RDD include sample, map, filter, and groupByKey.
- Some special operations are only available on RDDs of key-value pairs.
- In Java, these are represented using the scala.Tuple2 class.

- To build a Spark project with SBT, create the build and test it.
- Broadcast variables allow to keep a read-only variable cached on every machine, while accumulators are added only through an associative operation.
- An RDD acts as a handle for a collection of individual data partitions.
- In Spark, there are four extensions to the RDD API: DoubleRDDFunctions, PairRDDFunctions, OrderedRDDFunctions, and SequenceFileRDDFunctions.
- The given method aggregates the elements of each partition.
- Transformations in RDD include sample, map, filter, and groupByKey.
- Some special operations are only available on RDDs of key-value pairs.
- In Java, these are represented using the scala.Tuple2 class.

This concludes ‘Using RDD for Creating Applications in Spark.’

The next lesson is ‘Running SQL Queries Using SparkSQL’

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 3 “Using RDD for Creating Applications in Spark” of the Apache Spark and Scala course. The next lesson is Running SQL Queries Using SparkSQL.

## Lesson 4—Introduction to Cassandra



## Objectives



After completing this lesson, you will be able to:



- Explain the importance and features of Spark SQL
- Describe the methods to convert RDDs to DataFrames
- Explain a few concepts of Spark SQL
- Describe the concept of hive integration

## Objectives

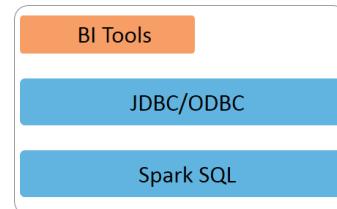
After completing this lesson, you will be able to:

- Explain the importance and features of SparkSQL
- Describe the methods to convert RDDs to DataFrames
- Explain a few concepts of SparkSQL, and
- Describe the concept of hive integration

## Importance of Spark SQL

Spark SQL is a Spark module used for structured data processing. It:

- Acts as a distributed SQL query engine
- Provides DataFrames for programming abstraction
- Allows to query structured data in Spark programs
- Can be used with platforms such as Scala, Java, R, and Python



SQL and DataFrames provide a collective method to access multiple data sources, including Avro, Hive, ORC, JSON, Parquet, and JDBC. Spark offers industry standard ODBC and JDBC connectivity.

© Copyright 2015, Simplilearn. All rights reserved.

3

## Importance of Spark SQL

Let us first understand the importance of SparkSQL. It is a module of Spark that is used for structured data processing. The module is capable of acting as a distributed SQL query engine and provides a feature called DataFrames that provides programming abstraction.

The module allows you to query structured data in programs of Spark by using SQL or a similar DataFrame API. Moreover, it can be used with different platforms such as Scala, Java, R, and Python.

SQL and DataFrames provide a collective method for accessing multiple data sources, which include Avro, Hive, ORC, JSON, Parquet, and JDBC. Data can also be joined among these sources.

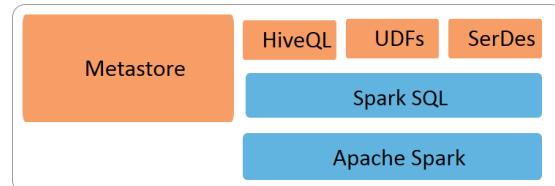
For ETL and business intelligence tools, Spark offers industry standard ODBC and JDBC connectivity.

The image given on the screen shows the integration of various components.

## Benefits of Spark SQL

The benefits of Spark SQL are listed below:

- **Hive Compatibility:** Compatible with the existing Hive queries, UDFs, and data
- **SQL Queries Support:** Mixes SQL queries with Spark programs
- **Components Support:** Includes a cost-based optimizer, code generations, and columnar storage
- **Spark Engine Inclusion:** Allows it to scale to multi hour queries and thousands of nodes
- **Comprehensive:** Does not require a different engine for the historical data



© Copyright 2015, Simplilearn. All rights reserved.

4

## Benefits of Spark SQL

SparkSQL provides hive compatibility. It reuses the Hive metastore and frontend. Due to this, it is completely compatible with the existing Hive queries, UDFs, and data. To get benefitted with this feature, just install it with Hive. The image on the screen shows the interaction of these components.

In addition, it mixes SQL queries with programs of Spark easily.

For making queries fast, it also includes a cost-based optimizer, code generations, and columnar storage.

Moreover, this module uses the Spark engine that allows it to scale to multi hour queries and thousands of nodes. This feature gives full mid-query fault tolerance.

With this module, you do not need to use a different engine for the historical data.

## DataFrames



DataFrames represent a distributed collection of data, in which data is organized into columns that are named.

Construct a DataFrame

Use sources like tables in Hive, structured data files, existing RDDs, and external databases.

Convert them to RDDs

Call the rdd method that returns the DataFrame content as an RDD of rows.



In prior versions of Spark SQL API, SchemaRDD has been renamed to DataFrame.

© Copyright 2015, Simplilearn. All rights reserved.

5

## DataFrames

Let us now understand the concept of dataframes. It is a distributed collection of data, in which data is organized into columns that are named. You can compare it with a data frame in R or Python or a table in a relational database, however with much richer optimization functionality.

To construct a dataframe, you can use various sources like tables in Hive, structured data files, existing RDDs, and external databases. To convert them to RDDs, you can call the rdd method that returns the DataFrame content as an RDD of rows.

The DataFrame API is available for different platforms like R, Scala, Python, and Java.

Note that in prior versions of Spark SQL API, SchemaRDD has been renamed to DataFrame.

## SQLContext

The SQLContext class or any of its descendants acts like the entry point into all functionalities.

**Example:**

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

**How to get the benefit of a superset of the basic SQLContext functionality?**

Build a HiveContext to:

- Use the writing ability for queries
- Access Hive UDFs and read data from Hive tables

## SQLContext

The other component of SparkSQL is SQLContext. The SQLContext class or any of its descendants acts like the entry point into all functionalities. You just need a SparkContext to build a basic SQLContext. The code given on the screen shows how to create an SQLContext object.

In addition, you can also build a HiveContext for availing the benefit of a superset of the basic SQLContext functionality. It also provides more features such as the writing ability for queries by using the more comprehensive HiveSQL parser. Other features are accessing Hive UDFs and the read data ability from Hive tables. The entire data sources that are available to an SQLContext still exist. Therefore, you do not require an existing Hive setup for using a HiveContext.

**Points to Remember:**

- HiveContext is just packaged separately to avoid the dependencies of Hive in the default Spark build.
- You can also use the spark.sql.dialect option to select the specific variant of SQL used for parsing queries; use the SET key=value command in SQL or the setConf method on an SQLContext.
- Sql is the only dialect available for an SQLContext; the default is "hiveql".
- On an SQLContext, the sql function allows applications to programmatically run SQL queries and then return a DataFrame as a result.

**Example:**

```
val df = sqlContext.sql("SELECT * FROM table")
```

## SQLContext (contd.)

HiveContext is just packaged separately for avoiding the dependencies of Hive in the default Spark build. For your applications, if these dependencies are not a concern, then HiveContext is recommended for Spark 1.3. The future releases will be focused on getting SQLContext up to feature parity with a HiveContext.

You can also use the spark.sql.dialect option to select the specific variant of SQL, which is used for parsing queries. To change this parameter, you can use the SET key=value command in SQL or the setConf method on an SQLContext.

The only dialect available for an SQLContext is "sql" that makes use of a simple SQL parser. However, in a HiveContext, the default is "hiveql", which is much more comprehensive. Therefore, it is recommended for most use cases.

On an SQLContext, the sql function allows applications to programmatically run SQL queries and then return a DataFrame as a result. The code given on the screen shows the use of the sql function.

## Creating a DataFrame

**Example:**

```
val sc: SparkContext // An existing SparkContext.  
val sqlContext = new org.apache.spark.sql.SQLContext(sc)  
val df = sqlContext.read.json("examples/src/main/resources/customers.json")  
// Displays the content of the DataFrame to stdout  
df.show()
```

## Creating a DataFrame

Let us now view an example code to create a DataFrame. In this example, a DataFrame is being created based on a JSON file content.

## Using DataFrame Operations

DataFrames provide a domain-specific language that can be used for structured data manipulation in Java, Scala, and Python.



### Example:

```
val sc: SparkContext // An existing SparkContext.  
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)  
    // Create the DataFrame  
    val df = sqlContext.read.json("examples/src/main/resources/customers..json")  
    Show the content of the DataFrame  
    df.show()  
    // Print the schema in a tree format  
    df.printSchema()  
    // Select only the "name" column  
    df.select("name").show()
```

## Using DataFrame Operations

DataFrames are capable of providing a domain-specific language that can be used for structured data manipulation in Java, Scala, and Python.

The example given on the screen shows structured data processing using DataFrames. In this code, DataFrames are providing functions like printschema, show, groupby, filter, and more.

## Using DataFrame Operations (contd.)

**Example:**

```
// Select everybody, but increment the age by 1  
df.select(df("name"), df("age") + 1).show()  
// Select people older than 21  
df.filter(df("age") > 21).show()  
// Count people by age  
df.groupBy("age").count().show()
```

**Using DataFrame Operations (contd.)**

The further code is displayed.

## Demo—Run SparkSQL with a Dataframe

This demo will show the steps to run SparkSQL with a dataframe.

## Demo—Run SparkSQL with a DataFrame

In this demo, you will learn how to run Spark sql with a dataframe.

“spark-shell” available in the bin directory of the Spark installation path enables us to run SQL queries using data frame.

First, to run sql using spark, we need to create an instance of the SparkContext object. The schema of a table can be defined using case classes in spark sql. Here, we are defining the Customer case class to represent the schema of the customer table.

We can read local or HDFS file by using the `textFile` method and represent it as an RDD. The resultant RDD can be converted into a data frame by calling the `toDF()` method. Note the code dispayed on the screen

Once we have a variable representing a data frame, we can call the “`registerTempTable`” method on it to treat it as a database table.

After registering the data frame as a table, we can run a SQL query on it as if we are running a query on the table. Here, we are going to use `show()` , `printSchema()`, `filter()` and `groupby()` methods to run various types of queries on the customer table.

## Interoperating with RDDs

To convert existing RDDs into DataFrames, Spark SQL supports two methods:

**Reflection Based**

- Infers an RDD schema containing specific types of objects
- Works well when the schema is already known when writing the Spark application

**Programmatic**

- Lets to build a schema and apply to an already existing RDD
- Allows to build DataFrames when you do not know the columns and their types until runtime

## Interoperating with RDDs

To convert existing RDDs into DataFrames, SparkSQL supports two methods. The first method uses the reflection based approach for inferring an RDD schema containing specific types of objects. It works well when the schema is already known when you are writing your Spark application. In such cases, it leads to more concise code.

The second method is a programmatic approach that lets you build a schema and apply to an already existing RDD. However, this method is more verbose, but it lets you build DataFrames when you do not know the columns and their types until runtime.

## Using the Reflection-Based Approach

For Spark SQL, the Scala interface allows to convert an RDD with case classes to a DataFrame automatically.

The case class:

- Has the table schema, where the arguments names to the case class are read using the reflection method
- Can be nested and used to contain complex types like sequence of arrays

Implicitly convert the resultant RDD to a DataFrame and register it as a table. Use it in the subsequent SQL statements.

## Using the Reflection-Based Approach

Let us first talk about the reflection based approach.

For SparkSQL, the Scala interface allows to convert an RDD with case classes to a DataFrame automatically. The case class has the table schema, where the arguments names to the case class are read using the reflection method. These then become the columns names.

In addition, you can also nest the case classes and use them to contain complex types like sequence of arrays. You can also implicitly convert the resultant RDD to a DataFrame and register it as a table. The tables can then be used in the subsequent SQL statements.

## Using the Reflection-Based Approach (contd.)

**Example:**

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._

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

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/people.txt").map(_.split(",")).map(p => Person(p(0),
p(1).trim.toInt)).toDF()
people.registerTempTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sqlContext.sql("SELECT name, age FROM people WHERE age >= 13 AND age <= 19")
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
// or by field name:
teenagers.map(t => "Name: " + t.getAs[String]("name")).collect().foreach(println)
// row.getValuesMap[T] retrieves multiple columns at once into a Map[String, T]
teenagers.map(_.getValuesMap[Any](List("name", "age"))).collect().foreach(println)
```

## Using the Reflection-Based Approach

The example given on the screen shows how to infer the schema using the reflection based approach.

## Using the Programmatic Approach

This method is used when you cannot define case classes ahead of time; for example, when the records structure is encoded in a text dataset or a string. Use the steps below:

Use the existing RDD to create an RDD of rows

↳ Create the schema represented by a StructType and matches the rows structure

↳ Apply the schema to the RDD of rows using the createDataFrame method

## Using the Programmatic Approach

Now, let's talk about the second method, which is a programmatic approach.

This method is used when you cannot define case classes ahead of time. For instance, when the records structure is encoded in a text dataset or a string. In such cases, you can create a DataFrame using the three steps listed on the screen.

As the first step, you need to use the existing RDD to create an RDD of rows. Next, you need to create the schema that is represented by a StructType and matches the rows structure just created.

As the final step, you need to apply the schema created to the RDD of rows using the createDataFrame method. This method is provided by SQLContext.

## Using the Programmatic Approach (contd.)

**Example:**

```
// sc is an existing SparkContext.  
val sqlContext = new org.apache.spark.sql.SQLContext(sc)  
// Create an RDD  
val people = sc.textFile("examples/src/main/resources/people.txt")  
// The schema is encoded in a string  
val schemaString = "name age"  
import org.apache.spark.sql.Row;  
import org.apache.spark.sql.types.{StructType,StructField,StringType};  
// Generate the schema based on the string of schema, Convert records of the RDD (people) to Rows and  
// Apply the schema to the RDD.  
val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))  
val rowRDD = people.map(_.split(",")).map(p => Row(p(0), p(1).trim))  
val peopleDataFrame = sqlContext.createDataFrame(rowRDD, schema)  
peopleDataFrame.registerTempTable("people")
```

**Using the Programmatic Approach (contd.)**

The example given on the screen shows how to specify the schema programmatically.

## Demo—Run Spark SQL Programmatically



This demo will show the steps to run Spark SQL by programmatically specifying the schema of a table.

© Copyright 2015, Simplilearn. All rights reserved.

17

## Demo—Run SparkSQL Programmatically

In this demo, you will learn how to run spark SQL by programmatically specifying the schema of a table.

“spark-shell” available in the bin directory of the spark installation path enables us to run SQL queries by programmatically specifying the schema of a table.

First, to run sql using spark, we need to create an instance of the SparkContext object.

We can read local or HDFS file by using the `textFile` method and represent it as an RDD. We can use `StructType` to define the schema of the table, which uses the reflation mechanism to define a field name and its data type.

We can use the “`createDataFrame`” method of the `SQLContext` object to create a dataframe by taking a variable representing the table’s row data and table’s schema as parameters.

Once we have a variable representing the data frame, we can call the “`registerTempTable`” method on it to treat as a database table.

After registering the data frame as a table, we can run an SQL query on it as if we are running a query on the table. Here, we are going to use the `SQLContext.sql()` method to execute the SQL query and then we will transform the output using the `map` method before printing them onto the console.

## Data Sources

The DataFrame interface allows to operate on various data sources as a normal RDD or by registering as a temporary table. There are two types of methods:

**General Methods**

- Allow to load and save data using the Spark data sources
- Use the default data source, which is parquet unless configured by spark.sql.sources.default

**Example:**

```
val df =
  sqlContext.read.load("examples/src/main/resources/users.parquet")
  df.select("name",
    "favorite_color").write.save("namesAndFavColors.parquet")
```

**Specific Methods**

- Allow to operate on built-in data sources
- Allow to specify the data source with any extra options to be passed to the data source using their fully qualified names, which is rg.apache.spark.sql.parquet

**Example:**

```
val df =
  sqlContext.read.format("json").load("examples
  /src/main/resources/people.json")
  df.select("name",
    "age").write.format("parquet").save("namesAn
  dAges.parquet")
```

## Data Sources

The DataFrame interface allows to operate on various data sources. You can operate it on data sources as a normal RDD or by registering as a temporary table. By registering a DataFrame as a table, you can run SQL queries on its data.

These methods are of two types: general methods that allow to load and save data using the Spark Data Sources and specific methods that allow you to operate on built-in data sources.

The general methods are the simplest ones and use the default data source, which is parquet unless configured by spark.sql.sources.default. They are used for all operations. An example is given on the screen.

For specific methods, the data source used with any extra options to be passed to the data source can be manually specified. These sources are specified using their fully qualified names, which is rg.apache.spark.sql.parquet. However, for the specific methods and built-in sources, you are allowed to use their short forms such as jdbc, json, and parquet. The example given on the screen shows the syntax that can be used to convert a DataFrame of any type to another.

## Save Modes



Save operations can acquire a SaveMode. These modes specify the method of handling any existing data. Note that these modes are not atomic and do not use any locking. The table below lists and explains the different save modes used in Scala and Java:

Save Mode	Explanation
SaveMode.ErrorIfExists	When saving a DataFrame to a data source, if data already exists, an exception is expected to be thrown.
SaveMode.Append	When saving a DataFrame to a data source, if data/table already exists, contents of the DataFrame are expected to be appended to the existing data.
SaveMode.Overwrite	When saving a DataFrame to a data source, if data/table already exists, the existing data is expected to be overwritten by the contents of the DataFrame.
SaveMode.Ignore	When saving a DataFrame to a data source, if data already exists, the save operation is expected to not save the contents of the DataFrame and to not change the existing data. This is similar to a CREATE TABLE IF NOT EXISTS in SQL.

## Save Modes

Optionally, save operations can acquire a SaveMode. These modes specify the method of handling any existing data. Note that these modes are not atomic and do not use any locking. Therefore, you cannot safely use multiple writers that try to write data to the same location. In addition, when overwriting, the data gets deleted before you could write any new data.

The table given on the screen lists and explains the different save modes used in Scala and Java.

## Saving to Persistent Tables

Use the **saveAsTable** command to:

- Materialize the DataFrames contents
- Create a data pointer in the HiveMetastore

Create a DataFrame for a persistent table

Call the table method with the table name on an SQLContext.



The saveAsTable command by default creates a managed table.

## Saving to Persistent Tables

You can use the `saveAsTable` command to save DataFrames as persistent tables when working with a `HiveContext`. This command is capable for materializing the DataFrames contents and creating a data pointer in the `HiveMetastore`. The persistent tables thus resulted exist even after you restart your Spark program, provided the connection to the same metastore is maintained. To create a DataFrame for a persistent table, you can call the `table` method with the table name on an `SQLContext`.

The `saveAsTable` command by default creates a managed table, which means that the data location is controllable by the metastore. When a table is dropped, these tables will also have automatically deleted their data.

## Parquet Files



Parquet represents a columnar format supported by various data processing systems. Spark SQL supports to read and write these files. The schema of the original data is automatically preserved.



### Example:

```
import sqlContext.implicits._  
val people: RDD[Person] = ... // An RDD of case class objects, from the previous example  
people.write.parquet("people.parquet")  
val parquetFile = sqlContext.read.parquet("people.parquet")  
parquetFile.registerTempTable("parquetFile")  
val teenagers = sqlContext.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")  
teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

## Parquet Files

Parquet represents a columnar format supported by various data processing systems. SparkSQL supports to read and write these files. The schema of the original data is automatically preserved.

The example given on the screen shows the related operations.

## Partition Discovery



A common optimization approach used in systems such as Hive:

- When a table is partitioned, data is generally saved in various directories.
- The Parquet data source can automatically discover and infer the information of partitioning.
- Pass path/to/table to SQLContext.read.load or SQLContext.read.parquet to allow Spark SQL to extract the information of partitioning from the paths automatically.

## Partition Discovery

A common optimization approach that is used in systems such as Hive is table partitioning. When a table is partitioned, data is generally saved in various directories. The partitioning column values are encoded in every directory path.

Now, the Parquet data source can automatically discover and infer the information of partitioning. You can pass path/to/table to SQLContext.read.load or SQLContext.read.parquet that will allow Spark SQL to extract the information of partitioning from the paths automatically.

## Schema Merging

Parquet supports schema evolution. Start with a basic schema and then add more columns gradually to it as required. The Parquet data source can now detect this case and merge these files' schemas.

**Example:**

```
import sqlContext.implicits._  
val df1 = sc.makeRDD(1 to 5).map(i => (i, i * 2)).toDF("single", "double")  
df1.write.parquet("data/test_table/key=1")  
val df2 = sc.makeRDD(6 to 10).map(i => (i, i * 3)).toDF("single", "triple")  
df2.write.parquet("data/test_table/key=2")  
// Read the partitioned table  
val df3 = sqlContext.read.parquet("data/test_table")  
df3.printSchema()
```

## Schema Merging

Parquet provides support to schema evolution similar to Avro, Thrift, and ProtocolBuffer. You can start with a basic schema and then add more columns gradually to it as required. In this manner, you can have various Parquet files at the end with various mutually compatible schemas. Now, the Parquet data source can detect this case automatically and merge these files' schemas.

An example to use this is given on the screen.

## JSON Data

Spark SQL can infer a JSON dataset schema and load it as a DataFrame. Use the SQLContext.read.json() method on a JSON file or an RDD of string. The JSON file here is not a typical one.

**Example:**

```
val path = "examples/src/main/resources/people.json"
val people = sqlContext.read.json(path)
people.printSchema()
people.registerTempTable("people")
val teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
val anotherPeopleRDD = sc.parallelize(
    """>{"name":"Yin","address":{"city":"Columbus","state":"Ohio"}}, Nil)
val anotherPeople = sqlContext.read.json(anotherPeopleRDD)
```

## JSON Data

Spark SQL is capable of inferring a JSON dataset schema and loading it as a DataFrame. You can perform this conversion using the SQLContext.read.json() method on a JSON file or an RDD of string.

An important point is that the JSON file here is not a typical one. Every line needs to have a different self-contained and valid JSON object. Therefore, as a result, a general multi-line JSON file will fail in most of the cases.

An example to use this is given on the screen.

## Hive Table

Spark SQL can also read and write data stored in an Apache Hive. The default Spark assembly does not include Hive and the support is enabled by adding the -Phive-thriftserver and the –Phive flags. To configure a Hive, place your `hive-site.xml` file in the configuration directory.

**Example:**

```
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
    sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
    sqlContext.sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE
src")
    // Queries are expressed in HiveQL
    sqlContext.sql("FROM src SELECT key, value").collect().foreach(println)
```

## Hive Table

Spark SQL is also capable to read and write data that is stored in an Apache Hive. Hive has multiple dependencies. This is the reason why the default Spark assembly does not include it. The related support is enabled by adding the -Phive-thriftserver and the –Phive flags to the build of Spark.

By doing this, it creates a new assembly jar including Hive. This new jar must also exist on all the worker nodes. This is important as they would need to access the serialization and deserialization libraries of the Hive for accessing the stored data.

To configure a Hive, you can place your `hive-site.xml` file in the configuration directory. The example to do so is given on the screen.

## DML Operation—Hive Queries

Create a HiveContext when working with Hive to write queries using HiveQL and find tables in the MetaStore. In case of existing deployment, it can still be created. It builds warehouse and metastore\_db automatically, if not configured by the hive-site.xml, in the present directory.

**Example:**

```
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
sqlContext.sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")
sqlContext.sql("FROM src SELECT key, value").collect().foreach(println)
```

## DML Operation—Hive Queries

You must create a HiveContext when you are working with Hive. It provides support to write queries using HiveQL and finds tables in the MetaStore. In case you do not have any existing deployment of Hive, you can still create a HiveContext. It builds warehouse and metastore\_db automatically, if it is not configured by the hive-site.xml, in the present directory.

An example of the same is given on the screen.

## Demo—Run Hive Queries Using Spark SQL



This demo will show the steps to run hive queries using Spark SQL.

© Copyright 2015, Simplilearn. All rights reserved.

27

## Demo—Run Hive Queries Using Spark SQL

In this demo, you will learn how to run hive queries using Spark SQL.

“spark-shell” available in the bin directory of the spark installation path enables us to run sql queries that can be used to perform batch analytics on Hive.

First, to run SQL on Hive using spark, we need to create an instance of the HiveContext object.

We can run Hive queries using the `sql()` method of the HiveContext object.

## JDBC to Other Databases

Spark SQL contains a data source using JDBC that can:

- Read data from various other databases
- Be used from Python or Java because Classtag is not required

To start

Include the related driver for the specific database on the classpath of Spark.

**Example:**

```
SPARK_CLASSPATH=postgresql-9.3-1102-jdbc41.jar bin/spark-shell  
val jdbcDF = sqlContext.load("jdbc", Map( "url" -> "jdbc:postgresql:dbserver", "dbtable" ->  
"schema.tablename"))
```

© Copyright 2015, Simplilearn. All rights reserved.

28

## JDBC to Other Databases

To read data from various other databases, SparkSQL also contains a data source using JDBC. The results are in the form of a DataFrame. Additionally, they can be easily joined with data sources and processed in Spark SQL. Moreover, the JDBC data source can be easily used from Python or Java because you need not provide Classtag. You should note that the Spark SQL JDBC server is different from it, which lets other applications for running queries using Spark SQL.

To start with, one must include the related driver for the specific database on the classpath of Spark. For instance, from the Spark shell, to connect to postgres, you need to run the command as depicted on the screen.

## Supported Hive Features



Spark SQL is compatible with User-Defined Serialization Formats (SerDes), User-Defined Functions (UDFs), and Hive Metastore. At present, it is based on Hive 0.12.0 and 0.13.1. It supports:

### Hive Query Statements

Select, OrderBy, GroupBy, ClusterBy, and SortBy

### Hive Operators

Relational ( $=$ ,  $\neq$ ,  $=$ ,  $\neq$ ,  $<$ ,  $>$ ,  $\geq$ ,  $\leq$ ), logical (AND,  $\&\&$ , OR,  $\|$ ), arithmetic (+, -, \*, /, %), Mathematical (sign, ln, cos), complex type constructors, string functions (instr, length, printf)

## Supported Hive Features

Spark SQL is compatible with user defined serialization formats or SerDes, user defined functions or UDFs, and Hive Metastore. At present, it is based on Hive 0.12.0 and 0.13.1.

The Hive query statements are listed on the screen. These are Select, Order By, Group By, Cluster By, and Sort By.

In addition, the Hive operators are relational operators, logical operators, arithmetic operators, mathematical functions, complex type constructors, and string functions.

## Supported Hive Features (contd.)

A few other supported hive features are:

- Join: Join, Left Semi Join, Cross Join
- Unions
- Sub Queries
- Sampling
- Explain
- Partitioned Tables
- Views
- Functions: CREATE TABLE, ALTER TABLE, CREATE TABLE AS SELECT

## Supported Hive Features (contd.)

A few other supported hive features are joins, which include join, left semi join, and cross join. It also supports unions, sub queries, sampling, explain, partitioned tables, views, and all hive DDL functions such as Create Table, Alter table, and create table as select.

## Supported Hive Data Types



The supported hive data types are:

TINYINT	STRING
SMALLINT	BINARY
INT	TIMESTAMP
BIGINT	DATE
BOOLEAN	ARRAY<>
FLOAT	MAP<>
DOUBLE	STRUCT<>

## Supported Hive Data Types

Let's now talk about the supported hive data types. These are listed on the screen, which include string, binary, date, array, map, Boolean, struct and more.

## Case Classes



Case classes represent the underneath data schema and the APIs include:

- CreateSchemaRDD
- SQLContext
- RegisterTempTable

## Case Classes

Let's now talk about case classes, which represent the underneath data schema.

The APIs that are used in case classes include createSchemaRDD, SQLContext, and registerTempTable.

## Case Classes (contd.)

**Example:**

```
import org.apache.spark._  
import org.apache.spark.rdd.RDD  
/** * Creating schemaRDD from case classes */  
object CaseClasses { def main(args: Array[String])  
{ val sc : SparkContext = new SparkContext(args(0), "caseclass")  
val sqlContext = new org.apache.spark.sql.SQLContext(sc)  
import sqlContext.createSchemaRDD  
val sales = sc.textFile(args(1)).map(_.split(",")).map(p=> Sales(p(0).trim.toInt, p(1).trim.toInt,  
p(2).trim.toInt, p(3).trim.toDouble))  
sales.registerTempTable("sales")  
val firstItemSales = sqlContext.sql("SELECT * FROM sales WHERE itemId=1")  
println(firstItemSales.map(t => "customerID: " + t(1)).collect().toList)  
}}
```

**Case Classes (contd.)**

The example given on the screen shows how to define a case class in Spark SQL.

**QUIZ  
1**

Which of the following statement is true about “SaveMode.Append”?

- a. If data already exists, the save operation is expected to not save the contents of the DataFrame and to not change the existing data.
- b. Contents of the DataFrame are expected to be replaced to the existing data.
- c. When saving a DataFrame to a data source, if data already exists, an exception is expected to be thrown.
- d. Contents of the DataFrame are expected to be appended to the existing data.

**QUIZ  
2**

Which of the following object has to be created for interacting with the Hive data?

- a. HiveContext
- b. SparkSqlContext
- c. SparkContext
- d. InteractContext



**QUIZ  
3**

Which of the following statements are true about Spark SQL? *Select all that apply.*

- a. Hive compatibility
- b. Cost-based optimizer, code generations, and columnar storage
- c. Different engine for the historical data
- d. SQL queries independence

**QUIZ  
4**

Which of the following statements are true about DataFrames? *Select all that apply.*

- a. DataFrames represent a distributed collection of data.
- b. RDDs can be converted into DataFrames.
- c. The existing data from a Hive table can be read as a DataFrame.
- d. DataFrames act like the entry point into all functionalities.



**ANSWERS:**

S.No.	Question	Answer & Explanation
1	Which of the following statement is true about “SaveMode.Append”?	d. For “SaveMode.Append”, content will be appended to the existing data. The other statements are true for “SaveMode.Ignore”, “SaveMode.Overwrite”, and “SaveMode.ErrorIfExists”.
2	Which of the following object has to be created for interacting with the Hive data?	a. A HiveContext object has to be used for querying data from a Hive table.
3	Which of the following statements are true about Spark SQL?	a. and b. Spark SQL provides hive compatibility, cost-based optimizer, code generations, and columnar storage. However, it does not require a different engine for the historical data and it mixes SQL queries with the Spark programs.
4	Which of the following statements are true about DataFrames?	a., b., and c. DataFrames represent a distributed collection of data and can be derived from RDDs. The existing data from a Hive table can be read as a DataFrame. However, the SQLContext class or any of its descendants acts like the entry point into all functionalities.

## Summary



Let us summarize the topics covered in this lesson:



- Spark SQL is a module of Spark that is used for structured data processing.
- DataFrames is a distributed collection of data, in which data is organized into columns that are named.
- The SQLContext class or any of its descendants acts like the entry point into all functionalities.
- To convert existing RDDs into DataFrames, Spark SQL supports two methods: reflection-based approach and programmatic approach.
- SaveModes specify the method of handling any existing data.
- You can use the saveAsTable command to save DataFrames as persistent tables.
- Parquet represents a columnar format supported by various data processing systems.

© Copyright 2015, Simplilearn. All rights reserved.

43

## Summary

Let us summarize the topics covered in this lesson:

- SparkSQL is a module of Spark that is used for structured data processing.
- DataFrames is a distributed collection of data, in which data is organized into columns that are names.
- The SQLContext class or any of its descendants acts like the entry point into all functionalities.
- To convert existing RDDs into DataFrames, SparkSQL supports two methods: reflection based approach and programmatic approach.
- SaveModes specify the method of handling any existing data.
- You can use the saveAsTable command to save DataFrames as persistent tables.
- Parquet represents a columnar format supported by various data processing systems.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- Spark SQL is capable of inferring a JSON dataset schema and loading it as a DataFrame and read and write data that is stored in an Apache Hive.
- You must create a HiveContext when you are working with Hive.
- To read data from various other databases, Spark SQL also contains a data source using JDBC.
- Spark SQL is compatible with SerDes, UDFs, and Hive Metastore.
- A few other supported hive features are joins, unions, sampling, explain, and more.
- The supported data types include string, binary, date, array, map, Boolean, struct, and more.
- Case classes represent the underneath data schema.

© Copyright 2015, Simplilearn. All rights reserved.

44

## Summary (contd.)

- Spark SQL is capable of inferring a JSON dataset schema and loading it as a DataFrame and read and write data that is stored in an Apache Hive.
- You must create a HiveContext when you are working with Hive.
- To read data from various other databases, SparkSQL also contains a data source using JDBC.
- Spark SQL is compatible with SerDes, UDFs, and Hive Metastore.
- A few other supported hive features are joins, unions, sampling, explain, and more.
- The supported data types include string, binary, date, array, map, Boolean, struct and more.
- Case classes represent the underneath data schema.

This concludes ‘Running SQL Queries Using Spark SQL.’

The next lesson is ‘Spark Streaming.’

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 4 “Running SQL Queries using SparkSQL” of the Apache Spark and Scala course. The next lesson is Spark Streaming.

## Lesson 5—Spark Streaming



## Objectives



After completing this lesson, you will be able to:

- Explain a few concepts of Spark Streaming
- Describe basic and advanced sources
- Explain how stateful operations work
- Explain window and join operations



© Copyright 2015, Simplilearn. All rights reserved.

2

## Objectives

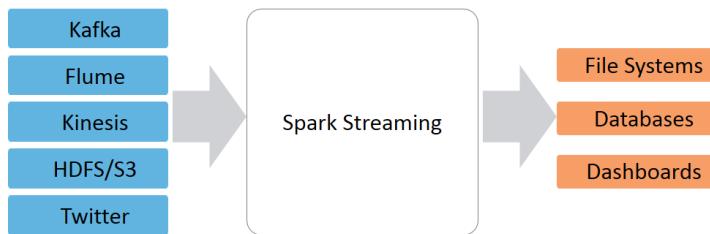
After completing this lesson, you will be able to:

- Explain a few concepts of Spark streaming
- Describe basic and advanced sources
- Explain how stateful operations work
- Explain window and join operations

## Introduction to Spark Streaming

Spark Streaming is the core Spark API's extension that allows:

- High-throughput, scalable, and fault-tolerant stream processing of live data streams
- TCP sockets processing using complex algorithms expressed using high-level functions such as join, reduce, map, and window
- Graph processing and machine learning algorithms application on data streams



© Copyright 2015, Simplilearn. All rights reserved.

3

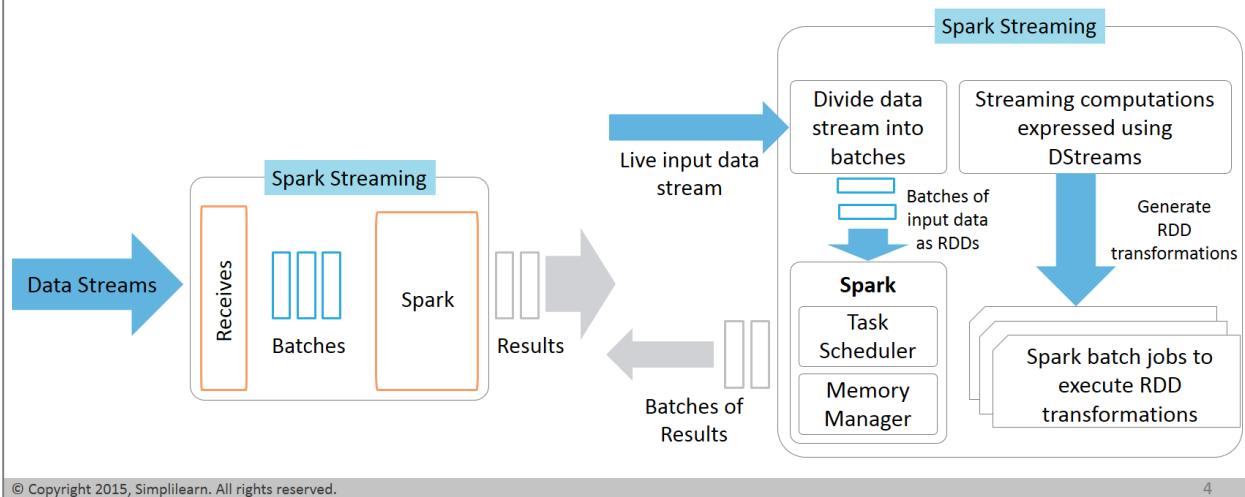
## Introduction to Spark Streaming

Let's first understand what Spark streaming is. It is the core Spark API's extension that allows high-throughput, scalable, and fault-tolerant stream processing of data streams that are live. There are many sources of ingesting data such as Flume, ZeroMQ, Twitter, Kinesis, and Kafka. Also, you can also process TCP sockets by the use of complex algorithms that are expressed using high-level functions such as join, reduce, map, and window. Once the data is processed, you can push it to databases, live dashboards, and file systems like S3 and HDFS. It is also possible to apply the graph processing and machine learning algorithms of Spark on data streams.

The image on the screen shows the data ingestion process and its output through Spark streaming.

## Working of Spark Streaming

Spark takes live input data streams and then divides them into batches. After this, the Spark engine processes those streams and generates the final stream results in batches.



© Copyright 2015, Simplilearn. All rights reserved.

4

## Working of Spark Streaming

Now, we will discuss how Spark streaming works.

It first takes live input data streams and then divides them into batches. After this, the Spark engine processes those streams and generates the final stream results in batches.

The working of Spark streaming is also shown graphically on the screen.

## Features of Spark Streaming

The features of Spark Streaming are listed below:

### Discretized Stream (DStream)

- Represents a continuous stream of data
- Can be created by either applying high-level operations on other DStreams or by using input data streams from sources like Flume, Kafka, and Kinesis

### Extensive Support

- Supports machine learning and graph processing algorithms
- Supports languages like Scala, Java, and Python

### ZooKeeper and HDFS

- Provides some state storage and leader election support
- Allows to launch multiple masters in your cluster connected to the same instance

© Copyright 2015, Simplilearn. All rights reserved.

5

## Features of Spark Streaming

Let's now talk about the features of Spark streaming.

Spark streaming provides a feature called discretized stream or DStream that is high-level abstraction. It represents a continuous stream of data and is represented as an RDD sequence internally. They can be created by either applying high-level operations on other Dstreams or by using input data streams from sources like Flume, Kafka, and Kinesis.

Spark streaming also supports machine learning and graph processing algorithms, and languages like Scala, Java, and Python.

For high availability in production, Spark streaming uses ZooKeeper and HDFS. ZooKeeper provides some state storage and leader election. Multiple masters can be launched in your cluster that are connected to the same instance of ZooKeeper. While others remain in the standby mode, one of them is elected as the leader. If the present leader dies, scheduling is resumed when another Master is elected and it recovers the state of the old master. This recovery process should complete between one and two minutes. The delay thus occurred affects only the new applications scheduling, while the already running applications remain unaffected.

## Streaming Word Count

**Example:**

```

import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))

val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()

```

**Streaming Word Count**

The code shown on the screen is an example to execute Spark streaming.

In this code, we are first importing the Spark Streaming classes' names along with a few implicit conversions from StreamingContext. This will add suitable and required methods to various other classes. StreamingContext acts as the key entrance point to all functionalities related to streaming. Here, a local StreamingContext is being created having two threads that can execute. The batch interval is of one second.

With the use of this code, we are creating a DStream representing streaming data. This data originates from a source of TCP that is defined as a port and host name.

This represents the data stream originated from the data server. Here, every record is a line of text.

With this code, the lines are being split into words with the use of space characters. An operation called flatMap is an operation of DStream that can create a new DStream. Being a one-to-many operation, it creates various new records from a single record existing in the source. Here, every line is split into various different words. The words stream is denoted by the wordsDStream. This is mapped to a DStream of pairs (word, 1).

The split words are being counted by this code. The words pairs are being compacted to obtain words frequency in every batch of data. The wordCounts.print() method is finally printing some counts generated per second.

## Micro Batch



Micro batching allows to:

- Handle a stream by a task/process as a sequence containing data chunks or small batches
- Pack events into various small batches and then delivered for processing to a batch system

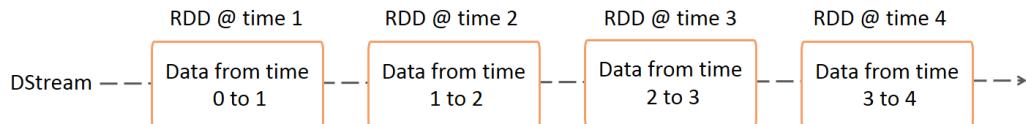
## Micro Batch

Micro batching is a core Spark technique that lets a task or process handle a stream as a sequence containing data chunks or small batches. In case of incoming streams, events can be packed into various small batches and then delivered for processing to a batch system.

**DStreams**

A DStream is:

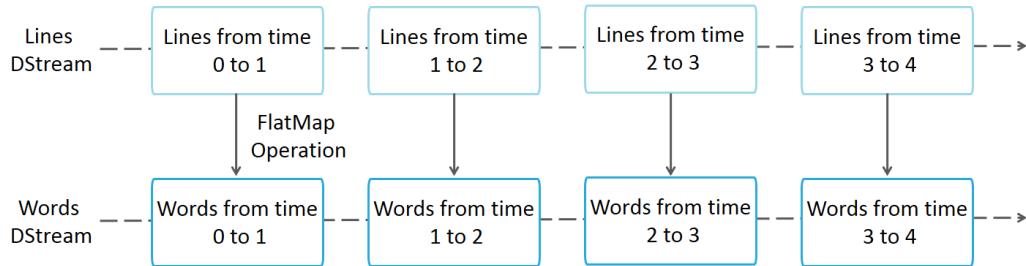
- Defined as the fundamental abstraction available from Spark Streaming
- Created by either applying high-level operations on other DStreams or by using input data streams
- Characterized through a series of continuous RDDs

**DStreams**

As discussed, DStream is defined as the fundamental abstraction that is available from Spark streaming. We have also discussed, that they can be created by either applying high-level operations on other Dstreams or by using input data streams. These streams are available from sources like Flume, Kinesis, and Kafka. Also, internally, it is characterized through a series of RDDs that is continuous. The image on the screen shows how every RDD that exists in a Dstream includes data related to a specific interval.

## DStreams (contd.)

All operations that you apply on a DStream get translated to operations applicable on the underlying RDDs.



© Copyright 2015, Simplilearn. All rights reserved.

9

## DStreams (contd.)

All operations that you apply on a DStream get translated to operations that are applicable on the underlying RDDs. As an example, recall that in the earlier example when a stream of lines was converted to words, to generate the RDDs of the “words DStream”, the flatMap operation was applied on each RDD in the “lines DStream”. This process is also shown through the given diagram.

Note that these underlying RDD transformations are performed by the engine of Spark. The operations of DStream provide you a higher-level API experience by hiding most of these details.

## Input DStreams and Receivers

Input DStreams represent the input data stream received from streaming sources. Except file stream, each input DStream is linked with a Receiver object that stores the data received from a source. Topologies of built-in streaming sources are:

### Basic Sources

Available in the StreamingContext API; **Examples:** socket connections, file systems

### Advanced Sources

Available from extra utility classes; **Examples:** Twitter, Flume, Kafka, Kinesis



You can create various input DStreams if it is required to receive various data streams in parallel in the streaming application. The application is required to be allocated sufficient cores for processing the received and running receivers.

© Copyright 2015, Simplilearn. All rights reserved.

10

## Input DStreams and Receivers

Input DStreams represent the input data stream that is received from sources of streaming. Except file stream, each input Dstream is linked with a Receiver object. This object stores the data received from a source in the memory of Spark for processing.

There are two topologies or categories of built-in streaming sources provided by Spark streaming: basic sources and advanced sources. Basic sources are available in the StreamingContext API directly. A few examples are socket connections and file systems. However, advanced sources include Twitter, Flume, Kafka, and Kinesis, and are available from extra utility classes. These sources need to be linked against extra dependencies.

Note that you can create various input DStreams in case it is required to receive various data streams in parallel in the streaming application. Doing so will create various receivers to receive multiple data streams in parallel. Spark worker or executor takes one of the cores that is allocated to the Spark streaming application. The reason is that it is a long-running task. Hence, you should remember that the application is required to be allocated sufficient cores for processing the received and running receivers.

**Points to Remember:**

- In case when a Spark Streaming program is run locally, do not use “local[1]” or “local” as the main or master URL; use “local[k]” as the main or master URL.
- The core number allocated to the application of Spark Streaming needs to be greater as compared to the receivers’ number.

## Input DStreams and Receivers (contd.)

You must remember that in case when a Spark Streaming program is run locally, you should not use “local[1]” or “local” as the main or master URL. If you do so, it would mean that a one thread is usable to locally run tasks. In case we use an input DStream on the basis of a receiver, then one thread would be used for running the receiver. This will not leave any thread to process the data received. Therefore, in such cases, you should every time use “local[k]” as the main or master URL. Here k is a number greater than the number of receivers to run.

Another important point is related to the logic extension to run on a cluster. The core number that is allocated to the application of Spark Streaming need to be greater as compared to the receivers’ number. If that does not happen, the system will not be able to process the received data

## Basic Sources



For basic sources, Spark streaming monitors the dataDirectory and processes all files created in it. The files must:

- Be in the same data format
- Be created by moving or renaming them atomically
- Not be altered once they are moved

**Syntax:** `streamingContextFileStream[KeyClass, ValueClass, InputFormatClass](dataDirectory)`

### Categories

- **Streams based on custom actors:** Allows to create DStreams with data streams received through Akka actors using `streamingContext.actorStream(actorProps, actor-name)`
- **Queue of RDDs as a stream:** Allows to create Dstreams using `streamingContext.queueStream(queueOfRDDs)`

## Basic Sources

Let us first talk about the first category of built-in streaming sources, basic sources.

For these sources, Spark streaming monitors the dataDirectory and processes all files that are created in it. The files in this directory must be in the same data format and must be created by moving or renaming them atomically. These files must not be altered once they are moved. The new data will not be read if the files are being appended continuously.

For simple text files, the given method is the easier one. File streams do not need allocating cores because they do not need running a receiver.

Basic sources can be divided into two categories, streams that are based on custom actors and queue of RDDs as a stream.

In case of streams based on custom actors, you can create DStreams with data streams that are received through Akka actors by using the given method. Note that in case of Python API, actors Stream is not available.

In case of queue of RDDs as a stream, you can also create a DStream using the given method to test a Spark streaming application. Every RDD that is pushed into the queue is processed like a stream and treated like a data batch in the DStream.

## Advanced Sources

Some advanced sources are:

**Kafka** (Spark Streaming 1.4.1 is compatible with Kafka 0.8.2.1.)

**Flume** (Spark Streaming 1.4.1 is compatible with Flume 1.4.0.)

**Kinesis**

**Twitter** (Utilities use Twitter4j 3.0.3 to get the public stream of tweets.)

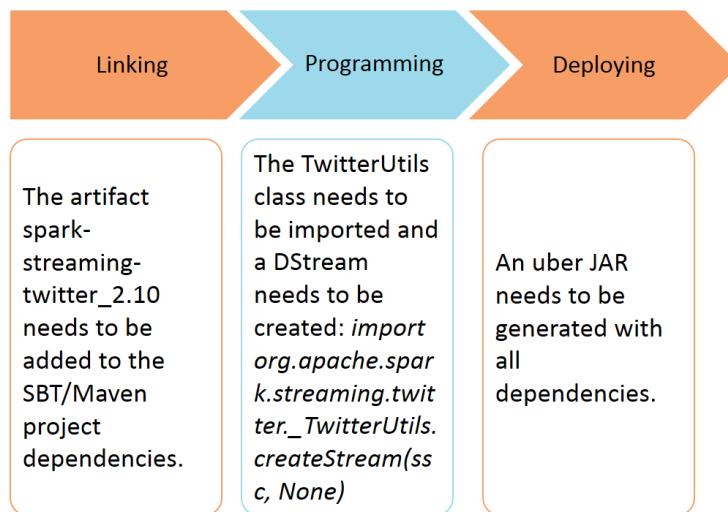
## Advanced Sources

The other type of built-in streaming sources is advanced sources. A few of these sources are listed on the screen. These include Kafka, Flume, Kinesis, and Twitter. Note that Spark Streaming 1.4.1 is compatible with Kafka 0.8.2.1, while Spark Streaming 1.4.1 is compatible with Flume 1.4.0.

In case of Twitter, Twitter utilities of Spark Streaming utilizes Twitter4j 3.0.3 for getting the public stream of tweets by the use of Streaming API of Twitter. The information required for authentication can be provided using any of the methods that is supported by the Twitter4J library. Either you can get the filtered stream on the basis of keywords or get the public stream.

## Advanced Sources—Twitter

To create a DStream using data from Twitter's stream of tweets, follow the steps listed below:



© Copyright 2015, Simplilearn. All rights reserved.

14

## Advanced Sources---Twitter

For example, consider that you want to create a DStream by the use of data from Twitter's stream of tweets. To do so, you would need to perform the steps listed on the screen.

The first step is linking, in which the artifact spark-streaming-twitter\_2.10 needs to be added to the SBT/Maven project dependencies. The second step is programming, in which the TwitterUtils class needs to be imported and a DStream with TwitterUtils.createStream needs to be created as shown on the screen.

The third step is deploying, in which an uber JAR needs to be generated with all dependencies. The application then needs to be deployed.

## Transformations on DStreams

Transformations on DStreams are similar to those of RDDs. A few of the common transformations on DStreams are given in the table below:

Transformation	Explanation
<code>map(func)</code>	Returns a new DStream by passing each element of the source DStream through a function func
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items
<code>filter(func)</code>	Returns a new DStream by selecting only the records of the source DStream on which func returns true
<code>repartition(numPartitions)</code>	Changes the level of parallelism in this DStream by creating more or fewer partitions
<code>union(otherStream)</code>	Returns a new DStream containing the union of the elements in the source DStream and other DStream
<code>count()</code>	Returns a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream

## Transformations on DStreams

The transformations on DStreams are similar to those of RDDs. They let the data from the input DStream to be altered. A few of the common transformations on DStreams are listed and explained through the given table.

## Transformations on Dstreams (contd.)

A few more common transformations on DStreams are given in the table below:

Transformation	Explanation
countByValue()	When called on a DStream of elements of type K, returns a new DStream of (K, Long) pairs, where the value of each key is its frequency in each RDD of the source Dstream
reduce( <i>func</i> )	Returns a new DStream of single-element RDDs by aggregating the elements in each RDD of the source Dstream using a function func that should be associative so that it can be computed in parallel
reduceByKey( <i>func</i> , [ <i>numTasks</i> ])	When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs, where the values for each key are aggregated using the given reduce function. <b>Note:</b> By default, this uses Spark's default number of parallel tasks to do the grouping. You can pass an optional numTasks argument to set a different number of tasks.
join( <i>otherStream</i> , [ <i>numTasks</i> ])	When called on two DStreams of (K, V) and (K, W) pairs, returns a new DStream of (K, (V, W)) pairs with all pairs of elements for each key

**Transformations on DStreams (contd.)**

A few more common transformations on DStreams are also listed and explained.

## Output Operations on DStreams

Output operations on Dstreams let the data of DStreams to be pushed to external systems. They trigger the real execution of all the DStream transformations. These are given in the table below:

Transformation	Explanation
print()	Prints first ten elements of every batch of data in a DStream on the driver node running the streaming application; useful for development and debugging
saveAsTextFiles(prefix, [suffix])	Saves this DStream's contents as text files; the file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".
saveAsObjectFiles(prefix, [suffix])	Saves this DStream's contents as a SequenceFile of serialized Java objects; the file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".
saveAsHadoopFiles(prefix, [suffix])	Saves this DStream's contents as a Hadoop file; the file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".
foreachRDD(func)	Applies a function, func, to each RDD generated from the stream; this function should push the data in each RDD to an external system, like saving the RDD to files, or writing it over the network to a database.

## Output Operations on DStreams

Let's now talk about the output operations on DStreams. These operations let the data of DStreams to be pushed to external systems such as file system or database. They act similar to RDDs in a way that they trigger the real execution of all the DStream transformations. The reason is that these operations let the transformed data be used by external systems. These operations are listed and explained through the given table.

## Design Patterns for Using ForeachRDD

`dstream.foreachRDD` is a powerful primitive that lets the data to be sent to external systems. It is explained as below:

- When writing data to an external system, try to use a connection object in a Spark worker for saving records in RDDs.
- Nothing will get executed if application does not include any output operation or includes operations such as `dstream.foreachRDD()` without any RDD action.
- Output operations, by default, are implemented one at a time and in the same order as they are defined in the application.

**Example:**

```
dstream.foreachRDD { rdd => val connection = createNewConnection() // executed at the driver
rdd.foreach { record => connection.send(record) // executed at the worker } }
```

## Design Patterns for Using ForeachRDD

`dstream.foreachRDD` is a powerful primitive that lets the data to be sent to external systems. But you must know the method of using it correctly and efficiently, as many common mistakes tend to occur.

Generally, when writing data to an external system, it needs to create a connection object and use it to send data to a remotely lying system. For this, you might try to create a connection object at the Spark driver unintentionally. Instead, you should try to use it in a Spark worker for saving records in RDDs.

The output operations execute DStreams lazily. To be specific, the RDD actions residing inside DStream output operations force to process the received data. Therefore, nothing will get executed in case your application does not include any output operation or includes operations such as `dstream.foreachRDD()` that do not have any RDD action inside them. In such cases, the system will just take the data and discard it.

Output operations, by default, are implemented one at a time. They are implemented in the same order as they are defined in the application. An example to use `dstream.foreachRDD` is given on the screen.

## DataFrame and SQL Operations

DataFrames and SQL operations can be easily used on the streaming data.

To use DataFrames and SQL operations

Create an SQLContext using the SparkContext being used by StreamingContext.

To allow restarting in case of driver failures

Create a lazily instantiated singleton instance of SQLContext.

## DataFrame and SQL Operations

DataFrames and SQL operations can be easily used on streaming data. For this, you would need to create an SQLContext by the use of the SparkContext that the StreamingContext is using. In addition, you need to do in a way that you may restart it in case of driver failures. For this, you would need to create a lazily instantiated singleton instance of SQLContext.

**Example:**

```
val words: DStream[String] = ...
words.foreachRDD { rdd =>
    // Get the singleton instance of SQLContext
    val sqlContext = SQLContext.getOrCreate(rdd.sparkContext)
    import sqlContext.implicits._
    val wordsDataFrame = rdd.toDF("word")
    // Register as table and Do word count on DataFrame using SQL and print it
    wordsDataFrame.registerTempTable("words")
    val wordCountsDataFrame =   sqlContext.sql("select word, count(*) as total from words group by word")
    wordCountsDataFrame.show() }
```

## DataFrame and SQL Operations (contd.)

For example, consider the given code that is modifying the earlier example of word count for generating word counts by the use of DataFrames and SQL. In this code, each RDD is being converted to a DataFrame, registered as a temporary table and finally queried by SQL.

A streaming application requires to operate in the 24x7 environment and therefore, must be resilient to failures. Hence, it requires to checkpoint sufficient information for a storage system that is fault-tolerant. It is of two types:

#### **Metadata Checkpointing**

- Usable for recovering from a node failure running the streaming application driver
- Has metadata that comprises of:
  - Streaming application configuration
  - Incomplete batches
  - DStream operations defining the streaming application

#### **Data Checkpointing**

- Usable in a few stateful transformations combining data across various batches
- Requires to checkpoint the RDDs related to stateful intermediate transformations on a periodic basis to an unfailing storage

## **Checkpointing**

A streaming application requires to operate in the 24.7 environment and therefore, it must be resilient to failures. Hence, it requires to checkpoint sufficient information for a storage system that is fault-tolerant, so that failure recovery can be performed. Checkpointing is of 2 kinds.

The first type is Metadata checkpointing. It used for recovering from a node failure that runs the driver of streaming applications. It is explained as saving information that defines the streaming computation to fault-tolerant storage such as HDFS. The related metadata comprises of configuration used for creating the application of streaming, and batches that incomplete with queued up jobs, and the operations of DStream defining the application of streaming.

The second type is Data checkpointing. It is explained as saving the generated RDDs to an unfailing storage. It is required in a few stateful transformations combining data across various batches. In such cases, the RDDs generated are dependent upon the RDDs of the earlier batches. This results in the increasing dependency chain length with time. For avoiding this, the RDDs related to stateful transformations that are intermediate are checkpointed on a periodic basis to an unfailing storage. It cuts off the dependency chains.

**When to enable checkpointing?**

It must be enabled for applications with any of the following requirements:

- Usage of stateful transformations
- Recovering from failures of the driver running the application

## Enabling Checkpointing

Now the question is when should checkpointing be enabled? You must enable in case of applications that have the requirements as listed. The first requirement is the use of stateful transformations. In case reduceByKeyAndWindow that is being used with the inverse function or updateStateByKey in an application, you must provide the checkpoint directory for allowing RDD checkpointing on a periodic basis.

Another requirement is to recover from driver failures that run an application. In such cases, you must implement the metadata checkpoints for recovering with the information on progress.

## Socket Stream

Socket stream is explained through the given points:

- The code residing outside the closure of the DStream is implemented in the driver, while the `rdd.foreach` method is implemented on every distributed RDD partition. Therefore, on the driver's machine, there is a socket created.
- The socket and computation are performed in the same host and hence it works.
- The nature of an RDD computation is distributed. Look for an alternative to have centralized access to distributed data. **Example:** Use Kafka.


**Example:**

```
crowd.foreachRDD(rdd => {rdd.collect.foreach(record=>{
    out.println(record)
})
})
```

## Socket Stream

The code that resides outside the closure of the DStream is implemented in the driver. However, the `rdd.foreach` method is implemented on every distributed RDD partition. Therefore, on the driver's machine, there is a socket created. The job attempts writing to it on another machine, which does not work for obvious reasons.

`DStream.foreachRDD` is implemented on the driver. In such cases, the socket and the computation are performed in the same host and hence it works.

The nature of an RDD computation is distributed. Therefore, this Server Socket method is hard to be implemented as dynamic service discovery is a difficulty. In such cases, you should look for an alternative so that you may have centralized access to distributed data. For example, you can use Kafka.

Consider the given example, which will work as it collects the RDD partitions from the workers and sends them to the driver to be written to socket.

## File Stream

**Example:**

```
streamingContext.fileStream[KeyClass, ValueClass, InputFormatClass](dataDirectory)  
streamingContext.fileStream<KeyClass, ValueClass, InputFormatClass>(dataDirectory);  
streamingContext.textFileStream(dataDirectory)
```

## File Stream

To read data files residing on any file system that is compatible with the HDFS API, you can create a DStream as shown on the screen.

## Stateful Operations



Stateful operations:

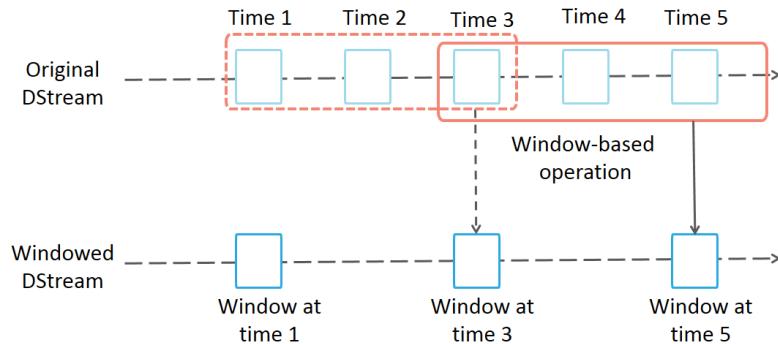
- Operate over various data batches
- Includes the updateStateByKey operation and all window-based operations
- Are dependent upon the earlier data batches

## Stateful Operations

Stateful operations are those operations that operate over various data batches. This includes the updateStateByKey operation and all window-based operations. These operations are dependent upon the earlier data batches. Therefore, they accumulate metadata over time continuously. For clearing this, Spark streaming saves intermediate data to HDFS and hence supports periodic checkpointing.

## Window Operations

Window operations let you implement transformations over a sliding window of data.



### Example:

```
val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10)) }
```

© Copyright 2015, Simplilearn. All rights reserved.

26

## Window Operations

Spark Streaming also supports window operations that let you implement transformations over a window of data that is sliding. This is illustrated through the given image.

As depicted, each time a window is sliding over a DStream source, the RDDs source falling within that particular window are being united. They are then being operated upon for producing the windowed DStream RDDs. In this case, slides are applied by two time units of data and the operation is applied over last three time units. It implies that all window operations require at least two parameters to be specified, which are window length and sliding interval. The window length is defined as the window duration, whereas the sliding interval is defined as the window operation interval at which it is being executed. Note that these parameters need to be the source batch interval multiples.

Consider an example in which you need to extend the previously discussed example. In addition, you need to generate word counts that have been completed over past 30 seconds of data in each 10 seconds. For this, you are required to use the reduceByKey method on the DStream (word, 1) pairs over this duration. To accomplish this, you would need to use the reduceByKeyAndWindow method, as shown on the screen.

## Types of Window Operations

Some of the general window operations are given in the table below. All these operations take window length and slide interval as parameters.

Operation	Explanation
<code>window(windowLength, slideInterval)</code>	Returns a new Dstream, which is computed based on windowed batches of the source
<code>countByWindow(windowLength,slideInterval)</code>	Returns a sliding window count of elements in the stream
<code>reduceByWindow(func, windowLength,slideInterval)</code>	Returns a new single-element stream, created by aggregating elements in the stream over a sliding interval using func, which should be associative so that it can be computed correctly in parallel
<code>reduceByKeyAndWindow(func,windowLength, slideInterval, [numTasks])</code>	When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> over batches in a sliding window

## Types of Window Operations

Some of the general window operations are listed and explained through the given table. Note that all these operations take the discussed parameters, window length and slide interval.

## Types of Window Operations Types (contd.)

A few more window operations are given in the table below:

Operation	Explanation
<code>reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])</code>	A more efficient version of <code>reduceByKeyAndWindow()</code> , where the reduce value of each window is calculated incrementally using the reduce values of the previous window; this is done by reducing the new data that enters the sliding window, and “inverse reducing” the old data that leaves the window. Note that checkpointing must be enabled for using this operation.
<code>countByValueAndWindow(windowLength, slideInterval, [numTasks])</code>	When called on a DStream of (K, V) pairs, returns a new DStream of (K, count) pairs where the value of each key is its frequency within a sliding window; like in <code>reduceByKeyAndWindow</code> , the number of reduce tasks is configurable through an optional argument.

## Types of Window Operations (contd.)

A few more window operations are also listed and explained.

## Join Operations—Stream-Stream Joins

The first type, stream-stream joins, allows to join streams with other streams.

**Example 1:**

```
val stream1: DStream[String, String] = ...
val stream2: DStream[String, String] = ...
val joinedStream = stream1.join(stream2)
```

**Example 2 (joining over windows of the streams):**

```
val windowedStream1 = stream1.window(Seconds(20))
val windowedStream2 = stream2.window(Minutes(1))
val joinedStream = windowedStream1.join(windowedStream2)
```

## Join Operations—Stream-Stream Joins

Spark streaming supports two types of join operations. The first one is stream-stream joins, which allows to join streams with other streams.

An example code to use it given on the screen. In this, every batch interval, which is the RDD generated by stream1 is being joined with the RDD generated by stream 2. leftOuterJoin, rightOuterJoin, and fullOuterJoin can also be used. In addition, it is also useful for joining over windows of the streams. An example is displayed.

## Join Operations—Stream-Dataset Joins

The second type, stream-dataset joins, allows to join a stream and a dataset.

**Example:**

```
val dataset: RDD[String, String] = ...
val windowedStream = stream.window(Seconds(20))...
val joinedStream = windowedStream.transform { rdd => rdd.join(dataset) }
```



The function that transforms is evaluated in each batch interval.

## Join Operations—Stream-Dataset Joins

The other type is stream-dataset joins, which allows to join a stream and a dataset. An example to use this join is displayed on the screen. It is also possible to change the dataset you need to join against.

Note that the function that transforms is evaluated in each batch interval.

## Monitoring Spark Streaming Application



Spark Web UI displays a streaming tab that shows the running receivers statistics and completed batches details to monitor the streaming application progress.

### Important Metrics:



- Processing Time: The time that it takes for processing every data batch
- Scheduling: The time a batch waits in a queue to process the earlier batches to complete

- If the batch processing time is continuously above the batch interval or the queue delay is increasing, you should reduce the batch processing time.
- You can also monitor a Spark Streaming program progress using the StreamingListener interface.

## Monitoring Spark Streaming Application

Spark streaming also provides additional features apart from Spark's monitoring features. The Web UI of Spark displays an additional streaming tab when a StreamingContext is used. This tab shows the running receivers statistics such as number of received records, active receivers, and receiver errors. It also shows completed batches details such as queueing delays and batch processing times. This information is usable to monitor the streaming application progress.

In this context, two metrics, which are processing time and scheduling time are important. Processing time represents the time that it takes for processing every data batch. However, scheduling data is the time a batch waits in a queue to process the earlier batches to complete.

In case the batch processing time is continuously above the batch interval or the queue delay is increasing, this implies that the system cannot process the data batches at the speed they are getting generated. In such cases, you should reduce the batch processing time.

You can also monitor a Spark streaming program progress using the StreamingListener interface. It lets you to receive processing times and receiver status. This is a developer API and hence, it would be improved upon in the future.

## Performance Tuning—High Level



To get the best performance from a Spark Streaming application on a cluster, you would need to tune it a bit. At a high level, you need to:

- Reduce the processing time of every data batch
- Set the correct batch size

## Performance Tuning—High Level

To get the best performance from a Spark Streaming application on a cluster, you would need to tune it a bit. At a high level, you need to reduce the processing time of every data batch by using cluster resources effectively and set the correct batch size so that the data batches can be processed as soon as they are received.

## Performance Tuning—Detail Level

At a detail level, you would need to consider the following parameters and configurations:

Data Serialization Overheads	You can reduce these overheads by tuning the serialization formats.
Task Launching Overheads	If the tasks number launched per second is high, the overhead of sending out tasks to slaves can be substantial. You can reduce it with the use of task serialization or by the execution mode.
Batch Interval	You should test it along with a low data rate and a conservative batch interval. Once you get a stable configuration idea, the data rate can be increased or the batch size can be reduced.
Memory Tuning	To perform an easy map-filter-store operation, low memory is required. You should see the use of memory on a small scale first and then estimate it.

## Performance Tuning---Detail Level

At a detail level, you would need to consider the listed parameters and configurations.

You can reduce data serialization overheads by tuning the serialization formats.

In case the tasks number is high that is launched per second, for example, more than 50, the overhead can be substantial in case of sending tasks to slaves. It makes achieving sub-second latencies difficult. In such cases, you can reduce the overhead with the use of task serialization, that is, with the use of Kryo serialization to serialize tasks that can ultimately decrease the sizes of tasks. It can also be reduced by the execution mode, as if you run Spark in the Mesos that is coarse-grained or standalone mode.

For performance tuning, you can also set the right interval of a batch. On the basis of the nature of the streaming computation nature, the interval can significantly affect the data rates, which the application can sustain on a fixed set of cluster resources. To analyze the correct batch size, a good method is to test it along with a low data rate and a conservative batch interval, for example, 5-10 seconds. For verifying if the system is capable of maintaining the data rate, you can check the end-to-end delay value that is experienced by every batch that is processed.

In case the delay can be compared with the batch size, it means that the system is in the stable state or else, if there is a continuous increase in the delay, this implies that the system is in the unstable state as it cannot keep up. Once you get a stable configuration idea, the data rate can be increased or the batch size can be reduced. You should note that a momentary delay increase because of the temporary data rate increase is alright until the value of delay gets reduced back to a value that is low.

Memory tuning is another configuration for performance tuning. The cluster memory amount that is needed by a Spark Streaming application is dependent upon the transformation type. For instance, consider a window operation is required to be used over the past ten minutes of data. In such cases, the cluster needs to contain enough memory that can handle ten minutes of data inside the memory. However, in case it is required to perform an easy map-filter-store operation, low memory would be required.

The received data is saved with the given storage level. Therefore, the data not fitting into memory spills over to the disk. It can reduce the streaming application performance. Therefore, you should use enough memory as needed by your application. As the best practice, you should see the use of memory on a small scale first and then estimate it. Garbage collection is another aspect of memory tuning. For application requiring low latency, large pauses are not desirable due to JVM Garbage Collection.

## Demo—Capture and Process the Netcat Data

This demo will show the steps to use Spark Streaming to capture and process the Netcat data.

### Demo---Capture and Process the Netcat Data

In this demo, you will learn how to use spark streaming to capture and process the netcat data.

First we are going to write a scala class “NetworkWordCount” to capture and process the NETCAT data.

This class will have a main method in which we are going to write the logic of capturing the incoming netcat data using the StreamingContext object. We will split the line into individual words and produce each word and 1 as output.

Here, we are using the StreamingContext Object with a batch interval of 1. This object will run as the listener service on the given IP address and port number.

Let's run the NetworkWordCount program by executing the command shown on the screen.

In another terminal, let's execute the “nc –lk 9977” command to start the “nc” client which connects with the NetworkWordCount streaming application and sends some data to it.

In the background terminal you can see that for each passed line such as “Word Count ”from “nc” client, the NetworkWordCount application generates output (Word, 1) , (Count, 1). Finally, For the “Apache Spark” incoming data , it produces (Apark, 1) and (Spark,1) as output.

## Demo—Capture and Process the Flume Data

This demo will show the steps to use Spark Streaming to capture and process the Flume data.

### Demo---Capture and Process the Flume Data

In this demo, you will learn how to use spark streaming to capture and process the flume data.

First we are going to write a scala class “NewsStremer” to capture and process the incoming Flume data. This class will have a main method in which we are going to write the logic of capturing the incoming flume data using the StreamingContext object. In this class, the case class will represent the structure of the incoming data from flume which is a JSON representation of the RSS feed data.

This class has a method “containFlu” which checks for the name of certain diseases in the incoming summary field of the RSS feed and returns true if those words are found.

In the main method, we are creating the instance of StreamingContext and starting a Flume receiver at the port number 44444 by using the “FlumeUtils.createStream” method. After that, we have written the logic of counting those disease names from the incoming data and printing them in the console . Finally, we are storing the output also in HDFS for historical reporting purposes.

You can see the sample RSS feed data here which contains category and summary fields.

Now, see the configuration file of Flume, which will be sending the RSS feed data to the Spark streaming receiver. In this file you can see that we have used the AVRO sink to dispatch the data to the Spark streaming receiver running on “localhost” port number 44444

Let's run the flume streaming application "NewsStreamer" by executing the command as shown in the screen.

In another terminal, let's run FLUME to send the RSS feed data to the Spark "NewsStreamer" application by running the command as shown on the screen.

From the output, you can see that the spark streaming application has found 2 diseases name in the incoming data as shown on the screen.

We can verify the output in HDFS as well as we have stored the output in HDFS. Let's type the command shown on the screen to check the content of the given file.

## Demo—Capture the Twitter Data



This demo will show the steps to use Spark Streaming to capture Twitter data.

© Copyright 2015, Simplilearn. All rights reserved.

36

## Demo—Capture the Twitter Data

In this demo, you will learn how to use spark streaming to capture the twitter data in a window operation of 10 and 60 seconds.

First we are going to write a scala class “TwitterPopularTags” to capture and process the incoming Twitter data. This class will have a main method in which we are going to write the logic of capturing the incoming twitter data using the StreamingContext object.

To Connect with Twitter, we need to register our application with Twitter and obtain consumerKey, consumerAccessToken, SecertToken and ConsumerSecretToken. Go to the given URL to register your application and receive these tokens.

We are going to use the TwitterUtils spark class to open a stream with Twitter. After that, we will parse the received tweets to take out the hashtag from them. Then, we are going to calculate the popular hashtags (topics) over sliding 10 and 60 second windows from a Twitter stream.

Finally, are going to print these 10 popular hashtags of 60 and 10 second windows in the console.

**QUIZ  
1**

Which of the following statement is true about the countByWindow operation?

- a. Returns a new Dstream, which is computed based on windowed batches of the source
- b. Returns a sliding window count of elements in the stream
- c. When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs
- d. Returns a new single-element stream, created by aggregating elements in the stream over a sliding interval using func

**QUIZ  
2**

Which of the following advance data sources are supported in Spark Streaming? *Select all that apply.*

- a. Kafka
- b. Kinesis
- c. Twitter
- d. Flume



QUIZ  
3

Which of the following statement is true about the saveAsObjectFile method?

- a. Saves a DStream's contents as a SequenceFile of serialized Java objects
- b. Saves a DStream's contents as text files
- c. Saves a DStream's contents as an Avro file in Hadoop
- d. Applies a function, func, to each RDD generated from the stream

QUIZ  
4

Which of the following statements are true about checkpointing? *Select all that apply.*

- a. Saves a streaming application from failures
- b. Must be enabled for applications with a set of requirements
- c. Has three types
- d. Usable in a few stateful transformations combining data across various batches

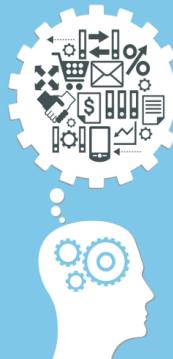


## ANSWERS

S.No.	Question	Answer & Explanation
1	Which of the following statement is true about the countByWindow operation?	b. The countByWindow operation returns a sliding window count of elements in the stream. The other statements are true for window, reduceByWindow, and reduceByKeyAndWindow operations respectively.
2	Which of the following advance data sources are supported in Spark Streaming?	a., b., c., and d. Spark Streaming supports all these advance data sources.
3.	Which of the following statement is true about the saveAsObjectFile method?	a. The saveAsObjectFile method Saves a DStream's contents as a SequenceFile of serialized Java objects. The other statements are true for saveAsTextFiles, saveAsHadoopFiles, and foreachRDD methods respectively.
4.	Which of the following statements are true about checkpointing?	a., b. and d. Checkpointing saves a streaming application from failures, must be enabled for applications with a set of requirements, and is usable in a few stateful transformations combining data across various batches. However, it has two types: metadata and data.

## Summary

Let us summarize the topics covered in this lesson:



- Spark Streaming is the core Spark API's extension that allows high-throughput, scalable, and fault-tolerant stream processing of data streams that are live.
- DStream is a high-level abstraction and represents a continuous stream of data and represented as an RDD sequence internally.
- Micro batching is a core Spark technique that lets a task or process handle a stream as a sequence containing data chunks or small batches.
- Input DStreams represent the input data stream that is received from sources of streaming.
- There are two topologies or categories of built-in streaming sources provided by Spark Streaming: basic sources and advanced sources.
- Transformations let the data from the input DStream be altered.
- Output operations let the data of DStreams be pushed to external systems such as file system or a database.

© Copyright 2015, Simplilearn. All rights reserved.

46

## Summary

- **Spark Streaming is the core Spark API's extension that allows high-throughput, scalable, and fault-tolerant stream processing of data streams that are live.**
- **DStream is a high-level abstraction and represents a continuous stream of data and represented as an RDD sequence internally.**
- **Micro batching is a core Spark technique that lets a task or process handle a stream as a sequence containing data chunks or small batches.**
- **Input DStreams represent the input data stream that is received from sources of streaming.**
- **There are two topologies or categories of built-in streaming sources provided by Spark Streaming: basic sources and advanced sources.**
- **Transformations let the data from the input DStream be altered.**
- **Output operations let the data of DStreams be pushed to external systems such as file system or a database.**

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- `dstream.foreachRDD` is a powerful primitive that lets the data be sent to external systems.
- DataFrames and SQL operations can be easily used on streaming data.
- A streaming application requires to checkpoint sufficient information for a fault-tolerant storage system.
- The socket and the computation are performed in the same host and hence it works.
- Stateful operations are those operations that operate over various data batches.
- Window operations that let you implement transformations over a sliding window of data.
- Spark Streaming supports two types of join operations: stream-stream joins and stream-dataset joins.
- The Web UI of Spark displays an additional streaming tab when a `StreamingContext` is used.
- To get the best performance from a Spark Streaming application on a cluster, you would need to tune it a bit.

**Summary**

- **`dstream.foreachRDD` is a powerful primitive that lets the data be sent to external systems.**
- **DataFrames and SQL operations can be easily used on streaming data.**
- **A streaming application requires to checkpoint sufficient information for a fault-tolerant storage system.**
- **The socket and the computation are performed in the same host and hence it works.**
- **Stateful operations are those operations that operate over various data batches.**
- **Window operations that let you implement transformations over a sliding window of data.**
- **Spark Streaming supports two types of join operations: stream-stream joins and stream-dataset joins.**
- **The Web UI of Spark displays an additional streaming tab when a `StreamingContext` is used.**
- **To get the best performance from a Spark Streaming application on a cluster, you would need to tune it a bit.**

simplilearn

This concludes ‘Spark Streaming’.

The next lesson is ‘Spark ML Programming’.

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 5 “Spark Streaming” of the Apache Spark and Scala course. The next lesson is Spark ML Programming.

## Lesson 6—Spark ML Programming



## Objectives



After completing this lesson, you will be able to:



- Explain the use cases and techniques of Machine Learning (ML)
- Describe the key concepts of Spark ML
- Explain the concept of an ML Dataset
- Discuss ML algorithm, model selection via cross validation

## Objectives

After completing this lesson, you will be able to:

- Explain the use cases and techniques of Machine Learning or ML
- Describe the key concepts of Spark ML
- Explain the concept of an ML Dataset, and
- Discuss ML algorithm, model selection via cross validation

**Machine Learning:**

- Is a sub field of Artificial Intelligence that has empowered various smart applications
- Deals with the construction and study of systems that can learn from data; **Examples:** Facebook photo album, Apple Siri, LinkedIn, and Google driverless car
- Lets a computer predict something

*"Machine Learning is a) field of study that gives computers the ability to learn without being explicitly programmed."* -- Arthur Samuel, 1959

*"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."* -- Tom Mitchell, 1997

## Introduction to Machine Learning

Let's first understand what machine learning is. It is a sub field of Artificial Intelligence that has empowered various smart applications. It deals with the construction and study of systems that can learn from data. For instance, consider the photo album feature of Facebook. This feature has the capability to learn from the data and hence recognize the faces that can be tagged in a picture. Similarly, the Siri application of Apple also has the capability to learn from the data and hence analyze the human voice meaning to perform the desired action or provide the desired answers. LinkedIn and Google driverless car also work on the same concept.

Therefore, the objective of Machine Learning is to let a computer predict something. An obvious scenario is to predict an event of the future. Apart from this, it also covers to predict unknown things or events. This means that something that has not been programmed or inputted in it. In other words, Computers act without being explicitly programmed. It can be seen as building blocks to make computers behave more intelligently. In the words of Arthur Samuel in 1959: "(Machine Learning is a) field of study that gives computers the ability to learn without being explicitly programmed".

Later, in 1997, Tom Mitchell gave another definition that proved more useful for engineering purposes, "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

## Common Terminologies in Machine Learning

A few common terminologies used in Machine Learning are:

Feature Vector	It is a typical setting, which is provided an objects or data points collection. Each item in this collection is described by a number of features such as categorical and continuous features.
Samples	They are the items to process. <b>Examples:</b> a row in a database, a picture, or a document
Feature Space	If a feature vector is a vector of length $l$ , each data point can be considered being mapped to a $d$ -dimensional vector space, called the feature space.
Labeled Data	It is the data with known classification results.



**Features:** Color (Radish/Red), Type (Fruit), Shape



**Features:** Color (Yellow), Type (Fruit), Shape

© Copyright 2015, Simplilearn. All rights reserved.

4

## Common Terminologies in Machine Learning

Before we move further, you should know the common terminologies used in Machine Learning like feature vector, feature space, samples, and labelled data.

Feature vector is defined as a typical setting that is done for machine learning, which is provided an objects or data points collection. Each item in this collection is described by a number of features. These features can be of different types. For example, they can be categorical like colors such as blue, black, and red, or they can be continuous such as integer-valued or real-valued. Features of a few of the items are displayed on the screen.

Samples are defined as the items to process, for example, a row in a database, a picture, or a document.

If a feature vector is a vector of length  $l$ , you can consider each data point being mapped to a  $d$ -dimensional vector space, which is referred as the feature space.

Another terminology is labelled data that is the data with known classification results.

## Applications of Machine Learning



Sometimes, Machine Learning is related with data mining. Pattern recognition and Machine Learning are also related. A few examples of the Machine Learning applications are:

Speech Recognition

Effective Web Search

Recommendation Systems

Computer Vision

Information Retrieval

Computational Finance

Fraud Detection

© Copyright 2015, Simplilearn. All rights reserved.

5

## Applicable of Machine Learning

Sometimes, Machine learning is related with data mining, however, it is more focused towards exploratory data analysis. In addition, pattern recognition and machine learning are also related and can be seen as two facets of the same area. A few examples of the machine learning applications are listed on the screen.

## Machine Learning in Spark

The scalable Machine Learning library of Spark is MLlib that contains general learning utilities and algorithms, including regression, collaborative filtering, classification, clustering, dimensionality reduction, and underlying optimization primitives.

**Types of API**

- **Primary API:** original spark.mllib
- **High-Level API:** "Pipelines" spark.ml

## Machine Learning in Spark

The scalable machine learning library of Spark is MLlib. It contains general learning utilities and algorithms, which include regression, collaborative filtering, classification, clustering, dimensionality reduction, and underlying optimization primitives. There are two types of API available. The primary API is original spark.mllib API and a higher-level API to construct Machine Learning workflows is Pipelines spark.ml.

## Spark ML API

The APIs for Machine Learning algorithms are standardized by Spark ML. The key concepts related to Spark ML API are listed below:

### ML Dataset

The Spark SQL DataFrame is used as a dataset that can contain various data types; **Example:** Different columns storing feature vectors, predictions, true labels, and text.

### Transformer

It is an algorithm that can transform one DataFrame into another; **Example:** An ML model can transform an RDD with features into an RDD with predictions.

### Estimator

It is another algorithm that can produce a transformer by fitting on a DataFrame; **Example:** A learning algorithm can train on a dataset and produce a model.

### Pipeline

It specifies an ML workflow by chaining various transformers and estimators together.

### Param

It is the common API to specify parameters for all transformers and estimators.

## Spark ML API

The APIs for machine learning algorithms are standardized by Spark ML. With this, it is easy to combine various algorithms in a single workflow or pipeline. The key concepts related to Spark ML API are listed on the screen.

The first concept is of an ML dataset. Spark machine learning utilizes the Spark SQL DataFrame as a dataset. It can contain various types of data types, for example, a dataset can contain different columns that store feature vectors, predictions, true labels, and text.

A transformer is defined as an algorithm that is capable of transforming one DataFrame into another. For instance, an ML model can transform an RDD with features into an RDD with predictions.

An estimator is another algorithm that can produce a transformer by fitting on a DataFrame. For instance, a learning algorithm can train on a dataset and produce a model.

A pipeline specifies an ML workflow by chaining various transformers and estimators together.

Param is the common API to specify parameters for all transformers and estimators.

We will talk about these concepts in more detail in the next screens.

## DataFrames



To support various data types under a unified Dataset concept, Spark ML includes the Spark SQL DataFrame. It supports:

- Basic types
- Structured types
- ML vector types



You can create a DataFrame from a regular RDD, either implicitly or explicitly.

© Copyright 2015, Simplilearn. All rights reserved.

8

## DataFrames

Let's first talk about DataFrames. Machine learning is applicable to various data types, which include text, images, structured data, and vectors. To support these data types under a unified Dataset concept, Spark ML includes the Spark SQL DataFrame.

These DataFrames provide support to various basic types, structured types, and ML vector types. You can create a DataFrame from a regular RDD, either implicitly or explicitly.

## Transformers and Estimators

Let's elaborate more on transformers and estimators.

**Transformer**

- Consists of learned models and feature transformers
- Is an abstraction and generally created by adding one or more columns
- **Example:** A feature transformer can take a dataset, read one of its columns, convert it into a new one, add it to the dataset, and output the updated dataset.
- Uses **transform()** to convert one DataFrame to another

**Estimator**

- Abstracts a learning algorithm concept or any other algorithm concept that trains or fits on data
- Uses **fit()** to accept a DataFrame and produce a transformer
- **Example:** The LogisticRegression learning algorithm estimator calls the fit() method and trains a LogisticRegressionModel, which is a transformer.

## Transformers and Estimators

Now, let's elaborate more on transformers and estimators.

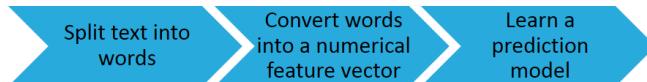
A transformer consists of learned models and feature transformers. It is an abstraction and generally created by adding one or more columns. For instance, a feature transformer can take a dataset, read one of its columns, convert it into a new one, add this new column to the original dataset, and then finally output the updated dataset. Technically, a transformer uses the transform() method to convert one DataFrame to another.

On the other hand, an estimator works by abstracting a learning algorithm concept or any other algorithm concept that trains or fits on data. From the technical standpoint, an estimator uses the fit () method to accept a DataFrame and produce a transformer. For instance, the LogisticRegression learning algorithm is an estimator that calls the fit () method and trains a LogisticRegressionModel, which is a transformer.

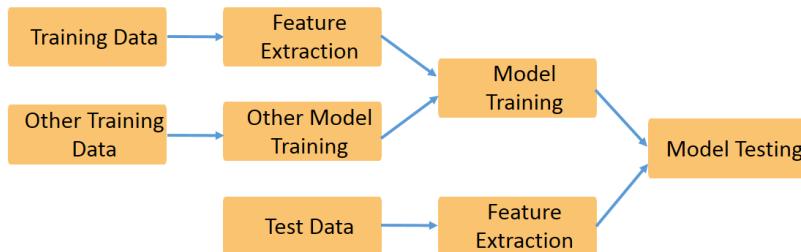
## Pipeline



Running an algorithms sequence for processing and learning from data is common in Machine Learning. The workflows such as below are represented through pipelines:



These pipelines include PipelineStages sequences, consisting of transformers and estimators, that are run in a specific order.



© Copyright 2015, Simplilearn. All rights reserved.

10

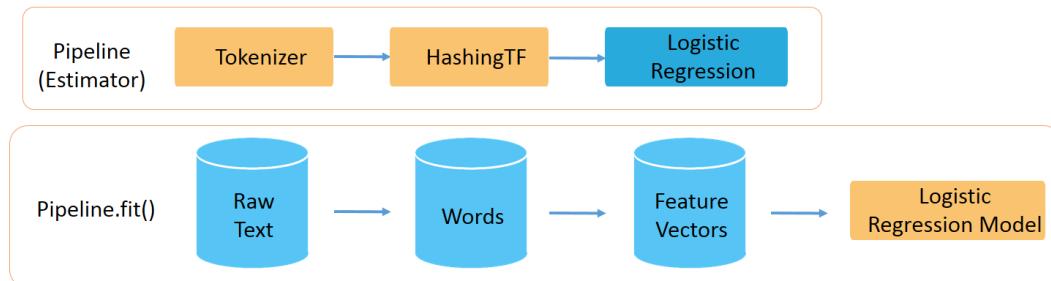
## Pipeline

The next concept of Spark ML API is pipeline. Running an algorithms sequence for processing and learning from data is common in machine learning. For instance, the process workflow of a simple text document includes the stages listed on the screen. First, the text of every document is split into words. Then, these words are converted into a numerical feature vector. And, at the final stage, the feature vectors and labels are used to learn a prediction model. Such workflows in Spark ML are represented through pipelines. These pipelines include PipelineStages sequences, consisting of transformers and estimators, that are run in a specific order.

A typical ML workflow is complex, such as depicted on the screen.

## Working of a Pipeline

A pipeline represents a sequence of stages that run in an order. The input dataset is altered as it passes through every stage. For the transformers stages, the `transform()` method is used, while for the estimators stages, the `fit()` method is used to create a transformer.



## Working of a Pipeline

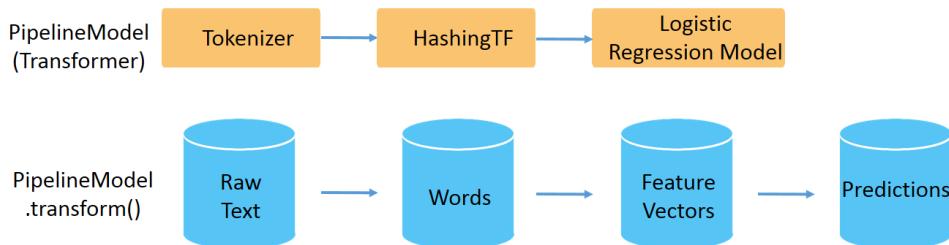
Let's now understand how a pipeline works. We have already discussed that a pipeline represents a sequence of stages, where every stage is a transformer or an estimator. All these stages run in an order and the dataset that is inputted is altered as it passes through every stage. For the stages of transformers, the `transform()` method is used, while for the stages of estimators, the `fit()` method is used to create a transformer. The transformer thus resulted becomes a part of the fitted pipeline or the `PipelineModel`. Let's understand this sequence with a simple text document workflow.

This pipeline has three stages, two of which, `Tokenizer` and `HashingTF`, are transformers, while the third, `LogisticRegression`, is an estimator. The row below it shows the flow of data through the pipeline, where the cylinders represent `DataFrames`.

The original dataset containing raw labels and text documents is being processed with the `Pipeline.fit()` method. The raw text documents are being split into words with the use of the `Tokenizer.transform()` method. This is adding a new column containing words into the dataset. The word column is being converted into feature vectors with the use of the `HashingTF.transform()` method, which is adding a new column containing these vectors to the dataset. Now, the pipeline first calls `LogisticRegression.fit()` to create the `LogisticRegressionModel` because `LogisticRegression` is an Estimator. In case it had more stages, it would have called the `transform()` method of `LogisticRegressionModel` on the dataset before passing the dataset to the next stage.

## Working of a Pipeline (contd.)

Now, the pipeline is also an estimator. Therefore, it produces a PipelineModel once the fit() method is run. The PipelineModel is a transformer and is used at the test time.

**Working of a Pipeline (contd.)**

Now, the pipeline is also an estimator. Therefore, it produces a PipelineModel once the fit () method is run. The PipelineModel is a transformer and is used at the test time. The image on the screen shows this use.

In this image, the PipelineModel contains the same number of stages as in the original pipeline. However, all estimators have become transformers. When a test dataset is processed with the transform () method of the PipelineModel, the data in an order through the pipeline. The transform () method in each stage updates the dataset and passes it. PipelineModels and pipelines make sure that the test and training data pass through similar feature processing steps.

## DAG Pipelines

In linear pipelines, each stage uses the data produced by the last one. However:

- It is also probable that a non-linear pipeline is created as long as the graph of data flow creates a Directed Acyclic Graph (DAG).
- The stages of a pipeline need to be specified in the topological order is the pipeline forms a DAG.



## DAG Pipelines

So far, we have discussed linear pipelines, in which each stage uses the data that is produced by the last one. However, it is also probable that a non-linear pipeline is created as long as the graph of data flow creates a Directed Acyclic Graph or DAG. Currently, such graphs are implicitly specified on the basis of the input and output column names of every stage. The stages of a pipeline need to be specified in the topological order is the pipeline forms a DAG.

## Runtime Checking



Pipelines can operate on datasets using various types. Therefore:

- Compile-time type checking cannot be used by them.
- PipelineModels and pipelines alternatively use runtime checking before running the pipeline and using the dataset schema.

## Runtime Checking

We have discussed that pipelines can operate on datasets using various types. Therefore, compile-time type checking cannot be used by them. PipelineModels and pipelines alternatively use runtime checking. This is done before running the pipeline and using the dataset schema.

## Parameter Passing

Param is the uniform API to specify parameters for estimators and transformers. It contains self-contained documentation and is a named parameter. A ParamMap represents a set of (parameter, value) pairs. To pass parameters to an algorithm, use any of the below two methods:

## Setting Parameters

Setting parameters for an instance; **Example:** `lr.setMaxIter(10)`

## Passing ParamMap

Passing a ParamMap to transform() or fit(). Here all parameters will override the parameters that have been formerly specified using setter methods.



Parameters are related to the specific instances of transformers and estimators.

© Copyright 2015, Simplilearn. All rights reserved.

15

## Parameter Passing

The last concept of Spark ML API is param, which is the uniform API to specify parameters for estimators and transformers. It contains self-contained documentation and is a named parameter. A ParamMap represents a set of (parameter, value) pairs. You can pass parameters to an algorithm using two methods.

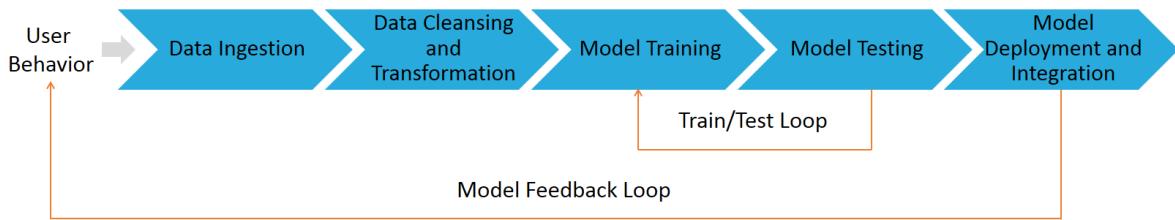
The first method is by setting parameters for an instance. For example, you can call the given method for LogisticRegression's instance lr. This will make `lr.fit()` use at most 10 iterations. This type of API is similar to the API used in MLlib.

The other method is by passing a ParamMap to transform () or fit (). All parameters in this ParamMap will override the parameters that have been formerly specified using setter methods.

Note that parameters are related to the specific instances of transformers and estimators.

## General Machine Learning Pipeline—Example

A Machine Learning pipeline includes various steps, as shown below:



## General Machine Learning Pipeline—Example

A machine learning pipeline includes various steps, as depicted in the diagram.

## General Machine Learning Pipeline—Example (contd.)

To test a model, perform the steps, as shown below:

Prepare test data using case classed

```
val training =
sc.parallelize(Seq(
LabeledPoint(1.0,
Vectors.dense(0.0, 1.1,
0.1)),
LabeledPoint(0.0,
Vectors.dense(2.0, 1.0, -1.0)),
LabeledPoint(0.0,
Vectors.dense(2.0, 1.3,
1.0)),
LabeledPoint(1.0,
Vectors.dense(0.0, 1.2, -0.5))))
```

Create a LogisticRegression instance

```
val lr = new
LogisticRegression()
lr.setMaxIter(10).setReg
Param(0.01)
```

Learn a LogisticRegression model

```
val model1 =
lr.fit(training.toDF)
```

Apply the model

```
model2.transform(test.t
oDF).select("features",
"label", "myProbability",
"prediction") .collect()
.foreach { case
Row(features: Vector,
label: Double, prob:
Vector, prediction:
Double) =>
println(s"$(features,
$label) -> prob=$prob,
prediction=$prediction") }
```

© Copyright 2015, Simplilearn. All rights reserved.

17

## General Machine Learning Pipeline—Example (contd.)

Further, to test a model, you need to perform the steps as listed on the screen. You need to first prepare training data. For this, you can use a case class called LabeledPoint. The RDDs of case classes are converted by Spark SQL into DataFrames. Here, it uses the metadata of the case class for inferring the schema.

Next, you need to create an estimator instance of LogisticRegression. You can use setter methods to set parameters.

After this, you need to learn a LogisticRegression model and then apply the model to make the prediction on the new incoming data with the use of the Transformer.transform() method.

## Model Selection via Cross-Validation



Model selection includes the use of data to figure out the best parameters/model for a task. Also termed as tuning. Pipelines facilitate model selection, it does not tune each element in the pipeline separately and makes tuning an entire pipeline in one go easy. Spark.ml uses the CrossValidator class that:

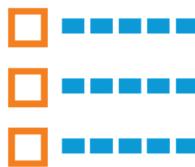
- Takes an evaluator, an estimator, and a set of ParamMaps, and splits the dataset into a folds set.  
**Example:** With three folds, it will generate three (training, test) dataset pairs
- Iterates through the set of ParamMaps
- Trains the specific estimator for each of the ParamMaps and evaluates it with the use of the specific evaluator
- Uses this best ParamMap and the complete dataset to fit the estimator

## Model Selection via Cross-Validation

Model selection is an important task in machine learning. It includes the use of data to figure out the best parameters or model for a task. It is also termed as tuning. Pipelines facilitate model selection, which does not tune each element in the pipeline separately and makes tuning an entire pipeline in one go easy.

At present, spark.ml uses the CrossValidator class to support model selection. This class takes an evaluator, an Estimator, and a set of ParamMaps. It splits the dataset into a folds set. These folds are used as different test and training datasets. For instance, with three folds, the class will generate three (training, test) dataset pairs. Each of these pairs utilizes 2/3 of the data for training and 1/3 of the data for testing. The class iterates through the set of ParamMaps. The class, for each of the ParamMaps, trains the specific estimator and evaluates it with the use of the specific evaluator. The ParamMap producing the best evaluation metric among all models is considered as the best model. Finally, the class uses this best ParamMap and the complete dataset to fit the estimator.

## Supported Types, Algorithms, and Utilities

**The types, algorithms, and utilities included in the Spark.MLLib are:**

- Data Types
- Basic Statistics
  - Summary statistics
  - Correlations
  - Stratified sampling
  - Hypothesis testing
  - Random data generation
- Classification and Regression
  - Linear models
  - Naive Bayes
  - Decision trees
  - Ensembles of trees
  - Isotonic regression
- Collaborative Filtering
  - Alternating Least Squares (ALS)
- Clustering
  - K-means
  - Gaussian mixture
  - Power Iteration Clustering (PIC)
  - Latent Dirichlet Allocation (LDA)
  - Streaming k-means
- Dimensionality Reduction
  - Singular Value Decomposition (SVD)
  - Principal Component Analysis (PCA)
- Feature Extraction and Transformation
- Frequent Pattern Mining
  - FP-growth
- Optimization
  - Stochastic gradient descent
  - Limited-memory BFGS (L-BFGS)
- PMML Model Export

## Supported Types, Algorithms, and Utilities

The types, algorithms, and utilities included in the spark.mllib, which is the main MLLib API, are listed on the screen. These are categorized under data types, basic statistics, classification and regression, collaborative filtering, clustering, dimensionality reduction, feature extraction and transformation, frequent pattern mining, and optimization. We will discuss these categories in detail in the next screens.

## Data Types

Mlib provides support to different data types listed below:

Local Vector	Includes 0-based indices, integer-based indices, and double-typed values that are saved on a single machine; Two types: sparse and dense
Labeled Point	A local vector related with a label/response; For classification and regression, use a double; For multiclass classification, use class indices starting from zero; For binary classification, use 0 or 1 indices.
Local Matrix	Includes double-typed values, and integer-typed row and column indices; Only dense matrices are supported; the related entry values are saved in a single double array in column major.

**Example:**

```
// Create a dense vector (1.0, 0.0, 3.0).
val dv: Vector = Vectors.dense(1.0, 0.0, 3.0) // Create a sparse vector (1.0, 0.0, 3.0) by specifying
// its indices and values corresponding to nonzero entries.
val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
```

© Copyright 2015, Simplilearn. All rights reserved.

20

## Data Types

Mlib provides support to different data types listed on the screen.

A local vector includes 0-based indices, integer-based indices, and double-typed values. These values are saved on a single machine. There are two types of local vectors supported by Mlib, which are sparse and dense. A sparse vector is backed up by two parallel arrays, which are values and indices. On the other hand, a dense vector is backed by a double array that represents its entry values.

A labeled point is also a local vector, but it is related with a label or a response. These are utilized in supervised learning algorithms in Mlib. To use labeled points in both classification and regression, a double is used to store a label. In case of multiclass classification, labels should be class indices that start from zero. However, in case of binary classification, these should be 0 or 1.

A local matrix includes double-typed values, and integer-typed row and column indices. Like local vectors, these values are also saved on a single machine. In this case, only dense matrices are supported by Mlib. The related entry values are saved in a single double array in column

## Feature Extraction and Basic Statistics

Term Frequency—Inverse Document Frequency (TF-IDF) is an easy way to generate feature vectors using text documents. It is calculated as two statistics: TF, the number of times the term appears in the particular document and IDF, the measure how often a term appears in the entire document corpus. The product of TD per times IDF signifies the relevance of a term with respect to a specific document.

**Example:**

```
val observations: RDD[Vector] = ... // an RDD of Vectors
// Compute column summary statistics.
val summary: MultivariateStatisticalSummary = Statistics.colStats(observations)
println(summary.mean) // a dense vector containing the mean value for each column
println(summary.variance) // column-wise variance
println(summary.numNonzeros) // number of nonzeros in each column
```

## Feature Extraction and Basic Statistics

Term Frequency—Inverse Document Frequency or TF-IDF is defined an easy way for generating feature vectors using text documents such as web pages. It is calculated as two statistics for every term in every document: TF that is defined as the number of times the term appears in the particular document and IDF that is defined as the measure how often a term appears in the entire document corpus. The product of TD per times IDF signifies the relevance of a term with respect to a specific document.

The code given on the screen shows how to compute column summary statistics.

## Clustering



Clustering is an unsupervised learning problem with the objective to group the entities subsets on the basis of some idea of similarity. It is generally used as a hierarchical supervised learning pipeline component or for exploratory analysis. MLlib supports various models of clustering, which are:

- K-means
- Gaussian mixture
- Power Iteration Clustering (PIC)
- Latent Dirichlet Allocation (LDA)
- Streaming k-means

## Clustering

Clustering is defined as an unsupervised learning problem in which the objective is to group the entities subsets on the basis of some idea of similarity. It is generally used as a hierarchical supervised learning pipeline component or for exploratory analysis. MLlib supports various models of clustering, which include K-means, Gaussian mixture, Power iteration clustering or PIC, Latent Dirichlet allocation or LDA, and Streaming k-means.

We will discuss about these models in detail in the next screens.

## K-Means



K-means works by clustering the data points into a predefined clusters number. A parallelized variant of the K-means++ method is also included, `Kmeans||`.

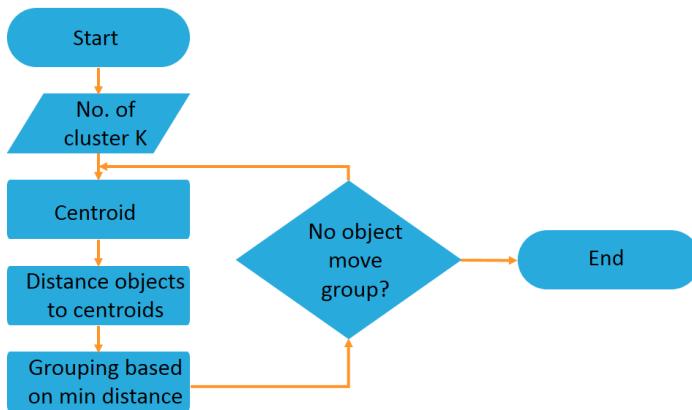
Parameters	Explanation
<i>K</i>	The number of required clusters
<i>maxIterations</i>	The maximum number of iterations to run
<i>initializationMode</i>	Random initialization or initialization via the K-means   method
<i>runs</i>	The number of times to run the K-means algorithm
<i>initializationSteps</i>	The number of steps in the K-means algorithm
<i>epsilon</i>	The distance threshold within which you consider K-means to have converged

## K-Means

K-means works by clustering the data points into a predefined clusters number. It is one of the most generally used clustering algorithms. A parallelized variant of the k-means++ method is also included in the MLlib implementation, which is given on the screen. The MLlib implementation includes the listed parameters. *K* is defined as the number of required clusters, *maxIterations* is defined as the maximum number of iterations to run, and *initializationMode* specifies random initialization or initialization via the given k-means method. *Runs* is defined as the number of times to run the k-means algorithm, *initializationSteps* defines the number of steps in the k-means algorithm, while the last parameter *epsilon* signifies the distance threshold within which you consider k-means to have converged.

## K-Means (contd.)

The flowchart below shows how the K-means algorithm works:



© Copyright 2015, Simplilearn. All rights reserved.

24

## K-Means (contd.)

The flowchart on the screen shows how the K-means algorithm works.

## Demo—Perform Clustering Using K-Means

This demo will show the steps to perform clustering using K-means in Spark.

## Demo—Perform Clustering Using K-Means

In this demo, you will learn how to perform clustering, which is also known as un-supervised learning using the K-Means technique in Spark.

We are going to run this example in spark-shell.

Before we write our logic we need to import Kmeans, KmeansModel, and Vectors classes into the spark-shell. Note the three import statements being shown on the screen.

Let's read the kmeans sample data from HDFS using the sc.textFile() method.

We are going to convert this data into double and cache them into memory.

Let's define parameters such as number of clusters and number of iterations

Let's use Kmeans to train the data by the train method of the Kmeans class and by passing data and other parameters to it.

We compute the WSSE to evaluate the goodness of fit for the training data.

You can see that WSSE for Kmeans is 0.12 for the given data.

## Gaussian Mixture



A Gaussian mixture model signifies a compound distribution, where points are drawn from one of the k Gaussian sub-distributions. Each of these sub-distributions has its own probability. The expectation-maximization algorithm is used by the MLlib implementation for inducing the maximum-likelihood model given a samples set.

Parameters	Explanation
<i>K</i>	The number of required clusters
<i>convergenceTol</i>	The maximum change in log-likelihood at which you think convergence is achieved
<i>maxIterations</i>	The maximum number of iterations for performing without reaching convergence
<i>initialModel</i>	An optional starting point from which the EM algorithm is started

## Gaussian Mixture

A Gaussian mixture model signifies a compound distribution. In this, points are drawn from one of the k Gaussian sub-distributions. Each of these sub-distributions has its own probability. The expectation-maximization algorithm is used by the MLlib implementation for inducing the maximum-likelihood model given a samples set. The parameters of the implementation are listed on the screen. K is defined as the number of required clusters and convergenceTol is defined as the maximum change in log-likelihood at which you think convergence is achieved. maxIterations is defined as the maximum number of iterations for performing without reaching convergence, while initialModel is an optional starting point from which the EM algorithm is started. In case this parameter is deleted, a random starting point is constructed from data.

## Power Iteration Clustering (PIC)

PIC is an efficient and scalable algorithm used to cluster a graph vertices. It uses power iteration to compute a pseudo-eigen vector of the graph normalized affinity matrix for clustering vertices. It outputs a model along with the clustering assignments by taking an RDD of tuples, srclId, dstId, and similarity. In the input data, a pair should appear at most once, irrespective of ordering.

Parameters	Explanation
<i>K</i>	The number of required clusters
<i>maxIterations</i>	The maximum number of iterations
<i>initializationModel</i>	The initialization model, which can be default “random” for using a random vector as vertex properties or “degree” for using normalized sum similarities

## Power Iteration Clustering (PIC)

PIC is an efficient and scalable algorithm that is used to cluster a graph vertices, in which the graph is provided pairwise similarities as edge properties. It uses power iteration to compute a pseudo-eigen vector of the graph normalized affinity matrix for clustering vertices. At the backend, MLlib has a PIC implementation using GraphX. It outputs a model along with the clustering assignments by taking an RDD of tuples. These tuples are of srclId, dstId, and similarity, where similarities need to be non-negative.

The similarity measure is assumed symmetric by PIC. In the input data, a pair should appear at most once, irrespective of ordering. In case a pair missing from the input, the related similarity is considered as 0. The MLlib implementation includes the listed parameters.

*K* is defined as the number of clusters, while *maxIterations* is the maximum number of power iterations, *initializationModel* is the initialization model, which can be default “random” for using a random vector as vertex properties or “degree” for using normalized sum similarities.

## Latent Dirichlet Allocation (LDA)



LDA works by inferring topics from a text documents collection. It is a topic model and can be considered as a clustering algorithm as depicted below:

- Topics are related to cluster centers and documents are related to rows or examples in a dataset.
- Both topics and documents reside in a feature space.
- Instead of using a traditional distance to estimating a clustering, LDA utilizes a function based on a statistical model of documents generation.

© Copyright 2015, Simplilearn. All rights reserved.

28

## Latent Dirichlet Allocation (LDA)

LDA works by inferring topics from a text documents collection. It is a topic model and can be considered as a clustering algorithm as given on the screen.

Topics are related to cluster centers and documents are related to rows or examples in a dataset. Both topics and documents reside in a feature space. Here, feature vectors are word counts vectors. Instead of using a traditional distance to estimating a clustering, LDA utilizes a function that is based on a statistical model of documents generation.

## Latent Dirichlet Allocation (LDA) (contd.)

LDA provides support to various inference algorithms via the `setOptimizer` function. While `OnlineLDAOptimizer` is generally memory friendly and utilizes iterative mini-batch sampling for online variational inference, `EMLDAOptimizer` utilizes expectation-maximization on the likelihood function to learn clustering and provide comprehensive results.

Parameters	Explanation
<code>K</code>	The number of topics
<code>maxIterations</code>	The limit on the number of iterations of EM used for learning
<code>docConcentration</code>	A Hyperparameter used for prior over distributions of documents over topics
<code>topicConcentration</code>	A Hyperparameter used for prior over distributions of topics over terms or words
<code>checkpointInterval</code>	The frequency with which checkpoints are created

**Latent Dirichlet Allocation (LDA) (contd.)**

LDA provides support to various inference algorithms via the `setOptimizer` function. While `OnlineLDAOptimizer` is generally memory friendly and utilizes iterative mini-batch sampling for online variational inference, `EMLDAOptimizer` utilizes expectation-maximization on the likelihood function to learn clustering and provide comprehensive results.

LDA provides topics and topic distributions for documents after fitting on the documents. The parameters taken by LDA are listed on the screen. `K` is defined as the number of topics, that is, cluster centers, and `maxIterations` is defined as the limit on the number of iterations of EM used for learning. `docConcentration` is a Hyperparameter used for prior over distributions of documents over topics. At present, it needs to be greater than 1, where for smoother inferred distributions, larger values should be used. `topicConcentration` is also a Hyperparameter used for prior over distributions of topics over terms or words. At present, it needs to be greater than 1, where for smoother inferred distributions, larger values should be used.

In case checkpointing is being used, `checkpointInterval` denotes the frequency with which checkpoints are created. Using checkpointing can help in reducing the shuffle file sizes on disk if `maxIterations` is large. It can also help with failure recovery.

You should note that LDA is a new feature and hence some functionality is missing. To be specific, it does not provide support to prediction on new documents yet. In addition, it does not include a Python API.

## Collaborative Filtering

Collaborative filtering is generally used for recommender systems. The objective of these techniques is to complete a user-term association matrix entries that are missing. At present, MLlib provides support to model-based collaborative filtering. In this, products and users are explained through a small latent factors set.

Parameters	Explanation
<i>numBlocks</i>	The number of blocks that are used for parallelizing computation
<i>rank</i>	The number of latent factors in the model
<i>Iterations</i>	The number of iterations to run
<i>lambda</i>	The parameter of regularization in ALS
<i>implicitPrefs</i>	Defines what should be used out of the variant adapted for implicit feedback data or the explicit feedback ALS variant
<i>alpha</i>	A parameter that is applicable to the implicit feedback ALS variant governing the baseline confidence in preference observation

## Collaborative Filtering

Collaborative filtering is generally used for recommender systems. The objective of these techniques is to complete a user-term association matrix entries that are missing. At present, MLlib provides support to model-based collaborative filtering. In this, products and users are explained through a small latent factors set. These factors are usable for predicting the missing entries. To learn these factors, MLlib utilizes the alternative least squares or ALS algorithm. The MLlib implementation includes the parameters listed on the screen.

*numBlocks* is defined as the number of blocks that are used for parallelizing computation and *rank* is defined as the number of latent factors in the model. *Iterations* is defined as the number of iterations to run, while *lambda* defines the parameter of regularization in ALS. *implicitPrefs* defines what should be used out of the variant adapted for implicit feedback data or the explicit feedback ALS variant. *alpha* is a parameter that is applicable to the implicit feedback ALS variant governing the baseline confidence in preference observation.

## Classification



MLlib provides support to different regression analysis, binary classification, and multiclass classification methods. The table below lists and explains the supported algorithms for every problem type:

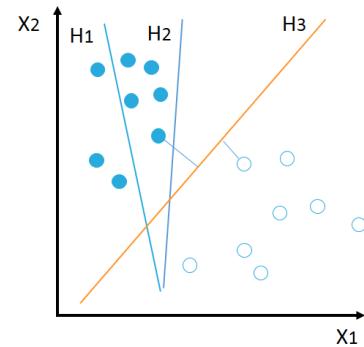
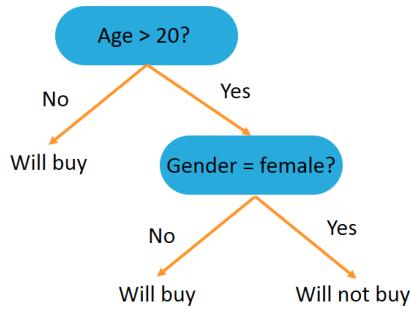
Problem Type	Supported Methods
Binary Classification	Linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes
Multiclass Classification	Logistic regression, decision trees, random forests, naive Bayes
Regression	Linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

## Classification

MLlib provides support to different regression analysis, binary classification, and multiclass classification methods. The table shown on the screen lists and explains the supported algorithms for every problem type.

## Classification (contd.)

Two examples of classification are shown below:



© Copyright 2015, Simplilearn. All rights reserved.

32

## Classification (contd.)

Two examples of classification are shown on the screen.

## Regression

In linear regression, multiple procedures have been developed for parameter inference and estimation. These procedures vary in terms of:

- Presence of a closed-form solution
- Robustness in terms of heavy-tailed distributions
- Computational simplicity of algorithms
- Theoretical assumptions required for validating the desirable statistical properties



One of the statistical problem-solving method is linear least squares.

## Regression

In linear regression, multiple procedures have been developed for the purpose of parameter inference and estimation. These procedures vary in terms of presence of a closed-form solution, robustness in terms of heavy-tailed distributions, and computational simplicity of algorithms. They also differ with respect to theoretical assumptions that are required for validating the desirable statistical properties like asymptotic efficiency and consistency.

One of the statistical problem-solving method is linear least squares. This allows to increase the accuracy of solution approximation with respect to the complexity of a specific problem.

## Example of Regression

For regression problems, linear least squares is the most general formulation. With the use of various regularization types, different related regression methods are derived. No regularization is used by the general least squares or linear least squares. At the back, a simple distributed version of stochastic gradient descent or SGD is implemented by MLlib. All the algorithms provided take a regularization parameter and SGD related parameters.

**Example:**

```
val points: RDD[LabeledPoint] = // ...
val lr = new
LinearRegressionWithSGD().setNumIterations(200).setIntercept(true)
val model = lr.run(points)
println("weights: %s, intercept: %s".format(model.weights, model.intercept))
```

## Example of Regression

For regression problems, linear least squares is the most general formulation. With the use of various regularization types, different related regression methods are derived. No Regularization is used by the general least squares or linear least squares. While Lasso uses L1 regularization, ridge regression uses L2 regularization. The training error or average loss of all these models is termed as the mean squared error. At the back, a simple distributed version of stochastic gradient descent or SGD is implemented by MLlib. It builds on the gradient descent primitive underlying. All the algorithms provided take a regularization parameter as an input. These also take different parameters that are related with SGD.

The code given on the screen shows how to implement linear regression.

## Demo—Perform Classification Using Linear Regression

This demo will show the steps to perform classification using linear aggression in Spark.

## Demo---Perform Classification Using Linear Regression

In this demo, you will learn how to perform classification, also known as supervised learning, using linear regression in Spark.

First, we are going to write a Scala class “LinearRegression”, which will have case class called Params for holding variables such as the number of iterations.

Using OptionParser will parse the Params and assign these variables to the scala option data type.

In the run method, we are going to load the sample data and split it into the test and training data. After that, we are going to perform data testing to build a model using the least square technique.

Now, we are going to use LinearRegressionWithSGD to build a simple linear model to predict the label values. Here, we are setting all the required parameters such as number of iterations, stepsize, updater and reg param.

Once we have a model created by running the algorithm, we can use that for doing the prediction for the test data.

We compute the mean squared error to evaluate the goodness of fit for the training data.

See the sample data going to be used for this example.

## Demo—Run Linear Regression

This demo will show the steps to run linear regression in Spark.

© Copyright 2015, Simplilearn. All rights reserved.

36

### Demo---Run Linear Regression

In this demo, you will learn how to run linear regression in spark.

You need to go to the execute command as shown on the screen to execute this program. Type the given Class name including the given package name.

Note that if the file is local file, you need to prefix the input file path as shown.

Once you execute this command, spark mLib program will get executed.

As you can see on the screen, it has used 405 training datasets and 96 testing datasets.

Test RMSE, which is an indicator of effectiveness of this algorithm, is 10.22 as shown on the screen.

## Demo—Perform Recommendation Using Collaborative Filtering

This demo will show the steps to perform recommendation.

## Demo—Perform Recommendation Using Collaborative Filtering

In this demo, you will learn how to perform recommendation using collaborative filtering (Alternating Least Squares algorithm) in Spark.

Collaborative Filtering or CF is a subset of algorithms that exploits other users and items along with their ratings and target user history to recommend an item that the target user does not have ratings for.

First, we are going to write a scala class “MovieLensALS” class which will have the case class called Params for holding variables such as number of iterations.

Using OptionParser will parse the Params and assign these variables to the scala option data type.

In the run method, we are going to load the sample data and split it into the test and training data. After that we are going to perform data testing to build a model using the least square technique.

Now, we are going to use Alternating Least Squares to build a recommendation system. Here, we are setting all the required parameters such as rank, number of iterations, lambda, ImplicitPrefs, UserBlocks, and ProductBlocks.

We compute the Root Mean Squared Error to evaluate the goodness of fit for the training data.

Once we have a model created by running the algorithm, we can use that for doing the prediction for the test data.

## Demo—Run Recommendation System



This demo will show the steps to run the recommendation system in Spark.

© Copyright 2015, Simplilearn. All rights reserved.

38

## Demo—Run Recommendation System

In this demo, you can learn how to run the recommendation system in Spark to recommend movies to users.

You need to go to the execute command as shown on the screen to execute this program. Type the given Class name including the given package name.

Note that if the file is local, you need to prefix the input file path as shown.

Once you execute this command, spark MLlib program will get executed .

As you can see on the screen, it has used 1501 ratings from 30 users on 100 movies.

Out of 1501 ratings datasets, it has taken 1199 for training and rest 302 for testing the model.

Test RMSE which is an indicator of effectiveness of this algorithm is 1.47 as shown on the screen.-

**QUIZ****1**

Which of the following statement is true about transformers in the Spark MLlib pipeline?

*Select all that apply.*

- a. A transformer uses a DataFrame as a dataset.
- b. A transformer is an abstraction that includes feature transformers and learned models.
- c. A transformer produces an estimator.
- d. A transformer implements the transform() method.

**QUIZ****2**

Which of the following Machine Learning libraries are supported in Spark? *Select all that apply.*

- a. Spark.MLlib
- b. Spark.ml
- c. Spark.mlearning
- d. Spark.Mlib



**QUIZ  
3**

A Spark pipeline's stages are specified as which of the following object?

- a. List
- b. Array
- c. Map
- d. Table

**QUIZ  
4**

For which of the Machine Learning technique collaborative filtering is used?

- a. Classification
- b. Clustering
- c. Recommendation
- d. Collaboration



## ANSWERS

S.No.	Question	Answer & Explanation
1	Which of the following statement is true about transformers in the Spark MLLib pipeline?	a., b., and d. A transformer uses a DataFrame as a dataset, is an abstraction that includes feature transformers and learned models, and implements the transform() method. It does not produce an estimator, but an estimator produces a transformer.
2	Which of the following Machine Learning libraries are supported in Spark?	a. and b. Both Spark.MLLib and Spark.ml libraries are supported in Spark.
3	A Spark pipeline's stages are specified as which of the following object?	b. Stages are specified as an array in a Spark pipeline.
4	For which of the Machine Learning technique collaborative filtering is used?	c. Collaborative filtering is used in recommendation.

## Summary

Let us summarize the topics covered in this lesson:



- Machine Learning is a sub field of Artificial Intelligence that has empowered various smart applications.
- Some terminologies used in ML are feature vector, feature space, samples, and labelled data.
- The scalable Machine Learning library of Spark is MLlib.
- Spark ML includes the Spark SQL DataFrame to support ML data types.
- A transformer consists of learned models and feature transformers.
- Workflows in Spark ML are represented through pipelines.
- A non-linear pipeline can also be created as long as the graph of data flow creates a DAG.
- Param is the uniform API to specify parameters for estimators and transformers.
- Model selection includes the use of data to figure out the best parameters or model for a task.
- MLlib supports data types including local vector, labeled point, and local matrix.
- TF-IDF is defined an easy way for generating feature vectors using text documents such as web pages.

© Copyright 2015, Simplilearn. All rights reserved.

48

## Summary

Let us summarize the topics covered in this lesson:

- Machine Learning is a sub field of Artificial Intelligence that has empowered various smart applications.
- Some terminologies used in ML are feature vector, feature space, samples, and labelled data.
- The scalable machine learning library of Spark is MLlib.
- Spark ML includes the Spark SQL DataFrame to support ML data types.
- A transformer consists of learned models and feature transformers.
- Workflows in Spark ML are represented through pipelines.
- A non-linear pipeline can also be created as long as the graph of data flow creates a DAG.
- Param is the uniform API to specify parameters for estimators and transformers.
- Model selection includes the use of data to figure out the best parameters or model for a task.
- MLlib supports data types including local vector, labeled point, and local matrix.
- TF-IDF is defined an easy way for generating feature vectors using text documents such as web pages.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- Clustering is defined as an unsupervised learning problem in which the objective is to group the entities subsets on the basis of some idea of similarity.
- K-means works by clustering the data points into a predefined clusters number.
- A Gaussian mixture model signifies a compound distribution.
- PIC is an efficient and scalable algorithm that is used to cluster a graph vertices.
- LDA works by inferring topics from a text documents collection.
- Collaborative filter is used to complete a user-term association matrix entries that are missing.
- MLlib provides support to different regression analysis, binary classification, and multiclass classification methods.
- In linear regression, multiple procedures have been developed for the purpose of parameter inference and estimation.

**Summary**

- Clustering is defined as an unsupervised learning problem in which the objective is to group the entities subsets on the basis of some idea of similarity.
- K-means works by clustering the data points into a predefined clusters number.
- A Gaussian mixture model signifies a compound distribution.
- PIC is an efficient and scalable algorithm that is used to cluster a graph vertices.
- LDA works by inferring topics from a text documents collection.
- Collaborative filter is used to complete a user-term association matrix entries that are missing.
- MLlib provides support to different regression analysis, binary classification, and multiclass classification methods.
- In linear regression, multiple procedures have been developed for the purpose of parameter inference and estimation.

This concludes ‘Spark ML Programming.’

The next lesson is ‘Spark GraphX Programming.’

© Copyright 2015, Simplilearn. All rights reserved.

## Conclusion

With this, we come to the end of the lesson 6 “Spark ML Programming” of the Apache Spark and Scala course. The next lesson is Spark GraphX Programming.

## Lesson 7—Spark GraphX Programming



## Objectives



After completing this lesson, you will be able to:



- Explain the key concepts of Spark GraphX programming
- Discuss the limitations of the Graph-Parallel system
- Describe the operations with a graph
- Discuss the Graph system optimizations

## Objectives

After completing this lesson, you will be able to:

- Explain the key concepts of Spark GraphX programming
- Discuss the limitations of the Graph Parallel system
- Describe the operations with a graph, and
- Discuss the Graph system optimizations

## Introduction to Graph-Parallel System

Big graphs exist in various important applications such as:



**Web Graphs**



**User-Item Graphs**

These graphs allow to perform tasks such as targeting advertising, identifying communities, and deciphering the documents meaning. The size and significance of graph data is growing. In its response, various new large-scale distributed graph-parallel frameworks, such as GraphLab, Giraph, and PowerGraph, have been developed.

## Introduction to Graph-Parallel System

Today, big graphs exist in various important applications, be it web, advertising, or social networks. A few of such graphs are represented graphically on the screen.

These graphs allow to perform tasks such as targeting advertising, identifying communities, and deciphering the documents meaning. This is possible by modeling the relations between products, users, and ideas. The size and significance of graph data is growing. In its response, various new large-scale distributed graph-parallel frameworks, such as GraphLab, Giraph, and PowerGraph, have been developed. With each framework, a new programming abstraction is available. These abstractions allow to explain graph algorithms in a compact manner and also, the related runtime engine that can execute these algorithms efficiently on distributed and multicore systems. Additionally, these frameworks abstract away the issues of the large-scale distributed system design. Therefore, they are capable of simplifying the design, application, and implementation of the new sophisticated graph algorithms to large-scale real-world graph problems.

## Limitations of Graph-Parallel System

The limitations of the Graph-Parallel system are listed below:

- Each framework presents a little different graph computation, which is custom-made for a specific graph applications and algorithms family or the original domain.
- All these frameworks depend on different runtime.
- These frameworks cannot resolve the data ETL issues and issues related to the process of deciphering and applying the computation results.

## Limitations of Graph-Parallel System

Before we move further, you should know the limitations of the Graph-Parallel system.

One of them is that although the current frameworks have various common properties, each of them presents a little different graph computation. These computations are custom-made for a specific graph applications and algorithms family or the original domain.

Additionally, all these frameworks depend on different runtime. Therefore, it is tricky to create these abstractions.

While these frameworks are capable of resolving the graph computation issues, they cannot resolve the data ETL issues. They also cannot address the issues related to the process of deciphering and applying the computation results. The new frameworks however have built-in support available for interactive graph computation.

## Introduction to GraphX

Graph is a graph computation system running in the framework of the data-parallel system. It:

- Extends the RDD abstraction and hence introduces a new feature called Resilient Distributed Graph (RDG)
- Simplifies the graph ETL and analysis process substantially by providing new operations for viewing, filtering, and transforming graphs
- Combines the benefits of graph-parallel and data-parallel systems
- Distributes graphs efficiently as tabular data structures by leveraging new ideas in their representations
- Uses in-memory computation and fault-tolerance
- Simplifies the graph construction and transformation process

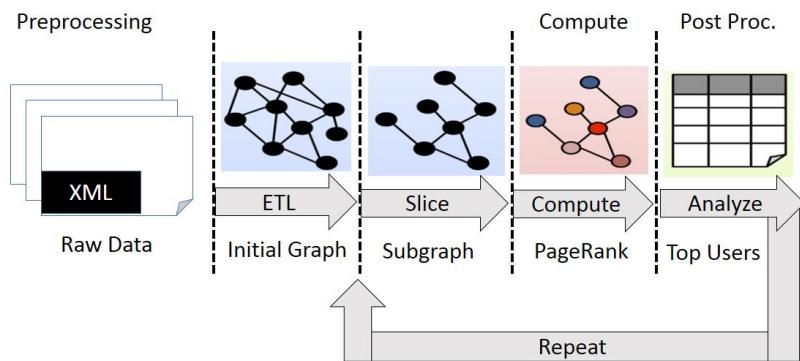
## Introduction to GraphX

Let's now talk about GraphX, which is a graph computation system running in the framework of the data-parallel system. It extends the RDD abstraction and hence introduces a new features called Resilient Distributed Graph or RDG. In a graph, RDG relates records with vertices and edges and produces an expressive computational primitives' collection. In addition, it simplifies the graph ETL and analysis process substantially by providing new operations for viewing, filtering, and transforming graphs.

GraphX combines the benefits of graph-parallel and data-parallel systems as it efficiently expresses graph computation within the framework of the data-parallel system. In addition, GraphX distributes graphs efficiently as tabular data structures by leveraging new ideas in their representations. In a similar way, GraphX uses in-memory computation and fault-tolerance by leveraging the improvements of the data flow systems. GraphX also simplifies the graph construction and transformation process by providing powerful new operations. With the use of these primitives, it is possible to implement the abstractions of PowerGraph and Pregel in a few lines. It is also possible to load, transform, and compute interactively on massive graphs.

## Introduction to GraphX (contd.)

The image below shows how GraphX works:



© Copyright 2015, Simplilearn. All rights reserved.

6

### Introduction to GraphX (contd.)

The image on the screen shows how GraphX works.

## Importing GraphX

To start working with GraphX, you first need to import it and Spark into your project.

**Example:**

```
import org.apache.spark._  
import org.apache.spark.graphx._  
import org.apache.spark.rdd.RDD
```

## Importing GraphX

To start working with GraphX, you first need to import it and Spark into your project. The code to do this is given on the screen.

## The Property Graph

The property graph is a directed multigraph that has properties related to every vertex and edge. In this:

- Every vertex is identified by a unique 64-bit long identifier, known as VertexID, and every edge has an individual source and destination vertex identifier.
- Parameterization happens over the edge or ED and vertex or VD types.
- GraphX reduces the memory footprint by optimizing the presentation of edge and vertex types when they exist as plain old data types and by saving them in specialized arrays.



### Example:

```
class Graph[VD, ED] {
    val vertices: VertexRDD[VD]
    val edges: EdgeRDD[ED]}
```

© Copyright 2015, Simplilearn. All rights reserved.

8

## The Property Graph

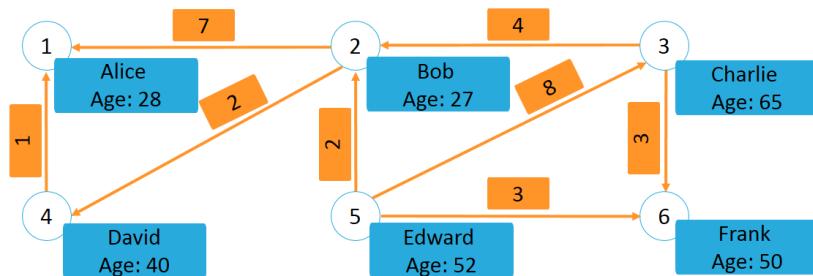
The property graph is defined as a directed multigraph that has properties related to every vertex and edge. Here, a directed graph is defined as a graph that has potentially various parallel edges that share the same source and destination vertexes. Every vertex is identified by a unique 64-bit long identifier, known as VertexID. In a similar manner, every edge has an individual source and destination vertex identifier. The properties of these graphs are saved as Scala or Java objects along with their every vertex and edge.

These graphs are parameterized over the edge or ED and vertex or VD types. Here, the types are the types of objects that are related with every edge and vertex. GraphX reduces the memory footprint by optimizing the presentation of edge and vertex types when they exist as plain old data types and by saving them in specialized arrays.

The code given on the screen also shows the same. Here, this class extends and is an optimized version of RDD[(VertexID, VD)]; however, this class is an optimized version of RDD[Edge[ED]]. Both VertexRDD[VD] and EdgeRDD[ED] leverage internal optimizations and offer additional functionality that is built around graph computation.

## The Property Graph (contd.)

An example of the property graph is shown below:



© Copyright 2015, Simplilearn. All rights reserved.

9

## The Property Graph

An example of the property graph is displayed on the screen.

## Features of the Property Graph

A few features of the property graph are listed below:

Fault-Tolerant, Distributed, Immutable	To perform any changes to the structure/values of the graph, produce a new graph. There are considerable parts of the original graph. These parts are reused in the new graph.
Vertex-Partitioning Heuristics	Use these heuristics to partition the graph across the workers. Similar to RDDs, every graph partition can be created again on a separate machine in case a failure happens.
Similar to a Typed Collections RDDs	It encodes each vertex and edge properties. As a result, it includes members for accessing the graph vertices and edges.

## Features of the Property Graph

A few more features of the property graph are also listed on the screen.

Similar to RDDs, the property graph is also fault-tolerant, distributed, and immutable. If you need to perform any changes to the structure or values of the graph, you would need to produce a new graph with the required changes. Note that there are considerable parts of the original graph, which include structure, indices, and attributes, which remain unaffected. These parts are reused in the new graph, which reduces this inherently functional data-structure cost.

You can use various vertex-partitioning heuristics to partition the graph across the workers. Similar to RDDs, every graph partition can be created again on a separate machine in case a failure happens.

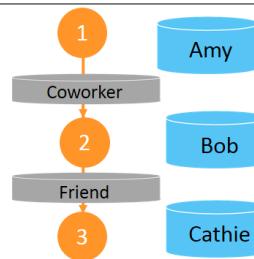
From the logical standpoint, the property graph is similar to a typed collections RDDs pair that encodes each vertex and edge properties. As a result, it includes members for accessing the graph vertices and edges.

## Creating a Graph

The code to create a simple graph of co-worker is shown below:

**Example:**

```
type VertexId = Long
val vertices:RDD[(VertexId,String)] = sc.parallelize(List((1L,"Alice"),(2L,"Bob"),(3L,"Charlie")))
class Edge[ED]( val srclId:VertexId, val dstId: VertexId, val attr: ED)
val edges: RDD[Edge[String]] = sc.parallelize(List(Edge(1L,2L,"coworker"),Edge(2L,3L,"friend")))
val graph = Graph(vertices,edges)
```



© Copyright 2015, Simplilearn. All rights reserved.

11

**Creating a Graph**

Now, let us understand how to create a graph. The code to create a simple graph of co-worker is given on the screen. A graphical representation of this graph is also given on the screen.

## Demo—Create a Graph Using GraphX



This demo will show the steps to create a graph using the GraphX library of Spark.

## Demo—Create a Graph Using Graph X

In this demo, you will learn how to create a graph using the GraphX library of Spark.

We need to import the given packages into spark-shell to perform a graph analysis in spark.

At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge.

The property graph is a directed multigraph with user defined objects attached to each vertex and edge. A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertices. The ability to support parallel edges simplifies modeling scenarios where there can be multiple relationships (for example, co-worker and friend) between the same vertices. Each vertex is keyed by a unique 64-bit long identifier (VertexID). GraphX does not impose any ordering constraints on the vertex identifiers.

Here, we are constructing a graph from a collection of RDDs: by defining them as an array of nodes. Each node has a vertex ID and set of properties name and role.

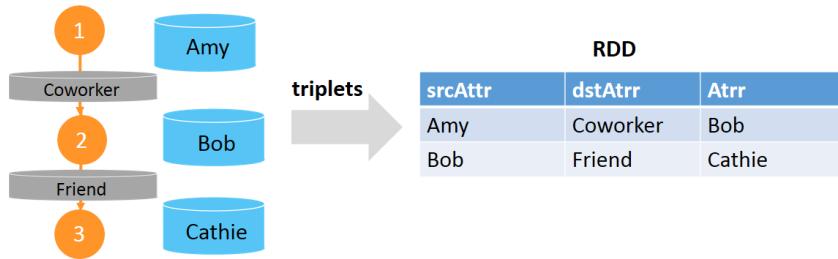
Similarly, we create an RDD for edges. Here each edge establishes a relationship between nodes.

We need define a default user in case there are relationships with missing users. Here this user is David who has not yet been assigned to a project.

We can build an initial graph by passing vertices, edges and default user.

## Triplet View

GraphX also includes a triplet view, which combines the properties of the vertices and edges logically that produce the given class. This class contains the EdgeTriplet class instances. The EdgeTriplet class adds the given members containing the source and destination properties respectively and hence extends the Edge class.



## Triplet View

Apart from the property graph's vertex and edge views, GraphX also includes a triplet view. This view combines the properties of the vertices and edges logically that produce the given class. This class contains the EdgeTriplet class instances.

The EdgeTriplet class adds the given members containing the source and destination properties respectively and hence extends the Edge class.

This view is also shown graphically on the screen.

## Graph Operators

Property graphs also provide various basic operators, which input user-defined functions and result into new graphs that have properties and structures transformed. The core operators with optimized implementations are defined in a graph and the convenient operators expressed as core operators compositions are defined in GraphOps.

**Example:**

```
val graph: Graph[(String, String), String]
// Use the implicit GraphOps.inDegrees operator
val inDegrees: VertexRDD[Int] = graph.inDegrees
```



The reason why core graph operators are differentiated from GraphOps is to be able for supporting various future graph representations.

## Graph Operators

Similar to RDDs, property graphs also provide various basic operators. These operators input user-defined functions and result into new graphs that have properties and structures transformed. The core operators with optimized implementations are defined in a graph. On the other hand, the convenient operators expressed as core operators compositions are defined in GraphOps. However, the GraphOps operators are available as Graph members automatically because of Scala implicits. To understand this, consider the given code example that can compute the in-degree of every vertex that is defined in GraphOps.

The reason why core graph operators are differentiated from GraphOps is to be able for supporting various future graph representations.

## List of Operators

**Example:**

```
class Graph[VD, ED] {
    // Information about the Graph
    val numEdges: Long
    val numVertices: Long
    val inDegrees: VertexRDD[Int]
    val outDegrees: VertexRDD[Int]
    val degrees: VertexRDD[Int]
    // Views of the graph as collections
    val vertices: VertexRDD[VD]
    val edges: EdgeRDD[ED]
    val triplets: RDD[EdgeTriplet[VD, ED]]
    // Change the partitioning heuristic
    def partitionBy(partitionStrategy: PartitionStrategy): Graph[VD, ED]
    // Transform vertex and edge attributes
    def mapVertices[VD2](map: (VertexID, VD) => VD2): Graph[VD2, ED]
    def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]
    def mapEdges[ED2](map: (PartitionID, Iterator[Edge[ED]]) => Iterator[ED2]): Graph[VD, ED2]
    def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
```

**List of Operators**

The code shown on the screen shows a functionality summary of the operators defined in Graph and GraphOps. For simplicity, these are presented as graph members. You should note that a few function signatures have been simplified and a few more advanced functionalities have been removed. Therefore, you should refer to the API docs to determine the official list of operations.

## List of Operators (contd.)


**Example:**

```
// Modify the graph structure
def reverse: Graph[VD, ED]
def subgraph(epred: EdgeTriplet[VD,ED] => Boolean = (x => true), vpred: (VertexID, VD) => Boolean = ((v, d) => true)) : Graph[VD, ED]
def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
// Join RDDs with the graph
def joinVertices[U](table: RDD[(VertexID, U)])(mapFunc: (VertexID, VD, U) => VD): Graph[VD, ED]
def outerJoinVertices[U, VD2](other: RDD[(VertexID, U)])
  (mapFunc: (VertexID, VD, Option[U]) => VD2) : Graph[VD2, ED]
// Aggregate information about adjacent triplets
def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[(VertexID, VD)]]
def mapReduceTriplets[A: ClassTag]( mapFunc: EdgeTriplet[VD, ED] => Iterator[(VertexID, A)], reduceFunc: (A, A) => A) : VertexRDD[A]
// Iterative graph-parallel computation
def pregel[A](initialMsg: A, maxIterations: Int, activeDirection: EdgeDirection)(
  vprog: (VertexID, VD, A) => VD, sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexID,A)], mergeMsg: (A, A) => A) : Graph[VD, ED]
// Basic graph algorithms
def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
def connectedComponents(): Graph[VertexID, ED]
def triangleCount(): Graph[Int, ED]
def stronglyConnectedComponents(numIter: Int): Graph[VertexID, ED]
```

© Copyright 2015, Simplilearn. All rights reserved.

16

## List of Operators (contd.)

The further code is displayed.

## Property Operators

The property graph also contains property operators.

**Example:**

```
class Graph[VD, ED] {  
    def mapVertices[VD2](map: (VertexId, VD) => VD2): Graph[VD2, ED]  
    def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]  
    def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]  
}
```

## Property Operators

Similar to the map operator of RDDs, the property graph also contains property operators. The code to define and use them is displayed on the screen. These operators are generally used for initializing the graph for a specific project or computation.

## Structural Operators

At present, GraphX provides support to just commonly-used structural operators; however, more are expected to be added in the future. The supported ones include reverse operators and subgraph operators.

**Example:**

```
class Graph[VD, ED] {
    def reverse: Graph[VD, ED]
    def subgraph(epred: EdgeTriplet[VD,ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
    def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
    def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
}
```

## Structural Operators

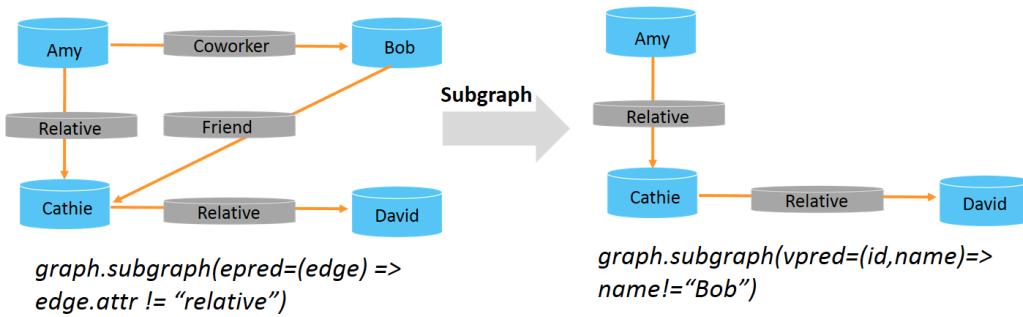
At present, GraphX provides support to just commonly used structural operators; however, more are expected to be added in the future. The supported ones include reverse operators and subgraph operators. The use of these operators is explained through the given code.

The reverse operators reverse all the edge directions and return new graphs. For instance, they can be used in case of computing the inverse PageRank. These operators do not change the properties of vertices and edges and the edges number. Therefore, they can be used without data duplication or movement efficiently.

On the other hand, the subgraph operators input the predicates of vertices and edges and return graphs that contain only the vertices satisfying the vertex predicate and edges satisfying the edge predicate. They also connect the vertices satisfying the vertex predicate. These operators are usable for restricting the graph to the suitable vertices and edges by eliminating the broken links.

## Subgraphs

Let's learn more about subgraphs.



## Subgraphs

Let's learn more about subgraphs. In the first image shown on the screen, this operator is being used to return the graph that contains only those vertices where the relation type is not “relative”.

However, in the second image, it is being used to return the graph that contains only those vertices who value is Bob.

## Join Operators

Sometimes, it is required to join data originating from RDDs or external collections that have graphs. In such cases, join operators are useful. The supported ones include `joinVertices` operator and `outerJoinVertices` operators.

**Example:**

```
class Graph[VD, ED] {
    def joinVertices[U](table: RDD[(VertexId, U)])(map: (VertexId, VD, U) => VD) : Graph[VD, ED]
    def outerJoinVertices[U, VD2](table: RDD[(VertexId, U)])(map: (VertexId, VD, Option[U]) =>
    VD2)
        : Graph[VD2, ED]
}
```

## Join Operators

Sometimes, it is required to join data originating from RDDs or external collections that have graphs. For instance, in cases when you need to pull the vertex properties from one graph to the other, you might require extra properties. In such cases, join operators are useful. The supported ones include `joinVertices` operator and `outerJoinVertices` operators. The use of these operators is explained through the given code.

The `joinVertices` operator is capable of joining the vertices with an RDD. It then returns a graph having its vertex properties received by the application of the user-defined map function to the joined vertices result. For the vertices with matching value in the RDD, the original value is retained.

On the other hand, the `outerJoinVertices` operator is more general and operates similar to `joinVertices`. The only difference is that the user-defined map function is applied to all vertices. It can alter the type of vertex property. The map function takes an Option type, as all vertices may not have a matching value in the RDD being inputted.

## Demo—Perform Graph Operations Using GraphX

This demo will show the steps to perform graph operations using GraphX.

## Demo—Perform Graph Operations Using GraphX

In this demo, you will learn how to perform a graph analysis using the GraphX library of Spark.

Let's filter out those vertices from the earlier created graph to count the number of vertices having property "MR Developer". We can use the filter method of Vertex RDD for this.

You can see the number of such vertices is just one.

Lets count those edges whose source ID is greater than the distinction ID. We can use the filter method of edge RDD for this.

In addition to the vertex and edge views of the property graph, GraphX also exposes a triplet view. The triplet view logically joins the vertex and edge properties yielding an RDD[EdgeTriplet[VD, ED]] containing instances of the EdgeTriplet class.

Use the triplets view to create an RDD of facts to find out all the relationship in the graph.

Let's print all these relationships by typing the command shown on the screen.

## Demo—Perform Subgraph Operations

This demo will show the steps to perform the subgraph operations using GraphX.

## Demo—Perform Subgraph Operations

In this demo, you will learn how to perform a graph analysis using the GraphX library of Spark.

The subgraph operator takes vertex and edge predicates and returns the graph containing only the vertices that satisfy the vertex predicate and edges that satisfy the edge predicate and connect vertices that satisfy the vertex predicate. The subgraph operator can be used in a number of situations to restrict the graph to the vertices and edges of interest or eliminate broken links. For example, in the following code we will remove broken links.

We can use subgraph to get all the vertices and its relationships by checking the vertex predicate properties not equal to “Work not yet assigned”.

To print all these vertices, we can use the command as shown on the screen. Here we are using collect and foreach methods to println all the vertices.

## Neighborhood Aggregation

An important step in various graph analytics tasks is to aggregate the neighborhood information of every vertex. Various iterative graph algorithms such as Shortest Path and PageRank perform this operation. The primary aggregation operator, `mapReduceTriplets`, inputs a user-defined map function applied to every triplet and then provides messages that are destined to none, both, or either vertices in the triplet.

**Example:**

```
class Graph[VD, ED] {
  def mapReduceTriplets[A]{
    map: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)], reduce: (A, A) => A : VertexRDD[A] }
```



For improving performance, this primary operator has been changed to the new `graph.AggregateMessages` operator.

## Neighborhood Aggregation

An important step in various graph analytics tasks is to aggregate the neighborhood information of every vertex. For instance, you might require to identify the number of every user's followers. Various iterative graph algorithms such as Shortest Path and PageRank perform this operation.

The primary aggregation operator, `mapReduceTriplets`, inputs a user-defined map function applied to every triplet and then provides messages that are destined to none, both, or either vertices in the triplet. Its use is as depicted in the given code.

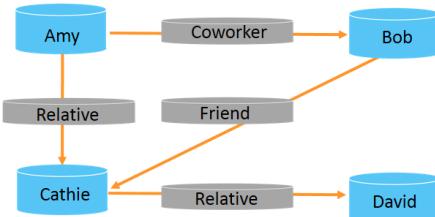
For improving performance, this primary operator has been changed to the new `graph.AggregateMessages` operator.

## mapReduceTriplets

With mapReduceTriplets, the map function is applied to every edge graph triplet. The messages thus yielded are destined to the vertices that are adjacent. With the reduce function, messages that are destined to the same vertex are aggregated. As a result, a VertexRDD is obtained that contains aggregate messages for every vertex.

**Example:**

```
def mapReduceTriplets[A: ClassTag]( mapFunc: EdgeTriplet[VD, ED] => Iterator[(VertexID, A)],  
reduceFunc: (A, A) => A ) : VertexRDD[A]  
graph.mapReduceTriplets(edge=>Iterator((edge.srcId,1),(edge.dstId,1)),_+_)
```



**mapReduce  
Triplets**

Vertex ID	Degree
Amy	2
Bob	2
Cathie	3
David	1

© Copyright 2015, Simplilearn. All rights reserved.

24

## mapReduceTriplets

Let's discuss more about the primary aggregation operator, mapReduceTriplets.

As discussed, with this operator, the map function is applied to every edge graph triplet. The messages thus yielded are destined to the vertices that are adjacent. With the reduce function, messages that are destined to the same vertex are aggregated. As a result, a VertexRDD is obtained that contains aggregate messages for every vertex.

For instance, consider the given code in which mapReduceTriplets is being used for counting number of degree for each vertex.

The image on the screen also shows the application of this operator.

## Demo—Perform MapReduce Operations

This demo will show the steps to perform MapReduce operations in Graph using GraphX.

## Demo—Perform MapReduce Operations

In this demo, you will learn how to perform the map reduce graph analysis using the GraphX library of Spark.

A key step in many graph analytics tasks is aggregating the information about the neighborhood of each vertex. For example, we might want to know the number of followers each user has or the average age of the followers of each user. Many iterative graph algorithms (for example, PageRank, Shortest Path, and connected components) repeatedly aggregate the properties of neighboring vertices (for example, current PageRank Value, shortest path to the source, and smallest reachable vertex ID).

For this example, we need to import the GraphGenerators class apart from the graphX classes. See the import statement shown on the screen.

Create a graph with "age" as the vertex property. Here we use a random graph for simplicity.

The core aggregation operation in GraphX is aggregateMessages. This operator applies a user defined sendMsg function to each edge triplet in the graph and then uses the mergeMsg function to aggregate those messages at their destination vertex.

The user defined sendMsg function takes an EdgeContext, which exposes the source and destination attributes along with the edge attribute and functions (sendToSrc, and sendToDst) to send messages to the source and destination attributes. Think of sendMsg as the map function in map-reduce. The user defined mergeMsg function takes two messages destined to the same vertex and yields a single

message. Think of mergeMsg as the reduce function in map-reduce. The aggregateMessages operator returns a VertexRDD[Msg] containing the aggregate message (of type Msg) destined to each vertex. Vertices that did not receive a message are not included in the returned VertexRDD.

Here, we created a graph with "age" as the vertex property. We used a random graph for simplicity. Now, we are going to compute the number of older followers and their total age.

Here , as the map method we are Sending message to the destination vertex containing counter and age. In the reduce method we are going to add counter and age

Using the mapValue method we are going to divide the total age by the number of older followers to get the average age of older followers

To Display the results, we can use the command as shown on the screen . Here we are using the collect and foreach methods to print the results.

## Counting Degree of Vertex

One of the common aggregation tasks is to compute the degree of every vertex, which is defined as the number of edges that are adjacent to every vertex. Directed graphs are generally required to identify the out-degree, in-degree, and the total degree of every vertex. The operators to compute these degrees of every vertex are included in the GraphOps class.

**Example:**

```
// Compute the max degrees
val maxInDegree: (VertexId, Int) = graph.inDegrees.reduce(max)
val maxOutDegree: (VertexId, Int) = graph.outDegrees.reduce(max)
val maxDegrees: (VertexId, Int) = graph.degrees.reduce(max)
```

## Counting Degree of Vertex

One of the common aggregation tasks is to compute the degree of every vertex, which is defined as the number of edges that are adjacent to every vertex. When it comes to directed graphs, it is generally required to identify the out-degree, in-degree, and the total degree of every vertex. The operators to compute these degrees of every vertex are included in the GraphOps class. For instance, consider the given code that is computing the maximum in, out, and total degrees.

## Collecting Neighbors

Sometimes, it is easy to express computation by performing a collection of neighboring vertices and the related attribute at every vertex. To do so, you can use the `collectNeighbors` and `collectNeighbors` operators.

**Example:**

```
class GraphOps[VD, ED] {  
    def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[VertexId]]  
    def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[ Array[(VertexId, VD)] ]  
}
```



If possible, try to express the same computation by the use of the `aggregateMessages` operator.

© Copyright 2015, Simplilearn. All rights reserved.

27

## Collecting Neighbors

Sometimes, it is easy to express computation by performing a collection of neighboring vertices and the related attribute at every vertex. To do so, you can use the given operators. The code to use them is given on the screen.

These operators can prove to be very costly because they need substantial communication and duplicate information. If possible, try to express the same computation by the use of the `aggregateMessages` operator.

## Caching and Uncaching

In GraphX, caching and uncaching is explained as below:

- GraphX must be cached explicitly when using multiple times using the Graph.cache() method first.
- In case of iterative computations, you may also need to uncache to obtain best performance.
- Cached graphs and RDDs, by default, exist in memory until a pressure evicts in an LRU order. Therefore, it is more efficient if you uncache these intermediate results as soon as they are not required.
- Graphs include various RDDs and therefore, it is tricky to correctly unpersist them.
- In case of iterative computations, you should use the Pregel API that unpersists intermediate results correctly.

## Caching and Uncaching

Similar to RDDs, GraphX must be cached explicitly when using multiple times, as they are not persisted in memory by default. Therefore, you should always call the Graph.cache() method first.

In case of iterative computations, you may also need to uncache to obtain best performance. Cached graphs and RDDs, by default, exist in memory until a pressure evicts in an LRU order. In such computations, intermediate results originating from previous computations fill the cache. However, they get evicted eventually, the data that is unnecessarily stored in memory slows down garbage collection. Therefore, it is more efficient if you uncache these intermediate results as soon as they are not required. This includes uncaching all other datasets, materializing graphs or RDDs, and using only the materialized datasets for further iterations. Graphs include various RDDs and therefore, it is tricky to correctly unpersist them. In case of iterative computations, you should use the Pregel API that unpersists intermediate results correctly.

## Graph Builders

To build a graph from a vertices and edges collection existing on a disk or in an RDD, GraphX provides various ways. By default, none of these graph builders repartitions the edges of a graph. Instead, those are left in their as is default partitions. These graph builders are listed below:

Graph.groupEdges	Needs that the graph should be repartitioned; therefore, before calling this, you must call Graph.partitionBy.
Graph.apply	Lets you create a graph from RDDs containing vertices and edges, picks duplicate vertices arbitrarily and the vertices found in the edge RDD, but does not pick the vertex RDD assigned the default attribute.
Graph.fromEdges	Lets you create a graph only from an RDD of edges and creates any vertices mentioned by edges automatically and assigns them the default value
Graph.fromEdgeTuples graph	Lets you create a graph only from an RDD of edge tuples and provides support to deduplicate the edges

## Graph Builders

To build a graph from a vertices and edges collection existing on a disk or in an RDD, GraphX provides various ways. By default, none of these graph builders repartitions the edges of a graph. Instead, those are left in their as is default partitions. These graph builders are listed on the screen.

Graph.groupEdges needs that the graph should be repartitioned. This is because of its assumption that identical edges are collocated on the same partition. Therefore, before calling this, you must call Graph.partitionBy.

The next graph builder, Graph.apply, lets you create a graph from RDDs containing vertices and edges. It picks duplicate vertices arbitrarily. It also picks the vertices that are found in the edge RDD, but does not pick the vertex RDD that is assigned the default attribute.

The Graph.fromEdges builder lets you create a graph only from an RDD of edges. It creates any vertices mentioned by edges automatically and assigns them the default value.

With the Graph.fromEdgeTuples graph builder, you can create a graph only from an RDD of edge tuples. This assigns the value 1 to the edges and then creates any vertices mentioned by edges automatically while assigning them the default value. This graph builder also provides support to deduplicate the edges. For this, you would need to pass some of a PartitionStrategy as the uniqueEdges parameter. It also requires a partition strategy to collocate similar edges on the same partition in order to deduplicate them.

## Vertex and Edge RDDs

Vertex and edge RDDs are explained as below:

**Vertex RDDs**

- The VertexRDD[A] is an extension of the RDD[(VertexID, A)] class.
- It adds additional constraints that every VertexID appears just once and represents a vertices set.
- This is accomplished by saving the attributes of vertices in a hash-map and reusable data structure.

**Edge RDDs**

- The EdgeRDD[ED] is an extension of the RDD[Edge[ED]] class.
- It organizes the edges into blocks partitioned by using one of the partitioning strategies defined in PartitionStrategy.
- The operations on the Edge RDDs are achieved by using graph operators, or they depend upon the operations that are defined in the base RDD class.

## Vertex and Edge RDDs

Another concept related to GraphX is vertex RDDs. The VertexRDD[A] is an extension of the given class. It adds additional constraints that every VertexID appears just once. In addition, it represents a vertices set, where each vertex has an attribute of type A. This is accomplished by saving the attributes of vertices in a hash-map and reusable data structure. As a result, two VertexRDDs can be combined in constant time with no hash evaluations if they are derived from the same base.

Similarly, the EdgeRDD[ED] is an extension of the given class. It organizes the edges into blocks that are partitioned by the use of one of the partitioning strategies that are defined in PartitionStrategy. The attributes of edges and the adjacency structure are saved differently that enables the maximum reuse when it comes to the changing attribute values. The use of three additional functions exposed by it is explained through the given code.

Generally, the operations on the Edge RDDs are achieved by the use of graph operators, or they depend upon the operations that are defined in the base RDD class.

## Graph System Optimizations

Graph system optimization is explained as below:

- GraphX uses the vertex-cut approach in case of distributed graph partitioning.
- Logically, it corresponds to the assignment of edges to machines and letting the vertices to span across various machines. You can choose any strategy by the use of the `Graph.partitionBy` operator that repartitions the graph and switch to 2D-partitioning or other heuristics easily too.
- The key challenge to the effective graph-parallel computation after the edges have been partitioned is to join the vertex attributes with the edges efficiently.

## Graph System Optimizations

GraphX uses the vertex-cut approach in case of distributed graph partitioning. Instead of splitting the graphs along edges, it partitions them along vertices. Doing so helps in the reduction of storage overhead and communication.

From the logical standpoint, it corresponds to the assignment of edges to machines and letting the vertices to span across various machines. The correct and exact method to assign edges is dependent upon the `PartitionStrategy`. You can choose any strategy by the use of the `Graph.partitionBy` operator that repartitions the graph. By default, the initial partitioning of the edges is used as the partitioning strategy that is provided on graph construction. However, you can switch to 2D-partitioning and other heuristics easily too.

The key challenge to the effective graph-parallel computation after the edges have been partitioned is to join the vertex attributes with the edges efficiently. You move vertex attributes to edges because real-world graphs include more edges as compared to vertices. In addition, you maintain a routing table internally that explains where to broadcast vertices when it comes to implement the join needed for `aggregateMessages` and triplets like operations. This is because all partitions do not include edges that are adjacent to all vertices.

## Built-In Algorithms

For simplifying analytics tasks, GraphX also contains a few graph algorithms. These are included in the org.apache.spark.graphx.lib package and are accessible through GraphOps as directed methods on graphs. These algorithms are:

PageRank	Assumes that each edge from a to b represents an endorsement of b's importance by a; <b>Example:</b> <code>def pageRank(tol:Double): Graph[Double, Double]</code>
Connected Components	Works by labeling every connected graph component with ID of its lowest-numbered vertex; <b>Example:</b> <code>def triangleCount(): Graph[Int, ED] def connectedComponents()</code>
Triangle Counting	Assumes a vertex as a part of a triangle, which has two adjacent vertices and an edge between them; <b>Example:</b> <code>def connectedComponents(): Graph[VertexId, ED]</code>

## Built-In Algorithms

For simplifying analytics tasks, GraphX also contains a few graph algorithms. These are included in the org.apache.spark.graphx.lib package and are accessible through GraphOps as directed methods on graphs. These algorithms are listed as page rank, connected components, and triangle counting.

PageRank assumes that each edge from a to b represents an endorsement of b's importance by a. It thus measures the importance in a graph. For instance, in Twitter, if a person is followed by various people, he or she will be ranked highly.

On the PageRank object, GraphX is available with various static and dynamic PageRank implementations as methods. While dynamic ones run till the ranks coverage, static ones run for a fixed iterations number. It can be directly called as methods on a graph. The code to use it is given on the screen.

The next algorithm, the connected components algorithm works by labeling every connected graph component with ID of its lowest-numbered vertex. For instance, in case of social networks, these components can approximate clusters. It is called by one of its implementation, theConnectedComponents object. An example code to use it is given on the screen.

The Triangle Counting algorithm assumes a vertex as part of a triangle, which has two adjacent vertices and an edge between them. It is implemented in the TriangleCount object, which computes the triangle number passing through every vertex and provides them a clustering measure.

**QUIZ  
1**

Which of the following statement is true about the property graph ?

- a. The property graph is a directed multigraph with user defined objects attached to each vertex and edge.
- b. There can be only one to one relationship between each vertex.
- c. GraphX imposes ordering constraints on the vertex identifiers.
- d. It includes various views.

**QUIZ  
2**

Which of following graph algorithms are available as direct methods in GraphOps? *Select all that apply.*

- a. PageRank
- b. VertexRDD
- c. Connected components
- d. Triangle counting



**QUIZ  
3**

Which of the following is an operator of GraphX?

- a. mapReduce
- b. mapReduceTriplets
- c. mapRelationships
- d. mapTriplets

**QUIZ  
4**

Which of the following is used for building a graph from a collection of vertices and edges in an RDD or on a disk? *Select all that apply.*

- a. apply
- b. fromEdges
- c. fromEdgeTuples graph
- d. fromEdgeTuples



## ANSWERS

S.No.	Question	Answer & Explanation
1	Which of the following statement is true about the property graph?	a. The property graph is a directed multigraph with user-defined objects attached to each vertex and edge.
2	Which of following graph algorithms are available as direct methods in GraphOps?	a., c., and d. PageRank, connected components, and triangle counting algorithms are available as direct methods in GraphOps.
3	Which of the following is an operator of GraphX?	b. mapReduceTriplets is an operator supported in GraphX.
4	Which of the following is used for building a graph from a collection of vertices and edges in an RDD or on a disk?	a., b., c., and d. All these methods are the ways of building a graph from a collection of vertices and edges in an RDD or on a disk.

## Summary



Let us summarize the topics covered in this lesson:



- Graphs allow to perform tasks such as targeting advertising, identifying communities, and deciphering the documents meaning.
- There are several limitations of the Graph-Parallel system such as runtime dependency and data ETL issues.
- GraphX is a graph computation system running in the framework of the data-parallel system.
- To start working with GraphX, you first need to import it and Spark into your project.
- The property graph is defined as a directed multigraph that has properties related to every vertex and edge.
- GraphX also includes a triplet view.
- GraphX operators input user-defined functions and results into new graphs that have properties and structures transformed. These include property operators, structural operators, and join operators.
- An important step in various graph analytics tasks is to aggregate the neighborhood information of every vertex.
- One of the common aggregation tasks is to compute the degree of every vertex.

© Copyright 2015, Simplilearn. All rights reserved.

42

## Summary

Let us summarize the topics covered in this lesson:

- Graphs allow to perform tasks such as targeting advertising, identifying communities, and deciphering the documents meaning.
- There are several limitations of the Graph-Parallel system such as runtime dependency and data ETL issues.
- GraphX is a graph computation system running in the framework of the data-parallel system.
- To start working with GraphX, you first need to import it and Spark into your project.
- The property graph is defined as a directed multigraph that has properties related to every vertex and edge.
- GraphX also includes a triplet view.
- GraphX operators input user-defined functions and result into new graphs that have properties and structures transformed. These include property operators, structural operators, and join operators.
- An important step in various graph analytics tasks is to aggregate the neighborhood information of every vertex.
- One of the common aggregation tasks is to compute the degree of every vertex.

## Summary (contd.)

Let us summarize the topics covered in this lesson:



- Sometimes, it is easy to express computation by performing a collection of neighboring vertices and the related attributes at every vertex.
- GraphX must be cached explicitly when using multiple times.
- To build a graph from a vertices and edges collection existing on a disk or in an RDD, GraphX provides Graph Builders.
- The VertexRDD[A] adds additional constraints that every VertexID appears just once.
- The EdgeRDD[ED] organizes the edges into blocks that are partitioned by the use of one of the partitioning strategies.
- GraphX uses the vertex-cut approach in case of distributed graph partitioning.
- For simplifying analytics tasks, GraphX also contains a few graph algorithms.

## Summary (contd.)

- Sometimes, it is easy to express computation by performing a collection of neighboring vertices and the related attribute at every vertex.
- GraphX must be cached explicitly when using multiple times.
- To build a graph from a vertices and edges collection existing on a disk or in an RDD, GraphX provides Graph Builders.
- The VertexRDD[A] adds additional constraints that every VertexID appears just once.
- The EdgeRDD[ED] organizes the edges into blocks that are partitioned by the use of one of the partitioning strategies.
- GraphX uses the vertex-cut approach in case of distributed graph partitioning.
- For simplifying analytics tasks, GraphX also contains a few graph algorithms.

simplilearn

This concludes ‘Spark GraphX Programming.’

This was the last lesson of the course.

© Copyright 2015, Simplilearn. All rights reserved.

## Thank You

With this, we come to the end of the lesson 7 “Spark GraphX Programming” of the Apache Spark and Scala course. This was the last lesson of the course.



Simplilearn offers flexible modes of learning to suit the learning needs of working professionals like you.

### Self-Learning



### Instructor-led Training



Classroom



Online

### Courses we offer:

- ▶ PMP® Training
- ▶ PRINCE2® Training
- ▶ PMI-ACP® Training
- ▶ CSM Training
- ▶ CBAP® Training
- ▶ Agile and Scrum Training
- ▶ RMP® Training

- ▶ Big Data & Hadoop Developer Training
- ▶ Cloud Computing Training
- ▶ ITIL® Training
- ▶ TOGAF Training
- ▶ Six Sigma Training
- ▶ CFA Training
- ▶ Financial Modeling Training

- ▶ FRM Training
- ▶ CEH Training
- ▶ CISSP® Training
- ▶ CCNA® Training
- ▶ SAP Training
- ▶ Microsoft Training
- ▶ Salesforce Training

+200 more courses

[www.simplilearn.com](http://www.simplilearn.com)

The **Rewards of Friendship** are even greater now!

## Refer & Earn

Receive Amazon Voucher worth  
up to \$100 when you refer your friends

How it works



### Refer Your Friends

Refer your friends by entering their name and email address  
on the website



### Friends Enroll for a Course

Your friends enroll for any of our courses available on our website



### Win Rewards

- For each referral for a course >\$500, win Amazon Voucher worth \$100
- For each referral for a course <\$500, win Amazon Voucher worth \$20

#### For India:

- For each referral for a course >INR 20,000, win Amazon Voucher worth INR 5000
- For each referral for a course <INR 20,000, win Amazon Voucher worth INR 1000

To know more, visit  
[www.simplilearn.com/refer-and-earn](http://www.simplilearn.com/refer-and-earn)

Stay Connected with us at



Simplilearn is a leading provider of a suite of professional certification courses  
that address unique learning needs of working professionals

## ACCREDITATIONS



Global Registered  
Education Provider (REP)  
for PMI, USA.  
Our REP ID is 3147



Accredited Training  
Organization (ATO) by  
APMG International, UK  
for PRINCE2® Foundation  
and Practitioner



Accredited Training  
Organization (ATO) by  
APMG International, UK  
for ITIL® 2011  
Foundation and ITIL® 2011  
Intermediate



Listed on CFA Institute  
Prep Providers for CFA  
Level I. Follows CFA  
Institute Prep Provider  
Guidelines



Listed on GARP Exam  
Preparation Providers for  
FRM Part 1



Accredited Training  
Provider (ATP) and  
Authorized Examination  
Center (AEC) by ISTQB



IASSC Accredited  
Examination Center (AEC)  
by PEOPLECERT



Accredited Training  
Provider (ATP) and  
Accredited Examination  
Center (AEC) by EXIN



Accredited Training  
Provider (ATP) by ASTQB



Accredited Training  
Organization (ATO) by  
TUV-SUD for ITIL® 2011  
Foundation and ITIL® 2011  
Intermediate



Simplilearn is a Registered  
Education Provider (REP)  
of Scrum Alliance, Inc.  
Scrum Alliance is the  
governing body for CSM  
and CSPO certifications



Simplilearn is an  
Authorized Training  
Partner (ATP) and  
Accredited Training  
Center (ATC) for  
EC-Council

## Disclaimer

"PMI®", "PMBOK®", "PMP®" and "PMI-ACP®" are registered marks of the Project Management Institute, Inc.

The Swirl logoTM is a trade mark of AXELOS Limited.

ITIL® is a registered trade mark of AXELOS Limited.

PRINCE2® is a Registered Trade Mark of AXELOS Limited.

MSP® is a Registered Trade Mark of AXELOS Limited

"CSM", "CST" are Registered Trade Marks of The Scrum Alliance, USA.

COBIT® is a trademark of ISACA® registered in the United States and other countries.

CISA® is a Registered Trade Mark of the Information Systems Audit and Control Association (ISACA) and the IT Governance Institute

CISSP is a registered mark of the The International Information Systems Security Certification Consortium ((ISC)2)

FRM®, GARPTM and Global Association of Risk ProfessionalsTM, are trademarks owned by the Global Association of Risk Professionals, Inc. Global Association of Risk Professionals, Inc. (GARPTM) does not endorse, promote, review or warrant the accuracy of the products or services offered by Simplilearn for FRM® related information, nor does it endorse any pass rates claimed by the provider. Further, GARP is not responsible for any fees or costs paid by the user to Simplilearn nor is GARP responsible for any fees or costs of any person or entity providing any services to Simplilearn Study Program.

CFA Institute does not endorse, promote, or warrant the accuracy or quality of the products or services offered by Simplilearn. CFA Institute, CFA®, and Chartered Financial Analyst® are trademarks owned by CFA Institute.

SAP® is the trademark or registered trademark of SAP AG. Simplilearn is not affiliated with or endorsed by SAP AG. All training material is proprietary contents of Simplilearn and its Partners. SAP Access is provided by Simplilearn in partnership with Leading C, a licensed SAP provider in accordance with SAP EULA dated March 2004.

The image features a world map centered on the Atlantic Ocean. Overlaid on the map are several yellow callout boxes containing text and icons, representing different training categories and their global reach.

- PROJECT MANAGEMENT:** Includes PMP® Training, PRINCE2® Certification, CSM Training, and Agile & Scrum Certification (PMI-ACP)® Training.
- BIG DATA & CLOUD COMPUTING:** Includes COMPTIA Cloud+ Training and Big Data & Cloud Computing Training.
- FINANCE MANAGEMENT:** Includes Financial Modeling with MS Excel Training and CFA Level 1 Training.
- SOFTWARE TESTING:** Includes Software Testing Training and CTFL® Training.
- CISCO CERTIFICATION:** Includes CCNA® Training and CCNP® Training.
- SAP FUNCTIONAL MODULES CERTIFICATION:** Includes SAP Technical Modules Certification and SAP Functional Modules.
- IT SERVICE MANAGEMENT:** Includes ITIL® Intermediate Certification and ITIL® Expert Training.
- IT SECURITY MANAGEMENT:** Includes CISSP® Training and COBIT® Training.
- QUALITY MANAGEMENT:** Includes Six Sigma Certification and Lean Management Training.
- MICROSOFT® CERTIFICATION:** Includes Microsoft® Excel Training, Microsoft® Project Training, and Microsoft® Project Certification.

At the top of the page, four circular icons represent key statistics:

- 200,000+ PROFESSIONALS TRAINED (Icon: People)
- 150+ COUNTRIES (Icon: World Map)
- 200+ COURSES (Icon: Books)
- ACCREDITED BY 20+ GLOBAL BODIES (Icon: Ribbon)

At the bottom, there are two contact options:

- support@simplilearn.com** (Icon: Envelope)
- Live Chat 24/7 on simplilearn.com** (Icon: Chat bubble)