|  |
| --- |
| **Apache Spark** |
| https://upload.wikimedia.org/wikipedia/commons/e/ea/Spark-logo-192x100px.png |

Apache Spark is an open-source ***cluster-computing framework,* processing engine** built around speed, ease of use, and sophisticated analytics. Spark provides an interface for programming entire clusters with *implicit data parallelism* and *fault-tolerance*.

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

Spark vs. Hadoop

* **Speed**: Engineered from the bottom-up for performance, Spark can be 100x faster than Hadoop for large scale data processing *by exploiting in memory computing* and other optimizations. Spark is also fast *when data is stored on disk*, and currently holds the world record *for large-scale on-disk sorting*.
* **Ease of Use**: Spark has easy-to-use APIs for operating on large datasets. This includes a collection of over *100 operators for transforming data* and familiar *data frame APIs* for manipulating semi-structured data.
* **A Unified Engine**: Spark comes packaged with higher-level libraries, including *support for SQL queries, streaming data, machine learning and graph processing.* These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows.

Data structure: **resilient distributed dataset (RDD**), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.

It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's **RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory**.

The **availability** of RDDs facilitates the implementation of *both iterative algorithms, that visit their dataset multiple times in a loop*, *and interactive/exploratory data analysis, i.e., the repeated database-style querying of data*. The *latency* of such applications (compared to a MapReduce implementation, as was common in Apache Hadoop stacks) may be reduced by several orders of magnitude. Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.

Apache Spark requires a **cluster manager** and a **distributed storage system**. For cluster management, Spark supports standalone (*native Spark cluster*), *Hadoop YARN*, *or Apache Mesos*. For distributed storage, Spark can interface with a wide variety, including *Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu*, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.

Spark Core

**Spark Core** is the foundation of the overall project. It provides *distributed task dispatching, scheduling, and basic I/O functionalities*, exposed through an application programming interface (for Java, Python, Scala, and R) centered on the RDD abstraction (the Java API is available for other JVM languages, but is also usable for some other non-JVM languages, such as Julia, that can connect to the JVM).

This interface mirrors a functional/higher-order model of programming: a "driver" program invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster. These operations, and additional ones such as joins, take RDDs as input and produce new RDDs.

*RDDs are* ***immutable*** *and their operations are* ***lazy****;* ***fault-tolerance*** *is achieved by keeping track of the "****lineage****" of each RDD (the sequence of operations that produced it) so that it can be reconstructed in the case of data loss*. RDDs can contain any type of Python, Java, or Scala objects.

Aside from the RDD-oriented functional style of programming, Spark provides *two restricted forms of shared variables*: **broadcast variables** reference *read-only data* that needs to be *available on all nodes*, while **accumulators** can be used to program reductions in an imperative style.

A typical example of RDD-centric functional programming is the following Scala program that computes the frequencies of all words occurring in a set of text files and prints the most common ones. Each map, flatMap (a variant of map) and reduceByKey takes an anonymous function that performs a simple operation on a single data item (or a pair of items), and applies its argument to transform an RDD into a new RDD.

val conf = new SparkConf().setAppName("wiki\_test") // **create** a spark config object

val sc = new SparkContext(conf) // **Create** a spark context

val **data** = sc.textFile("/path/to/somedir") // **Read** files **from** "somedir" **into** an RDD **of** (filename, content) pairs.

val tokens = **data**.flatMap(\_.split(" ")) // Split **each** file **into** a list **of** tokens (words).

val wordFreq = tokens.map((\_, 1)).reduceByKey(\_ + \_) // **Add** a **count** **of** one **to** **each** token, **then** **sum** the counts per word type.

wordFreq.sortBy(s => -s.\_2).map(x => (x.\_2, x.\_1)).top(10) // **Get** the top 10 words. Swap word **and** **count** **to** sort **by** **count**.

Spark SQL

**Spark SQL** is a component on top of Spark Core that introduced a *data abstraction* called **DataFrames**, which provides support for structured and semi-structured data. Spark SQL provides a **domain-specific language (DSL**) to *manipulate DataFrames* in Scala, Java, or Python. It also provides *SQL language support*, with command-line interfaces and ODBC/JDBC server. Although DataFrames lack the compile-time type-checking afforded by RDDs, as of Spark 2.0, the strongly typed DataSet is fully supported by Spark SQL as well.

Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data. It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning).

**import** org.apache.spark.sql.SQLContext

**val** url = "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword" *// URL for your database server.*

**val** sqlContext = **new** org.apache.spark.sql.SQLContext(sc) *// Create a sql context object*

**val** df = sqlContext

.read

.format("jdbc")

.option("url", url)

.option("dbtable", "people")

.load()

df.printSchema() *// Looks the schema of this DataFrame.*

**val** countsByAge = df.groupBy("age").count() *// Counts people by age*

Spark Streaming

**Spark Streaming** *leverages Spark Core's fast scheduling capability to perform* ***streaming analytics***. It *ingests data in mini-batches and performs RDD transformations* on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture. However, this convenience comes with the penalty of *latency equal to the mini-batch duration*. Other *streaming data engines that process event by event* rather than in mini-batches include *Storm* and the streaming component of Flink. Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.

Many applications need the ability to process and analyze not only batch data, but also streams of new data in real-time. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark’s ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS, Flume, Kafka, and Twitter.

MLlib Machine Learning Library

**Spark MLlib** is a *distributed machine learning framework* on top of Spark Core that, due in large part to the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by Apache Mahout (according to benchmarks done by the MLlib developers against the Alternating Least Squares (ALS) implementations, and before Mahout itself gained a Spark interface), and scales better than Vowpal Wabbit. Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which *simplifies large scale machine learning pipelines*, including:

* summary statistics, correlations, stratified sampling, hypothesis testing, random data generation
* classification and regression: support vector machines, logistic regression, linear regression, decision trees, naive Bayes classification
* collaborative filtering techniques including alternating least squares (ALS)
* cluster analysis methods including k-means, and Latent Dirichlet Allocation (LDA)
* dimensionality reduction techniques such as singular value decomposition (SVD), and principal component analysis(PCA)
* feature extraction and transformation functions
* optimization algorithms such as stochastic gradient descent, limited-memory BFGS (L-BFGS)

Machine learning has quickly emerged as a critical piece in mining Big Data for actionable insights. Built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms (e.g., multiple iterations to increase accuracy) and blazing speed (up to 100x faster than MapReduce).

GraphX

**GraphX** is a *distributed graph processing framework* on top of Apache Spark. Because it is based on RDDs, which are immutable, *graphs are immutable* and thus GraphX is unsuitable for graphs that need to be updated, let alone in a transactional manner like a graph database. GraphX provides two separate APIs for implementation of massively parallel algorithms (such as PageRank): a Pregel abstraction, and a more general MapReduce style API. Unlike its predecessor Bagel, which was formally deprecated in Spark 1.6, GraphX has full support for property graphs (graphs where properties can be attached to edges and vertices).

GraphX can be viewed as being the Spark in-memory version of Apache Giraph, which utilized Hadoop disk-based MapReduce.

Like Apache Spark, GraphX initially started as a research project at UC Berkeley's AMPLab and Databricks, and was later donated to the Apache Software Foundation and the Spark project.

GraphX is a graph computation engine built on top of Spark that enables users to interactively build, transform and reason about graph structured data at scale. It comes complete with a library of common algorithms.

**Downloading**

Get Spark from the [downloads page](http://spark.apache.org/downloads.html) of the project website. This documentation is for Spark version 2.2.0. Spark uses Hadoop’s client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a “Hadoop free” binary and run Spark with any Hadoop version [by augmenting Spark’s classpath](https://spark.apache.org/docs/latest/hadoop-provided.html). Scala and Java users can include Spark in their projects using its Maven coordinates and in the future Python users can also install Spark from PyPI.

If you’d like to build Spark from source, visit [Building Spark](https://spark.apache.org/docs/latest/building-spark.html).

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS). It’s easy to run locally on one machine — all you need is to have javainstalled on your system PATH, or the JAVA\_HOME environment variable pointing to a Java installation.

Spark runs on Java 8+, Python 2.7+/3.4+ and R 3.1+. For the Scala API, Spark 2.2.0 uses Scala 2.11. You will need to use a compatible Scala version (2.11.x).

Note that support for Java 7, Python 2.6 and old Hadoop versions before 2.6.5 were removed as of Spark 2.2.0.

Note that support for Scala 2.10 is deprecated as of Spark 2.1.0, and may be removed in Spark 2.3.0.

**Running the Examples and Shell**

Spark comes with several sample programs. Scala, Java, Python and R examples are in the examples/src/main directory. To run one of the Java or Scala sample programs, use bin/run-example <class> [params] in the top-level Spark directory. (Behind the scenes, this invokes the more general [spark-submit script](https://spark.apache.org/docs/latest/submitting-applications.html) for launching applications). For example,

./bin/run-example SparkPi 10

You can also run Spark interactively through a modified version of the Scala shell. This is a great way to learn the framework.

./bin/spark-shell --master local[2]

The --master option specifies the [master URL for a distributed cluster](https://spark.apache.org/docs/latest/submitting-applications.html#master-urls), or local to run locally with one thread, or local[N] to run locally with N threads. You should start by using local for testing. For a full list of options, run Spark shell with the --help option.

Spark also provides a Python API. To run Spark interactively in a Python interpreter, use bin/pyspark:

./bin/pyspark --master local[2]

Example applications are also provided in Python. For example,

./bin/spark-submit examples/src/main/python/pi.py 10

Spark also provides an experimental [R API](https://spark.apache.org/docs/latest/sparkr.html) since 1.4 (only DataFrames APIs included). To run Spark interactively in a R interpreter, use bin/sparkR:

./bin/sparkR --master local[2]

Example applications are also provided in R. For example,

./bin/spark-submit examples/src/main/r/dataframe.R

**Launching on a Cluster**

The Spark [cluster mode overview](https://spark.apache.org/docs/latest/cluster-overview.html) explains the key concepts in running on a cluster. Spark can run both by itself, or over several existing cluster managers. It currently provides several options for deployment:

* [Standalone Deploy Mode](https://spark.apache.org/docs/latest/spark-standalone.html): simplest way to deploy Spark on a private cluster
* [Apache Mesos](https://spark.apache.org/docs/latest/running-on-mesos.html)
* [Hadoop YARN](https://spark.apache.org/docs/latest/running-on-yarn.html)

**Where to Go from Here**

**Programming Guides:**

* [Quick Start](https://spark.apache.org/docs/latest/quick-start.html): a quick introduction to the Spark API; start here!
* [Spark Programming Guide](https://spark.apache.org/docs/latest/programming-guide.html): detailed overview of Spark in all supported languages (Scala, Java, Python, R)
* Modules built on Spark:
  + [Spark Streaming](https://spark.apache.org/docs/latest/streaming-programming-guide.html): processing real-time data streams
  + [Spark SQL, Datasets, and DataFrames](https://spark.apache.org/docs/latest/sql-programming-guide.html): support for structured data and relational queries
  + [MLlib](https://spark.apache.org/docs/latest/ml-guide.html): built-in machine learning library
  + [GraphX](https://spark.apache.org/docs/latest/graphx-programming-guide.html): Spark’s new API for graph processing

**API Docs:**

* [Spark Scala API (Scaladoc)](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package)
* [Spark Java API (Javadoc)](https://spark.apache.org/docs/latest/api/java/index.html)
* [Spark Python API (Sphinx)](https://spark.apache.org/docs/latest/api/python/index.html)
* [Spark R API (Roxygen2)](https://spark.apache.org/docs/latest/api/R/index.html)

**Deployment Guides:**

* [Cluster Overview](https://spark.apache.org/docs/latest/cluster-overview.html): overview of concepts and components when running on a cluster
* [Submitting Applications](https://spark.apache.org/docs/latest/submitting-applications.html): packaging and deploying applications
* Deployment modes:
  + [Amazon EC2](https://github.com/amplab/spark-ec2): scripts that let you launch a cluster on EC2 in about 5 minutes
  + [Standalone Deploy Mode](https://spark.apache.org/docs/latest/spark-standalone.html): launch a standalone cluster quickly without a third-party cluster manager
  + [Mesos](https://spark.apache.org/docs/latest/running-on-mesos.html): deploy a private cluster using [Apache Mesos](http://mesos.apache.org/)
  + [YARN](https://spark.apache.org/docs/latest/running-on-yarn.html): deploy Spark on top of Hadoop NextGen (YARN)
  + [Kubernetes (experimental)](https://github.com/apache-spark-on-k8s/spark): deploy Spark on top of Kubernetes

**Other Documents:**

* [Configuration](https://spark.apache.org/docs/latest/configuration.html): customize Spark via its configuration system
* [Monitoring](https://spark.apache.org/docs/latest/monitoring.html): track the behavior of your applications
* [Tuning Guide](https://spark.apache.org/docs/latest/tuning.html): best practices to optimize performance and memory use
* [Job Scheduling](https://spark.apache.org/docs/latest/job-scheduling.html): scheduling resources across and within Spark applications
* [Security](https://spark.apache.org/docs/latest/security.html): Spark security support
* [Hardware Provisioning](https://spark.apache.org/docs/latest/hardware-provisioning.html): recommendations for cluster hardware
* Integration with other storage systems:
  + [OpenStack Swift](https://spark.apache.org/docs/latest/storage-openstack-swift.html)
* [Building Spark](https://spark.apache.org/docs/latest/building-spark.html): build Spark using the Maven system
* [Contributing to Spark](http://spark.apache.org/contributing.html)
* [Third Party Projects](http://spark.apache.org/third-party-projects.html): related third party Spark projects

**External Resources:**

* [Spark Homepage](http://spark.apache.org/)
* [Spark Community](http://spark.apache.org/community.html) resources, including local meetups
* [StackOverflow tag apache-spark](http://stackoverflow.com/questions/tagged/apache-spark)
* [Mailing Lists](http://spark.apache.org/mailing-lists.html): ask questions about Spark here
* [AMP Camps](http://ampcamp.berkeley.edu/): a series of training camps at UC Berkeley that featured talks and exercises about Spark, Spark Streaming, Mesos, and more. [Videos](http://ampcamp.berkeley.edu/6/), [slides](http://ampcamp.berkeley.edu/6/) and [exercises](http://ampcamp.berkeley.edu/6/exercises/) are available online for free.
* [Code Examples](http://spark.apache.org/examples.html): more are also available in the examples subfolder of Spark ([Scala](https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples), [Java](https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples), [Python](https://github.com/apache/spark/tree/master/examples/src/main/python), [R](https://github.com/apache/spark/tree/master/examples/src/main/r))

**Quick Start**

* [Interactive Analysis with the Spark Shell](https://spark.apache.org/docs/latest/quick-start.html#interactive-analysis-with-the-spark-shell)
  + [Basics](https://spark.apache.org/docs/latest/quick-start.html#basics)
  + [More on Dataset Operations](https://spark.apache.org/docs/latest/quick-start.html#more-on-dataset-operations)
  + [Caching](https://spark.apache.org/docs/latest/quick-start.html#caching)
* [Self-Contained Applications](https://spark.apache.org/docs/latest/quick-start.html#self-contained-applications)
* [Where to Go from Here](https://spark.apache.org/docs/latest/quick-start.html#where-to-go-from-here)

This tutorial provides a quick introduction to using Spark. We will first introduce the API through Spark’s interactive shell (in Python or Scala), then show how to write applications in Java, Scala, and Python.

To follow along with this guide, first download a packaged release of Spark from the [Spark website](http://spark.apache.org/downloads.html). Since we won’t be using HDFS, you can download a package for any version of Hadoop.

Note that, before Spark 2.0, the main programming interface of Spark was the Resilient Distributed Dataset (RDD). After Spark 2.0, RDDs are replaced by Dataset, which is strongly-typed like an RDD, but with richer optimizations under the hood. The RDD interface is still supported, and you can get a more complete reference at the [RDD programming guide](https://spark.apache.org/docs/latest/rdd-programming-guide.html). However, we highly recommend you to switch to use Dataset, which has better performance than RDD. See the [SQL programming guide](https://spark.apache.org/docs/latest/sql-programming-guide.html) to get more information about Dataset.

**Interactive Analysis with the Spark Shell**

**Basics**

Spark’s shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. It is available in either Scala (which runs on the Java VM and is thus a good way to use existing Java libraries) or Python. Start it by running the following in the Spark directory:

* [**Scala**](https://spark.apache.org/docs/latest/quick-start.html#tab_scala_0)
* [**Python**](https://spark.apache.org/docs/latest/quick-start.html#tab_python_0)

./bin/pyspark

Spark’s primary abstraction is a distributed collection of items called a Dataset. Datasets can be created from Hadoop InputFormats (such as HDFS files) or by transforming other Datasets. Due to Python’s dynamic nature, we don’t need the Dataset to be strongly-typed in Python. As a result, all Datasets in Python are Dataset[Row], and we call it DataFrame to be consistent with the data frame concept in Pandas and R. Let’s make a new DataFrame from the text of the README file in the Spark source directory:

>>> textFile = spark.read.text("README.md")

You can get values from DataFrame directly, by calling some actions, or transform the DataFrame to get a new one. For more details, please read the [*API doc*](https://spark.apache.org/docs/latest/api/python/index.html#pyspark.sql.DataFrame).

>>> textFile.count() *# Number of rows in this DataFrame*

126

>>> textFile.first() *# First row in this DataFrame*

Row(value=u'# Apache Spark')

Now let’s transform this DataFrame to a new one. We call filter to return a new DataFrame with a subset of the lines in the file.

>>> linesWithSpark = textFile.filter(textFile.value.contains("Spark"))

We can chain together transformations and actions:

>>> textFile.filter(textFile.value.contains("Spark")).count() *# How many lines contain "Spark"?*

15

**More on Dataset Operations**

Dataset actions and transformations can be used for more complex computations. Let’s say we want to find the line with the most words:

* [**Scala**](https://spark.apache.org/docs/latest/quick-start.html#tab_scala_1)
* [**Python**](https://spark.apache.org/docs/latest/quick-start.html#tab_python_1)

>>> **from** **pyspark.sql.functions** **import** \*

>>> textFile.select(size(split(textFile.value, "\s+")).name("numWords")).agg(max(col("numWords"))).collect()

[Row(max(numWords)=15)]

This first maps a line to an integer value and aliases it as “numWords”, creating a new DataFrame. agg is called on that DataFrame to find the largest word count. The arguments to select and agg are both [*Column*](https://spark.apache.org/docs/latest/api/python/index.html#pyspark.sql.Column), we can use df.colName to get a column from a DataFrame. We can also import pyspark.sql.functions, which provides a lot of convenient functions to build a new Column from an old one.

One common data flow pattern is MapReduce, as popularized by Hadoop. Spark can implement MapReduce flows easily:

>>> wordCounts = textFile.select(explode(split(textFile.value, "\s+")).**as**("word")).groupBy("word").count()

Here, we use the explode function in select, to transfrom a Dataset of lines to a Dataset of words, and then combine groupBy and count to compute the per-word counts in the file as a DataFrame of 2 columns: “word” and “count”. To collect the word counts in our shell, we can call collect:

>>> wordCounts.collect()

[Row(word=u'online', count=1), Row(word=u'graphs', count=1), ...]

**Caching**

Spark also supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly, such as when querying a small “hot” dataset or when running an iterative algorithm like PageRank. As a simple example, let’s mark our linesWithSpark dataset to be cached:

* [**Scala**](https://spark.apache.org/docs/latest/quick-start.html#tab_scala_2)
* [**Python**](https://spark.apache.org/docs/latest/quick-start.html#tab_python_2)

>>> linesWithSpark.cache()

>>> linesWithSpark.count()

15

>>> linesWithSpark.count()

15

It may seem silly to use Spark to explore and cache a 100-line text file. The interesting part is that these same functions can be used on very large data sets, even when they are striped across tens or hundreds of nodes. You can also do this interactively by connecting bin/pyspark to a cluster, as described in the [RDD programming guide](https://spark.apache.org/docs/latest/rdd-programming-guide.html#using-the-shell).

**Self-Contained Applications**

Suppose we wish to write a self-contained application using the Spark API. We will walk through a simple application in Scala (with sbt), Java (with Maven), and Python.

* [**Scala**](https://spark.apache.org/docs/latest/quick-start.html#tab_scala_3)
* [**Java**](https://spark.apache.org/docs/latest/quick-start.html#tab_java_3)
* [**Python**](https://spark.apache.org/docs/latest/quick-start.html#tab_python_3)

Now we will show how to write an application using the Python API (PySpark).

As an example, we’ll create a simple Spark application, SimpleApp.py:

*"""SimpleApp.py"""*

**from** **pyspark.sql** **import** SparkSession

logFile = "YOUR\_SPARK\_HOME/README.md" *# Should be some file on your system*

spark = SparkSession.builder().appName(appName).master(master).getOrCreate()

logData = spark.read.text(logFile).cache()

numAs = logData.filter(logData.value.contains('a')).count()

numBs = logData.filter(logData.value.contains('b')).count()

**print**("Lines with a: *%i*, lines with b: *%i*" % (numAs, numBs))

spark.stop()

This program just counts the number of lines containing ‘a’ and the number containing ‘b’ in a text file. Note that you’ll need to replace YOUR\_SPARK\_HOME with the location where Spark is installed. As with the Scala and Java examples, we use a SparkSession to create Datasets. For applications that use custom classes or third-party libraries, we can also add code dependencies to spark-submit through its --py-files argument by packaging them into a .zip file (see spark-submit --help for details). SimpleApp is simple enough that we do not need to specify any code dependencies.

We can run this application using the bin/spark-submit script:

*# Use spark-submit to run your application*

$ YOUR\_SPARK\_HOME/bin/spark-submit **\**

--master local[4] **\**

SimpleApp.py

...

Lines with a: 46, Lines with b: 23

**Where to Go from Here**

Congratulations on running your first Spark application!

* For an in-depth overview of the API, start with the [RDD programming guide](https://spark.apache.org/docs/latest/rdd-programming-guide.html) and the [SQL programming guide](https://spark.apache.org/docs/latest/sql-programming-guide.html), or see “Programming Guides” menu for other components.
* For running applications on a cluster, head to the [deployment overview](https://spark.apache.org/docs/latest/cluster-overview.html).
* Finally, Spark includes several samples in the examples directory ([Scala](https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples), [Java](https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples), [Python](https://github.com/apache/spark/tree/master/examples/src/main/python), [R](https://github.com/apache/spark/tree/master/examples/src/main/r)). You can run them as follows:

*# For Scala and Java, use run-example:*

./bin/run-example SparkPi

*# For Python examples, use spark-submit directly:*

./bin/spark-submit examples/src/main/python/pi.py

*# For R examples, use spark-submit directly:*

./bin/spark-submit examples/src/main/r/dataframe.R

What is Spark

[Apache Spark](https://spark.apache.org/) is an open source big data processing framework built around speed, ease of use, and sophisticated analytics. It was originally developed in 2009 in UC Berkeley’s AMPLab, and open sourced in 2010 as an Apache project.

Spark has several advantages compared to other big data and MapReduce technologies like Hadoop and Storm.

**Related Vendor Content**

[**Making the Move to Public Cloud: Best Practices**](https://www.infoq.com/vendorcontent/show.action?vcr=4575&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-introduction&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**How to build a CI/CD pipeline with Terraform**](https://www.infoq.com/vendorcontent/show.action?vcr=4403&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-introduction&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**All-in-one Monitoring: Introduction to Site24x7 for Cloud, APM, and Server Monitoring**](https://www.infoq.com/vendorcontent/show.action?vcr=4558&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-introduction&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Java Performance: Top 10 Problems**](https://www.infoq.com/vendorcontent/show.action?vcr=4502&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-introduction&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

**Related Sponsor**

Hadoop and Spark

Hadoop as a big data processing technology has been around for 10 years and has proven to be the solution of choice for processing large data sets. MapReduce is a great solution for one-pass computations, but not very efficient for use cases that require multi-pass computations and algorithms. Each step in the data processing workflow has one Map phase and one Reduce phase and you'll need to convert any use case into MapReduce pattern to leverage this solution.

The Job output data between each step has to be stored in the distributed file system before the next step can begin. Hence, this approach tends to be slow due to replication & disk storage. Also, Hadoop solutions typically include clusters that are hard to set up and manage. It also requires the integration of several tools for different big data use cases (like Mahout for Machine Learning and Storm for streaming data processing).

If you wanted to do something complicated, you would have to string together a series of MapReduce jobs and execute them in sequence. Each of those jobs was high-latency, and none could start until the previous job had finished completely.

Spark allows programmers to develop complex, multi-step data pipelines using directed acyclic graph ([DAG](http://en.wikipedia.org/wiki/Directed_acyclic_graph)) pattern. It also supports in-memory data sharing across DAGs, so that different jobs can work with the same data.

Spark runs on top of existing Hadoop Distributed File System ([HDFS](http://wiki.apache.org/hadoop/HDFS)) infrastructure to provide enhanced and additional functionality. It provides support for [deploying Spark applications](http://databricks.com/blog/2014/01/21/Spark-and-Hadoop.html) in an existing Hadoop v1 cluster (with SIMR – Spark-Inside-MapReduce) or Hadoop v2 YARN cluster or even [Apache Mesos](http://mesos.apache.org/).

We should look at Spark as an alternative to Hadoop MapReduce rather than a replacement to Hadoop. It’s not intended to replace Hadoop but to provide a comprehensive and unified solution to manage different big data use cases and requirements.

Spark Features

Spark takes MapReduce to the next level with less expensive shuffles in the data processing. With capabilities like in-memory data storage and near real-time processing, the performance can be several times faster than other big data technologies.

Spark also supports lazy evaluation of big data queries, which helps with optimization of the steps in data processing workflows. It provides a higher level API to improve developer productivity and a consistent architect model for big data solutions.

Spark holds intermediate results in memory rather than writing them to disk which is very useful especially when you need to work on the same dataset multiple times. It’s designed to be an execution engine that works both in-memory and on-disk. Spark operators perform external operations when data does not fit in memory. Spark can be used for processing datasets that larger than the aggregate memory in a cluster.

Spark will attempt to store as much as data in memory and then will spill to disk. It can store part of a data set in memory and the remaining data on the disk. You have to look at your data and use cases to assess the memory requirements. With this in-memory data storage, Spark comes with performance advantage.

Other Spark features include:

* Supports more than just Map and Reduce functions.
* Optimizes arbitrary operator graphs.
* Lazy evaluation of big data queries which helps with the optimization of the overall data processing workflow.
* Provides concise and consistent APIs in Scala, Java and Python.
* Offers interactive shell for Scala and Python. This is not available in Java yet.

Spark is written in [Scala Programming Language](http://www.scala-lang.org/) and runs on Java Virtual Machine (JVM) environment. It currently supports the following languages for developing applications using Spark:

* Scala
* Java
* Python
* Clojure
* R

Spark Ecosystem

Other than Spark Core API, there are additional libraries that are part of the Spark ecosystem and provide additional capabilities in Big Data analytics and Machine Learning areas.

These libraries include:

* **Spark Streaming:**
  + [Spark Streaming](https://spark.apache.org/streaming/) can be used for processing the real-time streaming data. This is based on micro batch style of computing and processing. It uses the DStream which is basically a series of RDDs, to process the real-time data.
* **Spark SQL:**
  + [Spark SQL](https://spark.apache.org/sql/) provides the capability to expose the Spark datasets over JDBC API and allow running the SQL like queries on Spark data using traditional BI and visualization tools. Spark SQL allows the users to ETL their data from different formats it’s currently in (like JSON, Parquet, a Database), transform it, and expose it for ad-hoc querying.
* **Spark MLlib:**
  + [MLlib](https://spark.apache.org/mllib/) is Spark’s scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.
* **Spark GraphX:**
  + [GraphX](https://spark.apache.org/graphx/) is the new (alpha) Spark API for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing the Resilient Distributed Property Graph: a directed multi-graph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

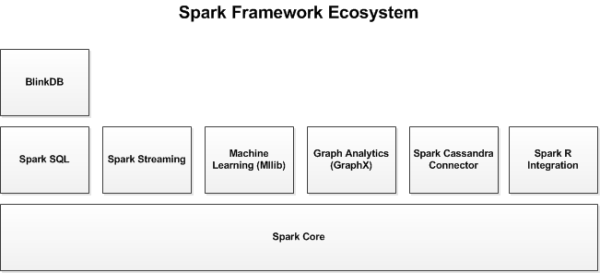
Outside of these libraries, there are others like BlinkDB and Tachyon.

[BlinkDB](http://blinkdb.org/) is an approximate query engine and can be used for running interactive SQL queries on large volumes of data. It allows users to trade-off query accuracy for response time. It works on large data sets by running queries on data samples and presenting results annotated with meaningful error bars.

[Tachyon](http://tachyon-project.org/index.html) is a memory-centric distributed file system enabling reliable file sharing at memory-speed across cluster frameworks, such as Spark and MapReduce. It caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

And there are also integration adapters with other products like Cassandra ([Spark Cassandra Connector](http://www.datastax.com/dev/blog/accessing-cassandra-from-spark-in-java)) and R (SparkR). With Cassandra Connector, you can use Spark to access data stored in a Cassandra database and perform data analytics on that data.

Following diagram (Figure 1) shows how these different libraries in Spark ecosystem are related to each other.



**Figure 1. Spark Framework Libraries**

We'll explore these libraries in future articles in this series.

Spark Architecture

Spark Architecture includes following three main components:

* Data Storage
* API
* Management Framework

Let’s look at each of these components in more detail.

**Data Storage:**

Spark uses HDFS file system for data storage purposes. It works with any Hadoop compatible data source including HDFS, HBase, Cassandra, etc.

**API:**

The API provides the application developers to create Spark based applications using a standard API interface. Spark provides API for Scala, Java, and Python programming languages.

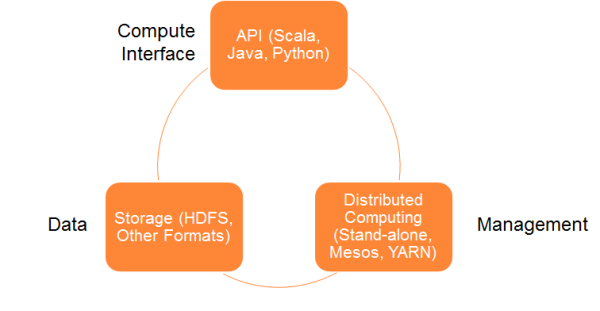
Following are the website links for the Spark API for each of these languages.

* [**Scala API**](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package)
* [**Java**](http://spark.apache.org/docs/latest/api/java/index.html)
* [**Python**](http://spark.apache.org/docs/latest/api/python/index.html)

**Resource Management:**

Spark can be deployed as a Stand-alone server or it can be on a distributed computing framework like Mesos or YARN.

Figure 2 below shows these components of Spark architecture model.



**Figure 2. Spark Architecture**

Resilient Distributed Datasets

[Resilient Distributed Dataset](https://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds) (based on Matei’s [research paper](https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf)) or RDD is the core concept in Spark framework. Think about RDD as a table in a database. It can hold any type of data. Spark stores data in RDD on different partitions.

They help with rearranging the computations and optimizing the data processing.

They are also fault tolerance because an RDD know how to recreate and recompute the datasets.

RDDs are immutable. You can modify an RDD with a transformation but the transformation returns you a new RDD whereas the original RDD remains the same.

RDD supports two types of operations:

* Transformation
* Action

**Transformation:** [Transformations](https://spark.apache.org/docs/latest/programming-guide.html#transformations) don't return a single value, they return a new RDD. Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD.

Some of the Transformation functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce.

**Action:** [Action](https://spark.apache.org/docs/latest/programming-guide.html#actions) operation evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned.

Some of the Action operations are reduce, collect, count, first, take, countByKey, and foreach.

How to Install Spark

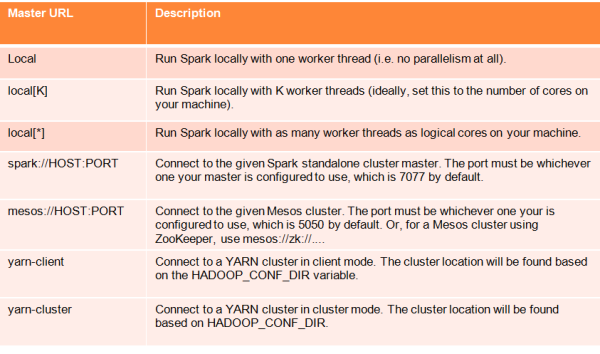
There are few different to install and use Spark. You can install it on your machine as a stand-alone framework or use one of Spark Virtual Machine (VM) images available from vendors like [Cloudera](http://www.cloudera.com/content/support/en/downloads/quickstart_vms/cdh-5-1-x.html), HortonWorks, or MapR. Or you can also use Spark installed and configured in the cloud (like [Databricks Cloud](http://databricks.com/product)).

In this article, we’ll install Spark as a stand-alone framework and launch it locally. Spark 1.2.0 version was released recently. We’ll use this version for sample application code demonstration.

How to Run Spark

When you install Spark on the local machine or use a Cloud based installation, there are few different modes you can connect to Spark engine.

The following table shows the Master URL parameter for the different modes of running Spark.



How to Interact with Spark

Once Spark is up and running, you can connect to it using the Spark shell for interactive data analysis. Spark Shell is available in both Scala and Python languages. Java doesn’t support an interactive shell yet, so this feature is currently not available in Java.

You use the commands spark-shell.cmd and pyspark.cmd to run Spark Shell using Scala and Python respectively.

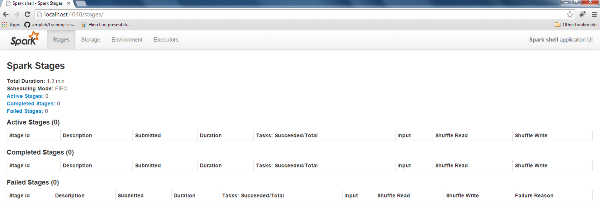
Spark Web Console

When Spark is running in any mode, you can view the Spark job results and other statistics by accessing Spark Web Console via the following URL:

http://localhost:4040

Spark Console is shown in Figure 3 below with tabs for Stages, Storage, Environment, and Executors.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-introduction/en/resources/4.png)

**Figure 3. Spark Web Console**

Shared Variables

Spark provides two types of shared variables to make it efficient to run the Spark programs in a cluster. These are Broadcast Variables and Accumulators.

**Broadcast Variables:** Broadcast variables allow to keep read-only variable cached on each machine instead of sending a copy of it with tasks. They can be used to give the nodes in the cluster copies of large input datasets more efficiently.

Following code snippet shows how to use the broadcast variables.

//

// Broadcast Variables

//

val broadcastVar = sc.broadcast(Array(1, 2, 3))

broadcastVar.value

**Accumulators:** Accumulators are only added using an associative operation and can therefore be efficiently supported in parallel. They can be used to implement counters (as in MapReduce) or sums. Tasks running on the cluster can add to an accumulator variable using the add method. However, they cannot read its value. Only the driver program can read the accumulator's value.

The code snippet below shows how to use Accumulator shared variable:

//

// Accumulators

//

val accum = sc.accumulator(0, "My Accumulator")

sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value

Sample Spark Application

The sample application I cover in this article is a simple Word Count application. This is the same example one would cover when they are learning Big Data processing with Hadoop. We’ll perform some data analytics queries on a text file. The text file and the data set in this example are small, but same Spark queries can be used for large size data sets, without any modifications in the code.

To keep the discussion simple, we’ll use the Spark Scala Shell.

First, let’s look at how to install Spark on your local machine.

**Pre-Requisites:**

* You will need Java Development Kit (JDK) installed for Spark to work locally. This is covered in Step 1 below.
* You will also need to install Spark software on your laptop. The instructions on how to do this are covered in the Step 2 below.

**Note:** These instructions are for Windows environment. If you are using a different operating system environment, you'll need to modify the system variables and directory paths to match your environment.

**I. INSTALL JDK:**

1) Download JDK from Oracle website.[JDK version 1.7](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html) is recommended.

Install JDK in a directory name without spaces. For Windows users, install JDK in a folder like c:\dev, not in "c:\Program Files". "Program Files" directory has a space in the name and this causes problems when software is installed in this folder.

**NOTE:** **DO NOT INSTALL** JDK or Spark Software (described in Step 2) in "c:\Program Files" directory.

2) After installing JDK, verify it was installed correctly by navigating to "bin" folder under JDK 1.7 directory and typing the following command:

java -version

If JDK is installed correctly, the above command would display the Java version.

**II. INSTALL SPARK SOFTWARE:**

Download the latest Spark version from [Spark website](https://spark.apache.org/downloads.html). Latest version at the time of publication of this article is Spark 1.2. You can choose a specific Spark installation depending on the Hadoop version. I downloaded Spark for Hadoop 2.4 or later, and the file name is spark-1.2.0-bin-hadoop2.4.tgz.

Unzip the installation file to a local directory (For example, c:\dev).

To verify Spark installation, navigate to spark directory and launch Spark Shell using the following commands. This is for Windows. If you are using Linux or Mac OS, please edit the commands to work on your OS.

c:

cd c:\dev\spark-1.2.0-bin-hadoop2.4

bin\spark-shell

If Spark was installed correctly, you should the see the following messages in the output on the console.

….

15/01/17 23:17:46 INFO HttpServer: Starting HTTP Server

15/01/17 23:17:46 INFO Utils: Successfully started service 'HTTP class server' on port 58132.

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_\_/ .\_\_/\\_,\_/\_/ /\_/\\_\ version 1.2.0

/\_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0\_71)

Type in expressions to have them evaluated.

Type :help for more information.

….

15/01/17 23:17:53 INFO BlockManagerMaster: Registered BlockManager

15/01/17 23:17:53 INFO SparkILoop: Created spark context..

Spark context available as sc.

You can type the following commands to check if Spark Shell is working correctly.

sc.version

(or)

sc.appName

After this step, you can exit the Spark Shell window by typing the following command:

:quit

To launch Spark Python Shell, you need to have Python installed on your machine. You can download and install [Anaconda](http://continuum.io/downloads) which is a free Python distribution and includes several popular Python packages for science, math, engineering, and data analysis.

Then you can run the following commands:

c:

cd c:\dev\spark-1.2.0-bin-hadoop2.4

bin\pyspark

Word Count Application

Once you have Spark installed and have it up and running, you can run the data analytics queries using Spark API.

These are simple commands to read the data from a text file and process it. We’ll look at advanced use cases of using Spark framework in the future articles in this series.

First, let’s use Spark API to run the popular Word Count example. Open a new Spark Scala Shell if you don’t already have it running. Here are the commands for this example.

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

val txtFile = "README.md"

val txtData = sc.textFile(txtFile)

txtData.cache()

We call the cache function to store the RDD created in the above step in the cache, so Spark doesn’t have to compute it every time we use it for further data queries. Note that cache() is a lazy operation. Spark doesn’t immediately store the data in memory when we call cache. It actually takes place when an action is called on an RDD.

Now, we can call the count function to see how many lines are there in the text file.

txtData.count()

Now, we can run the following commands to perform the word count. The count shows up next to each word in the text file.

val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)

wcData.collect().foreach(println)

If you want to look at more code examples of using Spark Core API, checkout [Spark documentation](http://spark.apache.org/docs/latest/programming-guide.html) on their website.

What's Next

In the future articles of this series, we'll learn more about other parts of Spark ecosytem starting with Spark SQL. Later, we'll look at Spark Streaming, Spark MLlib, and Spark GraphX. We'll also look at the upcoming frameworks like Tachyon and BlinkDB.

Conclusions

In this article, we looked at how Apache Spark framework helps with big data processing and analytics with its standard API. We also looked at how Spark compares with traditional MapReduce implementation like Apache Hadoop. Spark is based on the same HDFS file storage system as Hadoop, so you can use Spark and MapReduce together if you already have significant investment and infrastructure setup with Hadoop.

You can also combine the Spark processing with Spark SQL, Machine Learning and Spark Streaming as we’ll see in a future article.

With several integrations and adapters on Spark, you can combine other technologies with Spark. An example of this is to use Spark, Kafka, and Apache Cassandra together where Kafka can be used for the streaming data coming in, Spark to do the computation, and finally Cassandra NoSQL database to store the computation result data.

But keep in mind, Spark is a less mature ecosystem and needs further improvements in areas like security and integration with BI tools.

In the [previous article](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal) of the Apache Spark article series, we learned what Apache Spark framework is and how it helps with big data processing analytics needs in the organizations.

Spark SQL, part of Apache Spark big data framework, is used for structured data processing and allows running SQL like queries on Spark data. We can perform ETL on the data from different formats like JSON, Parquet, Database) and then run ad-hoc querying.

**Related Vendor Content**

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

[**Modernize your Enterprise Linux Data with SQL Server 2017**](https://www.infoq.com/vendorcontent/show.action?vcr=4561&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-sql%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**AWS Security Fitness Guide: 11 Exercises to Get CloudFit**](https://www.infoq.com/vendorcontent/show.action?vcr=4539&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-sql%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Microservices vs. Service-Oriented Architecture (By O'Reilly) - Download Now**](https://www.infoq.com/vendorcontent/show.action?vcr=4576&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-sql%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**4 Things to Consider When Delivering Apps in Hybrid or Multi-Cloud Environments**](https://www.infoq.com/vendorcontent/show.action?vcr=4535&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-sql%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

**Related Sponsor**

[](https://www.infoq.com/infoq/url.action?i=16688&t=f)

Fastly’s edge cloud platform powers secure, fast and reliable online experiences for the world’s most popular digital businesses. [**See for yourself.**](https://www.infoq.com/infoq/url.action?i=16687&t=f)

In this second installment of the article series, we'll look at the Spark SQL library, how it can be used for executing SQL queries against the data stored in batch files, JSON data sets, or Hive tables.

Spark 1.3 is the latest version of the big data framework which was [released](http://www.infoq.com/news/2015/03/apache-spark-1.3-released) last month. Prior to this version, Spark SQL module has been in an “Alpha” status but now the team has removed that label from the library. This release includes several new features some of which are listed below:

* **DataFrame:** The new release provides a programming abstraction called DataFrames which can act as distributed SQL query engine.
* **Data Sources:** With the addition of the data sources API, Spark SQL now makes it easier to compute over structured data stored in a wide variety of formats, including Parquet, JSON, and Apache Avro library.
* **JDBC Server:** The built-in JDBC server makes it easy to connect to the structured data stored in relational database tables and perform big data analytics using the traditional BI tools.

Spark SQL Components

The two main components when using Spark SQL are DataFrame and SQLContext.

Let’s look at DataFrame first.

DataFrame

A [DataFrame](https://spark.apache.org/docs/1.3.0/api/scala/index.html" \l "org.apache.spark.sql.DataFrame) is a distributed collection of data organized into named columns. It is based on the data frame concept in R language and is similar to a database table in a relational database.

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

DataFrames can be converted to RDDs by calling the [rdd method](https://spark.apache.org/docs/1.3.0/api/scala/index.html" \l "org.apache.spark.sql.DataFrame) which returns the content of the DataFrame as an RDD of Rows.

DataFrames can be created from different data sources such as:

* Existing RDDs
* Structured data files
* JSON datasets
* Hive tables
* External databases

Spark SQL and DataFrame API are available in the following programming languages:

* Scala (https://spark.apache.org/docs/1.3.0/api/scala/index.html#org.apache.spark.sql.package
* Java (https://spark.apache.org/docs/1.3.0/api/java/index.html?org/apache/spark/sql/api/java/package-summary.html)
* Python (https://spark.apache.org/docs/1.3.0/api/python/pyspark.sql.html)

Spark SQL code examples we discuss in this article use the Spark Scala Shell program.

SQLContext

Spark SQL provides [SQLContext](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.sql.SQLContext) to encapsulate all relational functionality in Spark. You create the SQLContext from the existing SparkContext that we have seen in the previous examples. Following code snippet shows how to create a SQLContext object.

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

There is also [HiveContext](https://spark.apache.org/docs/1.3.0/api/scala/index.html" \l "org.apache.spark.sql.hive.HiveContext) which provides a superset of the functionality provided by SQLContext. It can be used to write queries using the HiveQL parser and read data from Hive tables.

Note that you don't need an existing Hive environment to use the HiveContext in Spark programs.

JDBC Datasource

Other features in Spark SQL library include the data sources including the JDBC data source.

JDBC data source can be used to read data from relational databases using JDBC API. This approach is preferred over using the [JdbcRDD](https://spark.apache.org/docs/1.3.0/api/scala/index.html" \l "org.apache.spark.rdd.JdbcRDD) because the data source returns the results as a DataFrame which can be processed in Spark SQL or joined with other data sources.

Sample Spark SQL Application

In the previous article, we learned how to install the Spark framework on the local machine, how to launch it and interact with it using Spark Scala Shell program. To install the latest version of Spark, download the software from their [website](http://spark.apache.org/downloads.html).

For the code examples in this article, we will use the same Spark Shell to execute the Spark SQL programs. These code examples are for Windows environment. If you are using

To make sure Spark Shell program has enough memory, use the driver-memory command line argument when running spark-shell, as shown in the following command.

spark-shell.cmd --driver-memory 1G

Spark SQL Application

Once you have Spark Shell launched, you can run the data analytics queries using Spark SQL API.

In the first example, we’ll load the customer data from a text file and create a DataFrame object from the dataset. Then we can run DataFrame functions as specific queries to select the data.

Let’s look at the contents of the text file called customers.txt shown below.

100, John Smith, Austin, TX, 78727

200, Joe Johnson, Dallas, TX, 75201

300, Bob Jones, Houston, TX, 77028

400, Andy Davis, San Antonio, TX, 78227

500, James Williams, Austin, TX, 78727

Following code snippet shows the Spark SQL commands you can run on the Spark Shell console.

// Create the SQLContext first from the existing Spark Context

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

// Import statement to implicitly convert an RDD to a DataFrame

import sqlContext.implicits.\_

// Create a custom class to represent the Customer

case class Customer(customer\_id: Int, name: String, city: String, state: String, zip\_code: String)

// Create a DataFrame of Customer objects from the dataset text file.

val dfCustomers = sc.textFile("data/customers.txt").map(\_.split(",")).map(p => Customer(p(0).trim.toInt, p(1), p(2), p(3), p(4))).toDF()

// Register DataFrame as a table.

dfCustomers.registerTempTable("customers")

// Display the content of DataFrame

dfCustomers.show()

// Print the DF schema

dfCustomers.printSchema()

// Select customer name column

dfCustomers.select("name").show()

// Select customer name and city columns

dfCustomers.select("name", "city").show()

// Select a customer by id

dfCustomers.filter(dfCustomers("customer\_id").equalTo(500)).show()

// Count the customers by zip code

dfCustomers.groupBy("zip\_code").count().show()

In the above example, the schema is inferred using the reflection. We can also programmatically specify the schema of the dataset. This is useful when the custom classes cannot be defined ahead of time because the structure of data is encoded in a string.

Following code example shows how to specify the schema using the new data type classes StructType, StringType, and StructField.

//

// Programmatically Specifying the Schema

//

// Create SQLContext from the existing SparkContext.

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

// Create an RDD

val rddCustomers = sc.textFile("data/customers.txt")

// The schema is encoded in a string

val schemaString = "customer\_id name city state zip\_code"

// Import Spark SQL data types and Row.

import org.apache.spark.sql.\_

import org.apache.spark.sql.types.\_;

// Generate the schema based on the string of schema

val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))

// Convert records of the RDD (rddCustomers) to Rows.

val rowRDD = rddCustomers.map(\_.split(",")).map(p => Row(p(0).trim,p(1),p(2),p(3),p(4)))

// Apply the schema to the RDD.

val dfCustomers = sqlContext.createDataFrame(rowRDD, schema)

// Register the DataFrames as a table.

dfCustomers.registerTempTable("customers")

// SQL statements can be run by using the sql methods provided by sqlContext.

val custNames = sqlContext.sql("SELECT name FROM customers")

// The results of SQL queries are DataFrames and support all the normal RDD operations.

// The columns of a row in the result can be accessed by ordinal.

custNames.map(t => "Name: " + t(0)).collect().foreach(println)

// SQL statements can be run by using the sql methods provided by sqlContext.

val customersByCity = sqlContext.sql("SELECT name,zip\_code FROM customers ORDER BY zip\_code")

// The results of SQL queries are DataFrames and support all the normal RDD operations.

// The columns of a row in the result can be accessed by ordinal.

customersByCity.map(t => t(0) + "," + t(1)).collect().foreach(println)

You can also load the data from other data sources like JSON data files, Hive tables, or even relational database tables using the JDBC data source.

As you can see, Spark SQL provides a nice SQL interface to interact with data that’s loaded from diverse data sources, using the SQL query syntax which is familiar to the teams. This is especially useful for non-technical project members like data analysts as well as DBAs.

Conclusions

In this article, we looked at how Apache Spark SQL works to provide an SQL interface to interact with Spark data using the familiar SQL query syntax. Spark SQL is a powerful library that non-technical team members like Business and Data Analysts can use to run data analytics in their organizations.

In the next article, we’ll look at the [Spark Streaming library](http://spark.apache.org/streaming/) which can be used for processing real-time data or streaming data. This library is another important part of the overall data processing and management lifecycle in any organization because the streaming data processing gives us the real-time insights into the systems. This is critical for use cases like fraud detection, online trading systems, event processing solutions etc.

Introduction

In the first two articles in “Big Data Processing with Apache Spark” [series](https://www.infoq.com/apache_spark/?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal), we looked at what Apache Spark framework is ([Part 1](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)) and SQL interface to access data using Spark SQL library ([Part 2](http://www.infoq.com/articles/apache-spark-sql?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)).

These solutions are based on processing static data in a batch mode, for example as an hourly or daily job. But what about real-time data streams that need to be processed on the fly to perform analytics and create insights for data driven business decision making?

**Related Vendor Content**

[**AWS Security Fitness Guide: 11 Exercises to Get CloudFit**](https://www.infoq.com/vendorcontent/show.action?vcr=4539&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-streaming%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Tips, tricks and tools for creating your own data-driven app**](https://www.infoq.com/vendorcontent/show.action?vcr=4441&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-streaming%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Production-Ready Microservices (By O'Reilly) - Download Now**](https://www.infoq.com/vendorcontent/show.action?vcr=4577&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-streaming%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Microservices for Java Developers (By O'Reilly) - Download Now**](https://www.infoq.com/vendorcontent/show.action?vcr=4339&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-streaming%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

**Related Sponsor**

[](https://www.infoq.com/infoq/url.action?i=16688&t=f)

Fastly’s edge cloud platform powers secure, fast and reliable online experiences for the world’s most popular digital businesses. [**See for yourself.**](https://www.infoq.com/infoq/url.action?i=16687&t=f)

With streaming data processing, computing is done in real-time as data arrives rather than as a batch. Real-time data processing and analytics is becoming a critical component of the big data strategy for most organizations.

In this article, we'll learn about real-time data analytics using one of the libraries from Apache Spark, called [Spark Streaming](http://spark.apache.org/streaming/).

We'll look at a web server log analytics use case to show how Spark Streaming can help with running analytics on data streams that are generated in a continuous manner.

Streaming Data Analytics

Streaming data is basically a continuous group of data records generated from sources like sensors, server traffic and online searches. Some of the examples of streaming data are user activity on websites, monitoring data, server logs, and other event data.

Streaming data processing applications help with live dashboards, real-time online recommendations, and instant fraud detection.

If we are building applications to collect, process and analyze streaming data in real time, we need to take different design considerations into account than when we are working on applications used to process the static batch data.

There are different streaming data processing frameworks as listed below:

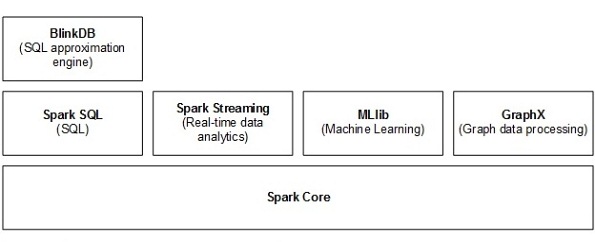
* Apache [Samza](http://samza.apache.org/)
* [Storm](http://storm.apache.org/)
* [Spark Streaming](http://spark.apache.org/streaming/)

We’ll focus on Spark Streaming in this article.

Spark Streaming

Spark Streaming is an extension of core Spark API. Spark Streaming makes it easy to build fault-tolerant processing of real-time data streams.

Figure 1 below shows how Spark Streaming fits into overall Apache Spark ecosystem.



**Figure 1. Spark Ecosystem with Spark Streaming Library**

The way Spark Streaming works is it divides the live stream of data into batches (called microbatches) of a pre-defined interval (N seconds) and then treats each batch of data as [Resilient Distributed Datasets](http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds) (RDDs). Then we can process these RDDs using the operations like map, reduce, reduceByKey, join and window. The results of these RDD operations are returned in batches. We usually store these results into a data store for further analytics and to generate reports and dashboards or sending event based alerts.

It's important to decide the time interval for Spark Streaming, based on your use case and data processing requirements. It the value of N is too low, then the micro-batches will not have enough data to give meaningful results during the analysis.

Compared to Spark Streaming, other stream processing frameworks process the data streams per each event rather than as a micro-batch. With micro-batch approach, we can use other Spark libraries (like Core, Machine Learning etc) with Spark Streaming API in the same application.

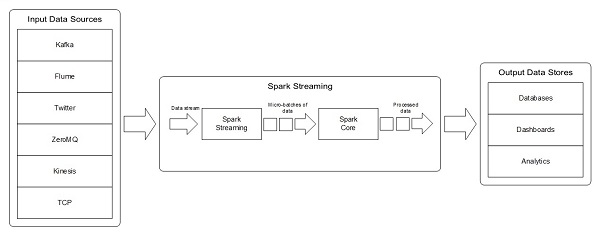
Streaming data can come from many different sources. Some of these data sources include the following:

* [Kafka](http://kafka.apache.org/)
* [Flume](https://flume.apache.org/)
* Twitter
* [ZeroMQ](http://zeromq.org/)
* Amazon’s [Kinesis](http://aws.amazon.com/kinesis/)
* TCP sockets

Another advantage of using a big data processing framework like Apache Spark is that we can combine batch processing and streaming processing in the same system. We can also apply Spark’s machine learning and graph processing algorithms on data streams. We'll discuss Machine Learning and Graph Processing libraries, called [MLlib](http://spark.apache.org/docs/latest/mllib-guide.html) and [GraphX](http://spark.apache.org/docs/latest/graphx-programming-guide.html) respectively, in future articles in this series.

Spark Streaming architecture is shown in Figure 2 below.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig2large.jpg)

**Figure 2. How Spark Streaming works**

Spark Streaming Use Cases

Spark Streaming is becoming the platform of choice to implement data processing and analytics solutions for real-time data received from Internet of Things (IoT) and sensors. It is used in a variety of use cases and business applications.

Some of the most interesting [use cases of Spark Streaming](http://www.datanami.com/2015/11/30/spark-streaming-what-is-it-and-whos-using-it/) include the following:

* [Uber](https://www.uber.com/), the company behind ride sharing service, uses Spark Streaming in their continuous Streaming ETL pipeline to collect terabytes of event data every day from their mobile users for real-time telemetry analytics.
* [Pinterest](https://www.pinterest.com/), the company behind the visual bookmarking tool, [uses Spark Streaming](http://www.infoq.com/news/2015/03/pinterest-memsql-spark-streaming), MemSQL and Apache Kafka technologies to provide insight into how their users are engaging with Pins across the globe in real-time.
* [Netflix](https://www.netflix.com/) uses Kafka and Spark Streaming to build a real-time online movie recommendation and data monitoring [solution](https://spark-summit.org/2015/events/spark-and-spark-streaming-at-netflix/) that processes billions of events received per day from different data sources.

Other real world examples of Spark Streaming include:

* Supply chain analytics
* Real-time security intelligence operations to find threats
* Ad auction platform
* Real-time video analytics to help with personalized, interactive experiences to the viewers

Let’s take a look at Spark Streaming architecture and API methods. To write Spark Streaming programs, there are two components we need to know about: DStream and StreamingContext.

DStream

[DStream](http://spark.apache.org/docs/latest/streaming-programming-guide.html#discretized-streams-dstreams) (short for Discretized Stream) is the basic abstraction in Spark Streaming and represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying operations on other DStreams. Internally, a DStream is represented as a sequence of RDD objects.

Similar to the transformation and action operations on RDDs, DStreams support the following [operations](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.streaming.dstream.DStream):

* map
* flatMap
* filter
* count
* reduce
* countByValue
* reduceByKey
* join
* updateStateByKey

Streaming Context

Similar to [SparkContext](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.SparkContext) in Spark, [StreamingContext](https://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.streaming.StreamingContext) is the main entry point for all streaming functionality.

StreamingContext has built-in methods for receiving streaming data into Spark Streaming program.

Using this context, we can create a DStream that represents streaming data from a TCP source, specified as hostname and port number. For example, if we are using a tool like [netcat](https://nmap.org/ncat/) to test the Spark Streaming program, we would receive data stream from the machine where netcat is running (e.g. localhost) and port number of 9999.

When the code is executed, Spark Streaming only sets up the computation it will perform when it is started, and no real processing is done yet. To start the processing after all the transformations have been setup, we finally call start() method to start the computation and awaitTermination() method to wait for the computation to terminate.

Spark Streaming API

Spark Streaming comes with several API methods that are useful for processing data streams. There are RDD-like operations like map, flatMap, filter, count, reduce, groupByKey, reduceByKey, sortByKey, and join. It also provides additional API to process the streaming data based on window and stateful operations. These include window, countByWindow, reduceByWindow, countByValueAndWindow, reduceByKeyAndWindow and updateStateByKey.

Spark Streaming library is currently supported in Scala, Java, and Python programming languages. Here are the links to Spark Streaming API in each of these languages.

* [Spark Streaming Scala API](https://spark.apache.org/docs/1.3.0/api/scala/index.html#org.apache.spark.sql.package)
* [Java API](https://spark.apache.org/docs/1.3.0/api/java/index.html?org/apache/spark/sql/api/java/package-summary.html)
* [Python API](https://spark.apache.org/docs/1.3.0/api/python/pyspark.sql.html)

Steps in a Spark Streaming program

Before we discuss the sample application, let’s take a look at different steps involved in a typical Spark Streaming program.

* Spark Streaming Context is used for processing the real-time data streams. So, the first step is to initialize the StreamingContext object using two parameters, SparkContext and sliding interval time. Sliding interval sets the update window where we process the data coming in as streams. Once the context is initialized, no new computations can be defined or added to the existing context. Also, only one StreamingContext object can be active at the same time.
* After Spark Streaming context is defined, we specify the input data sources by creating input DStreams. In our sample application, the input data source is a log message generator that uses Apache Kafka distributed database and messaging system. Log generator program creates random log messages to simulate a web server run-time environment where log messages are continuously generated as various web applications serve the user traffic.
* Define the computations using the Sparking Streaming Transformations API like map and reduce to DStreams.
* After streaming computation logic is defined, we can start receiving the data and process it using start method in StreamingContext object created earlier.
* Finally, we wait for the streaming data processing to be stopped using the awaitTerminationmethod of StreamingContext object.

Sample Application

The sample application we discuss in this article is a server log processing and analytics program. It can be used for real-time monitoring of application server logs and performing data analytics on those logs. These log messages are considered [time series data](https://en.wikipedia.org/wiki/Time_series), which is defined as a sequence of data points consisting of successive measurements captured over a specified time interval.

Time series data examples include sensor data, weather information, and click stream data. Time series analysis is about processing the time series data to extract insights that can be used for business decision making. This data can also be used for predictive analytics to predict future values based on historical data.

With a solution like this, we don’t need an hourly or daily batch job to process the server logs. Spark Streaming receives continuously generated data, processes it, and computes log statistics to provide insights into the data.

To follow a standard example on analyzing the server logs, we’ll use Apache Log Analyzer discussed in Data Bricks Spark Streaming [Reference Application](https://www.gitbook.com/book/databricks/databricks-spark-reference-applications/details) as a reference to our sample application. This application already has log message parsing code that we’ll reuse in our application. The reference application is an excellent resource to learn more about Spark framework in general and Spark Streaming in particular. For more details on Databricks Spark Reference Application, checkout their [website](https://databricks.gitbooks.io/databricks-spark-reference-applications/content/).

Use Case

The use case for the sample application is a web server log analysis and statistics generator. In the sample application, we analyze the web server logs to compute the following statistics for further data analysis and create reports and dashboards:

* Response counts by different HTTP response codes
* Response content size
* IP address of the clients to assess where the highest web traffic is coming from
* Top end point URLs to identify which services are accessed more than others

Unlike the previous two articles in this series, we will use Java instead of Scala for creating the Spark program in this article. We’ll also run the program as a stand-alone application instead of running the code from the console window. This is how we would deploy Spark programs in Test and Production environments. Shell console interface (using Scala, Python, or R languages) is for local developer testing only.

Technologies

We will use the following technologies in sample application to demonstrate how Spark Streaming library is used for processing the real time data streams.

Zookeeper

[Zookeeper](https://zookeeper.apache.org/) is a centralized service providing reliable distributed coordination for distributed applications. Kafka, the messaging system we use in the sample application, depends on Zookeeper for configuration details across the cluster.

Apache Kafka

[Apache Kafka](http://kafka.apache.org/documentation.html) is a real time, fault tolerant, scalable messaging system for moving data in real time. It's a good candidate for use cases like capturing user activity on websites, logs, stock ticker data, and instrumentation data.

Kafka works like a distributed database and is based on a partitioned and replicated low latency commit log. When we post a message to Kafka, it's replicated to different servers in the cluster and at the same time it’s also committed to disk.

Apache Kakfa includes client API as well as a data transfer framework called Kafka Connect.

**Kafka Clients:** Kafka includes [Java clients](http://kafka.apache.org/documentation.html#api) (for both message producers and consumers). We will use the Java producer client API in our sample application.

**Kafka Connect:** Kafka also includes [Kafka Connect](http://kafka.apache.org/documentation.html#connect), which is a framework for streaming data between Apache Kafka and external data systems to support the data pipelines in organizations. It includes import and export connectors to move data sets into and out of Kafka. Kafka Connect program can run as a standalone process or as a distributed service and supports REST interface to submit the connectors to Kafka Connect cluster using a REST API.

Spark Streaming

We’ll use Spark Streaming Java API to receive the data streams, calculate the log statistics, and run queries to answer questions like what are the IP addresses where more web requests are coming from, etc.

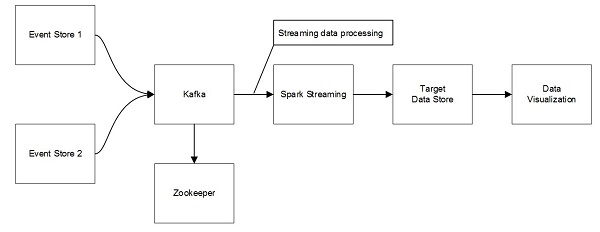
Table 1 below shows the technologies and tools and their versions used in the sample applications.

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **URL** |
| Zookeeper | 3.4.6 | https://zookeeper.apache.org/doc/r3.4.6/ |
| Kafka | 2.10 | http://kafka.apache.org/downloads.html |
| Spark Streaming | 1.4.1 | https://spark.apache.org/releases/spark-release-1-4-1.html |
| JDK | 1.7 | http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html |
| Maven | 3.3.3 | http://archive.apache.org/dist/maven/maven-3/3.3.3/ |

**Table 1.** Spark streaming sample application technologies and tools

Different architecture components of Spark Streaming sample application are illustrated in Figure 3.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig3large.jpg)

**Figure 3. Spark Streaming Sample Application Architecture**

Spark Streaming Application Run-time

To setup the Java project locally, you can download Databricks reference application [code from Github](https://github.com/databricks/reference-apps). Once you get the reference application code, you will need two additional Java classes to run our sample application.

* Log generator (SparkStreamingKafkaLogGenerator.java)
* Log analyzer (SparkStreamingKafkaLogAnalyzer.java)

These files are provided as a zip file ([spark-streaming-kafka-sample-app.zip](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/spark-streaming-kafka-sample-app.zip)) on the article website. If you want to run the sample application on your local machine, use the link to download the zip file, extract Java classes and add them to the Java project created in the previous step.

The sample application can be executed on different operating systems. I ran the application in both Windows and Linux (CentOS VM) environments.

Let’s look at each component in the application architecture and the steps to execute Sparking Streaming program.

Zookeeper Commands:

I used Zookeeper version 3.4.6 in the sample application. To start the server, set two environment variables, JAVA\_HOME and ZOOKEEPER\_HOME to point to JDK and Zookeeper installation directories respectively. Then navigate to Zookeeper home directory and run the following command to start Zookeeper server.

bin\zkServer.cmd

If you are using a Linux environment, the command is:

bin/zkServer.sh start

Kafka Server Commands:

Kafka version 2.10-0.9.0.0 was used in the program, which is based on Scala 2.10 version. Scala version you use with Kakfa is very important because if the correct version is not used, you get run-time errors when executing the spark streaming program. Here are the step to start Kafka server instance:

* Open a new command prompt
* Set JAVA\_HOME and KAFKA\_HOME variables
* Navigate to Kafka home directory
* Run the following command

bin\windows\kafka-server-start.bat config\server.properties

For Linux environment, the command is as follows:

bin/kafka-server-start.sh config/server.properties

Log Generator Commands:

Next step in our sample application is to run the message log generator.

Log generate creates test log messages with different HTTP response codes (like 200, 401, and 404) with different end point URLs.

Before we run the log generator, we need to create a Topic that we can write the messages to.

Similar to the previous step, open a new command prompt, set JAVA\_HOME and KAFKA\_HOMEvariables, and navigate to Kafka home directory. Then run the following command first to view the existing topics available in Kafka server.

bin\windows\kafka-run-class.bat kafka.admin.TopicCommand --zookeeper localhost:2181 --list

or in Linux:

bin/kafka-run-class.sh kafka.admin.TopicCommand --zookeeper localhost:2181 --list

We will create a new topic called "spark-streaming-sample-topic" using the following command:

bin\windows\kafka-run-class.bat kafka.admin.TopicCommand --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --create --topic spark-streaming-sample-topic

or in Linux:

bin/kafka-run-class.sh kafka.admin.TopicCommand --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --create --topic spark-streaming-sample-topic

You can run the list topics command again to see if the new topic has been created correctly.

After the topic has been created, we can run the log generator program. This is done by executing the Java class called SparkStreamingKafkaLogGenerator. Log generator class takes the following four arguments to specify the configuration parameters.

* Group Id: spark-streaming-sample-group
* Topic: spark-streaming-sample-topic
* Number of iterations: 50
* Interval: 1000

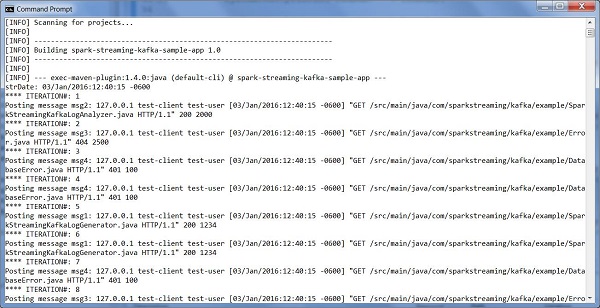
Open a new command prompt to run the log generator. We will set three environment variables (JAVA\_HOME, MAVEN\_HOME, and KAFKA\_HOME) for JDK, Maven, and Kakfa directories respectively. Then navigate to sample project root directory (e.g. c:\dev\projects\spark-streaming-kafka-sample-app) and run the following command.

mvn exec:java -Dexec.mainClass=com.sparkstreaming.kafka.example.SparkStreamingKafkaLogGenerator -Dexec.args="spark-streaming-sample-groupid spark-streaming-sample-topic 50 1000"

Once the log generator program is running, you should see the test log messages created with the debug messages shown on the console. This is only sample code, so the log messages are randomly generated to simulate the continuous flow of data from an event store like a web server.

Figure 4 below shows the screenshot of log message producer and log messages are being generated.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig4large.jpg)

**Figure 4. Spark streaming log generator program output**

Spark Streaming Commands:

This is the consumer of log messages using Spark Streaming API. We use a Java class called SparkStreamingKafkaLogAnalyzer to receive the data streams from Kafka server and process them to create log statistics.

Sparking Streaming processes server log messages and generates cumulative log statistics like web request content size (minimum, maximum, and average), response code counts, IP addresses and the top endpoints.

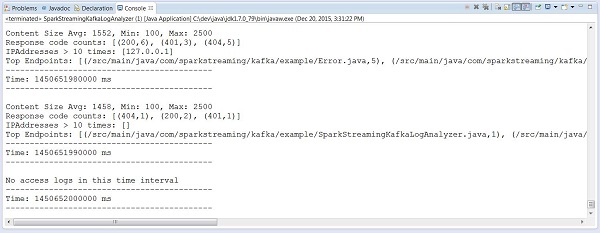
We create the Spark Context using "local[\*]" parameter, which detects the number of cores in the local system and uses them to run the program.

To run the Spark Streaming Java class, you will need the following JAR files in the classpath:

* kafka\_2.10-0.9.0.0.jar
* kafka-clients-0.9.0.0.jar
* metrics-core-2.2.0.jar
* spark-streaming-kafka\_2.10-1.4.0.jar
* zkclient-0.3.jar

I ran the program from Eclipse IDE after adding the above JAR files to the classpath. Log analysis Spark Streaming program output is shown in Figure 5.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig5large.jpg)

**Figure 5. Spark streaming log analytics program output**

Visualization of Spark Streaming Applications

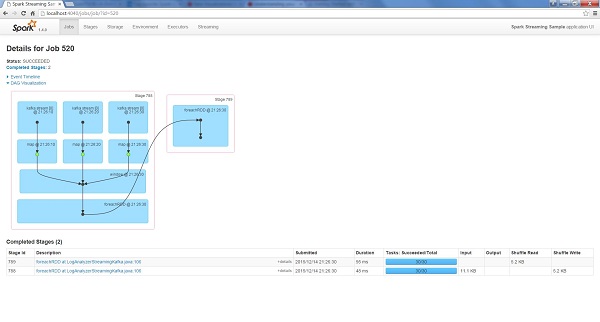
When Spark Streaming program is running, we can check the Spark console to view the details of the Spark jobs.

Open a new web browser window and navigate to URL [http://localhost:4040](http://localhost:4040/) to access the Spark console.

Let’s look at some of the graphs showing the Spark Streaming program statistics.

First visualization is the DAG (Direct Acyclic Graph) of a specific job showing the dependency graph of different operations we ran in the program, like map, window, and foreachRDD. Figure 6 below shows the screenshot of this visualization of Spark Streaming job from our sample program.

**(Click on the image to enlarge it)**

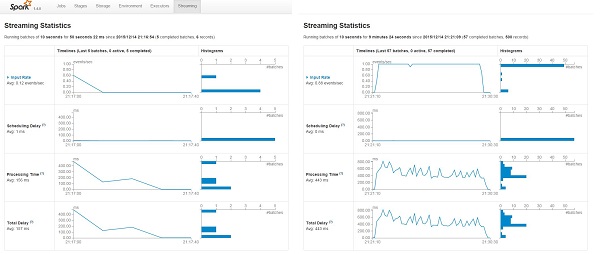
[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig6large.jpg)

**Figure 6. DAG visualization graph of spark streaming job**

Next graph we look at is the streaming statistics which include the input rate showing the number of events per second, processing time in milliseconds.

Figure 7 shows these statistics during the execution of Spark Streaming program when the streaming data is not being generated (left section) and when the data stream is being sent to Kafka and processed by Spark Streaming consumer (right section).

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-streaming/en/resources/fig7large.jpg)

**Figure 7. Spark visualization showing streaming statistics for the sample program**

Conclusions

[Spark Streaming library](http://spark.apache.org/docs/latest/streaming-programming-guide.html), part of Apache Spark eco-system, is used for data processing of real-time streaming data. In this article, we learned about how to use Spark Streaming API to process data generated by server logs and perform analytics on the real-time data streams.

**What's Next**

Machine Learning, Predictive Analytics, and Data Science are recently getting lot of attention for problem solving in different use cases. [Spark MLlib](http://spark.apache.org/mllib/), Spark’s Machine Learning library, provides several built-in methods to use different machine learning algorithms like Collaborative Filtering, Clustering, and Classification.

In the next article, we will explore Spark MLlib and look at couple of use cases to illustrate how we can leverage data science capabilities of Spark, to make it easy to run Machine Learning algorithms.

In the future articles of this series, we'll look at the upcoming frameworks like [BlinkDB](http://blinkdb.org/) and [Tachyon](http://tachyon-project.org/).

Machine learning, predictive analytics, and data science topics are getting a lot of attention in recent years for solving real world problems in different business domains in several organizations. [Spark MLlib](http://spark.apache.org/mllib/), Spark’s Machine Learning library, includes several different machine learning algorithms for Collaborative Filtering, Clustering, Classification and other machine learning tasks.

In the previous articles in “Big Data Processing with Apache Spark” [series](https://www.infoq.com/apache_spark/?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal), we have looked at what Apache Spark framework is ([Part 1](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), how to leverage the SQL interface to access data using Spark SQL library ([Part 2](http://www.infoq.com/articles/apache-spark-sql?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)) and real-time data processing & analytics of streaming data using Spark Streaming ([Part 3](http://www.infoq.com/articles/apache-spark-streaming?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)).

**Related Vendor Content**

[**Modernize your Enterprise Linux Data with SQL Server 2017**](https://www.infoq.com/vendorcontent/show.action?vcr=4561&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-machine-learning%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**The Six Essentials for DevOps Team Excellence**](https://www.infoq.com/vendorcontent/show.action?vcr=4540&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-machine-learning%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Microservices for Java Developers (By O'Reilly) - Download Now**](https://www.infoq.com/vendorcontent/show.action?vcr=4339&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-machine-learning%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**How to Overcome the Microservices Sprawl**](https://www.infoq.com/vendorcontent/show.action?vcr=4503&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-machine-learning%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

**Related Sponsor**

[](https://www.infoq.com/infoq/url.action?i=16688&t=f)

Fastly’s edge cloud platform powers secure, fast and reliable online experiences for the world’s most popular digital businesses. [**See for yourself.**](https://www.infoq.com/infoq/url.action?i=16687&t=f)

In this article, we'll discuss machine learning concepts and how to use Apache Spark MLlib library for running predictive analytics. We will use a sample application to illustrate the powerful API Spark provides in machine learning area.

Spark Machine Learning API includes two packages called spark.mllib and spark.ml.

The [spark.mllib](http://spark.apache.org/docs/latest/mllib-guide.html) package contains the original Spark machine learning API built on Resilient Distributed Datasets ([RDDs](http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds)). It offers machine learning techniques which include correlation, classification and regression, collaborative filtering, clustering, and dimensionality reduction.

On the other hand, [spark.ml](http://spark.apache.org/docs/latest/ml-guide.html) package provides machine learning API built on the [DataFrames](http://spark.apache.org/docs/latest/sql-programming-guide.html" \l "dataframes) which are becoming the core part of Spark SQL library. This package can be used for developing and managing the machine learning pipelines. It also provides Feature Extractors, Transformers, Selectors, and machine learning techniques like classification and regression, and clustering.

We’ll focus on Spark MLlib package in this article and discuss various machine learning techniques this package brings to the table. In the next article, we’ll cover the Spark ML package and look at how to create and manage data pipelines.

Machine Learning & Data Science

Machine Learning is about learning from existing data to make predictions about the future. It's based on creating models from input data sets for data-driven decision making.

Data science is the discipline of extracting the knowledge from large data sets (structured or unstructured) to provide insights to business teams and influence the business strategies and roadmaps. The [role of Data Scientist](http://www.infoq.com/articles/role-of-a-data-scientist-in-2016) is more critical than ever in solving the problems that are not easy to solve using the traditional numerical methods.

There are different types of machine learning models like:

* Supervised learning
* Unsupervised learning
* Semi-supervised Learning
* Reinforcement learning

Let’s briefly look at each of these machine learning models and how they compare with each other.

Supervised learning: This technique is used to predict an outcome by training the program using an existing set of training data (labeled data). Then we use the program to predict the label for a new unlabeled data set.

There are two sub-models under supervised machine learning, Regression and Classification.

Unsupervised learning: This is used to find hidden patterns and correlations within the raw data. No training data used in this model, so this technique is based on unlabeled data.

Algorithms like k-means and Principle Component Analysis (PCA) fall into this category.

**Semi-supervised Learning**: This [technique](https://en.wikipedia.org/wiki/Semi-supervised_learning) uses both supervised and unsupervised learning models for predictive analytics. It uses labeled and unlabeled data sets for training. It typically involves using a small amount of labeled data with a large amount of unlabeled data. It can be used for machine learning methods like classification and regression.

**Reinforcement learning**: The [Reinforcement Learning](https://en.wikipedia.org/wiki/Reinforcement_learning) technique is used to learn how to maximize a numerical reward goal by trying different actions and discovering which actions result in the maximum reward.

Following table shows some examples where these machine learning models can be used to solve the problems.

|  |  |
| --- | --- |
| **ML Model** | **Examples** |
| Supervised learning | Fraud detection |
| Unsupervised learning | Social network applications, language prediction |
| Semi-supervised Learning | Image categorization, Voice recognition |
| Reinforcement learning | Artificial Intelligence (AI) applications |

**Table 1. Machine Learning Models and Real-world Examples**

Machine Learning Algorithms

There are [several algorithms](https://en.wikipedia.org/wiki/List_of_machine_learning_concepts) to help with machine learning solutions. Let’s look at some of the algorithms supported by machine learning frameworks.

**Naive Bayes**: Naive Bayes is a supervised learning algorithm used for classification. It's based on applying Bayes theorem and a set of conditional independence assumptions.

k-means Clustering: k-means algorithm creates k groups from a set of objects so that the members of a group are more similar.

**Support vector machines**: Support vector machines (SVMs) is a supervised learning algorithm used to find the boundary that separates classes by as wide a margin as possible. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. Applications of SVM include bioinformatics, text, and image recognition.

**Decision Trees**: Decision trees are used in many types of machine learning problems including multi-class classification. MLlib supports both basic decision tree algorithm and ensembles of trees. Two ensemble algorithms are available, Gradient-Boosted Trees and Random Forests.

As documented on this [website](http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/), here is a summary of all the machine learning styles, problems, and methods to solve them, shown in Table 2 below.

|  |  |  |
| --- | --- | --- |
| **ML Model** | **Problems** | **Algorithms** |
| Supervised Learning | Classification, Regression, Anomaly Detection | Logistic Regression, Back Propagation Neural Network |
| Unsupervised Learning | Clustering, Dimensionality reduction | k-Means , Apriori algorithm |
| Semi-Supervised Learning | Classification, Regression | Self training, Semi-supervised Support Vector Machines (S3VMs) |

**Table 2. Machine learning styles, problems, and methods**

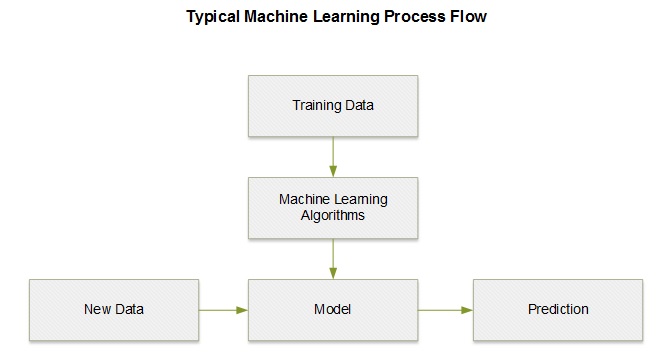
Steps in a Machine Learning Program

When working machine learning projects, other tasks like data preparation, cleansing and analysis are also very important tasks in addition to the actual learning models and algorithms used to solve the business problems.

Following are the steps performed in a typical machine learning program.

* Featurization
* Training
* Model Evaluation

Figure 1 below shows the process flow of a typical machine learning solution.



**Figure 1. Machine Learning Process Flow**

It's important to know that if the raw data isn't cleaned or prepared before running machine learning algorithms, the resulting patterns will not be accurate or useful as well as some anomalies may be missed.

The quality of training data we provide to machine learning programs also plays a critical role in the prediction results. If the training data is not random enough, resulting patterns won’t be accurate. And if the data is too small, the machine learning program may give inaccurate predictions.

Use Cases

The business use cases for machine learning span different domains and scenarios including recommendation engines ([food recommendation engine](https://chimpler.wordpress.com/2014/07/22/building-a-food-recommendation-engine-with-spark-mllib-and-play/)), predictive analytics ([stock price prediction](http://eugenezhulenev.com/blog/2014/11/14/stock-price-prediction-with-big-data-and-machine-learning/) or [predicting flights delay](http://www.datasciencecentral.com/profiles/blogs/predicting-flights-delay-using-supervised-learning)), targeted advertising, fraud detection, image and video recognition, self-driving cars and various other artificial intelligence.

Let’s look at two popular machine learning applications, recommendation engine and fraud detection, in more detail.

Recommendation Engines

Recommendation engines use the attributes of an item or a user or the behavior of a user or their peers, to make the predictions. There are different factors that drive an effective recommendation engine model. Some of these factors are list below:

* Peer based
* Customer behavior
* Corporate deals or offers
* Item clustering
* Market/Store factors

Recommendation engine solutions are implemented by leveraging two algorithms, content-based filtering and collaborative filtering.

**Content-based filtering**: This is based on how similar a particular item is to other items based on usage and ratings. The model uses the content attributes of items (such as categories, tags, descriptions and other data) to generate a matrix of each item to other items and calculates similarity based on the ratings provided. Then the most similar items are listed together with a similarity score. Items with the highest score are most similar.

Movie recommendation is a good example of this model. It recommends that "Users who liked a particular movie liked these other movies as well".

These models don’t take into account the overall behavior of other users, so they don't provide personalized recommendations compared to other models like collaborative filtering.

**Collaborative Filtering**: On the other hand, collaborative filtering model is based on making predictions to find a specific item or user based on similarity with other items or users. The filter applies weights based on the "peer user" preferences. The assumption is users who display similar profile or behavior have similar preferences for items.

An example of this model is the recommendations on ecommerce websites like Amazon. When you search for an item on the website you would see something like "Customers Who Bought This Item Also Bought."

Items with the highest recommendation score are the most relevant to the user in context.

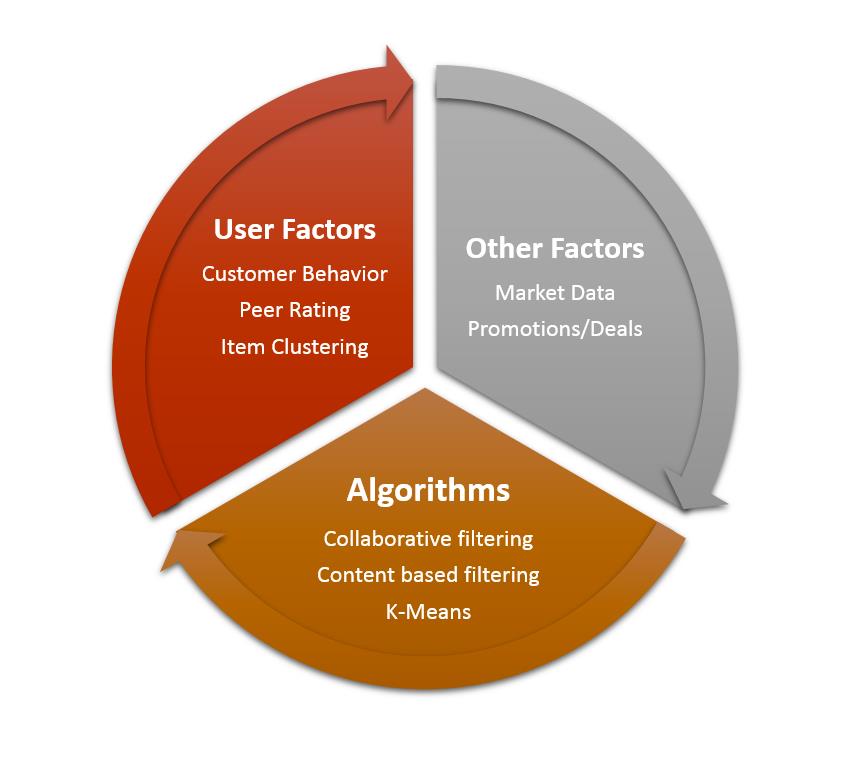
Collaborative filtering based solutions perform better compared to other models. Spark MLlib implements a collaborative filtering algorithm called [Alternating Least Squares](http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html) (ALS). There are two variations of the entries in collaborative filtering, called [explicit and implicit feedback](http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html#explicit-vs-implicit-feedback). Explicit feedback is based on the direct preferences given by the user to the item (like a movie). Explicit feedback is nice, but many times it's skewed because users who strongly like or dislike a product provide reviews on it. We don't get the opinion of many people towards the center of the bell shaped curve of data points.

Implicit feedback examples are user's views, clicks, likes etc. Implicit feedback data is used a lot in the industry for predictive analytics because of the ease to gather this type of data.

There are also [model based methods](http://www.infoq.com/articles/machine-learning-with-spark-book-review) for recommendation engines. These often incorporate methods from collaborative and content-based filtering. Model-based approach gets the best of both worlds, the power and performance of collaborative filtering and the flexibility and adaptability of content-based filtering. Deep learning [techniques](http://www.infoq.com/news/2014/04/Alchemy-Deep-Learning-Landscape) are good examples of this model.

You can also integrate other algorithms like [K-Means](https://en.wikipedia.org/wiki/K-means_clustering) into the recommendation engine solution to get more refined predictions. K-Means algorithm works by partitioning "n" observations into "k" clusters in which each observation belongs to the cluster with the nearest mean. Using K-Means technique, we can find similar items or users based on their attributes.

Figure 2 below shows different components of a recommendation engine, user factors, other factors like market data and different algorithms.



**Figure 2. Components of a Recommendation Engine**

Fraud Detection

Fraud detection is another important use case of using machine learning techniques because it addresses a critical problem in financial industry quickly and accurately. Financial services organizations only have few hundred milliseconds to determine if a particular online transaction is legitimate or a fraud.

Neural network techniques are used for point-of-sale (POS) fraud detection use cases. Organizations like [PayPal](http://www.paypal.com/) use different types of machine learning [algorithms for risk management](http://www.infoworld.com/article/2907877/machine-learning/how-paypal-reduces-fraud-with-machine-learning.html)like linear, neural network, and deep learning.

Spark MLlib library provides [several algorithms](http://spark.apache.org/docs/latest/mllib-classification-regression.html) for solving this use case, including linear SVMs, logistic regression, decision trees, and naive Bayes. In addition, ensemble models (which combine the predictions of a set of models) such as random forests or gradient-boosting trees are also available.

When you are working on a machine learning project in your organization, as recommended in [this article](http://machinelearningmastery.com/how-to-implement-a-machine-learning-algorithm/), you can follow the steps listed below to implement machine learning solutions in your own projects.

* Select the programming language
* Select the appropriate algorithm or algorithms
* Select problem
* Research the algorithms
* Unit test all the functions in the ML solution

Now, let’s look at how Apache Spark framework implements the machine learning algorithms.

Spark MLlib

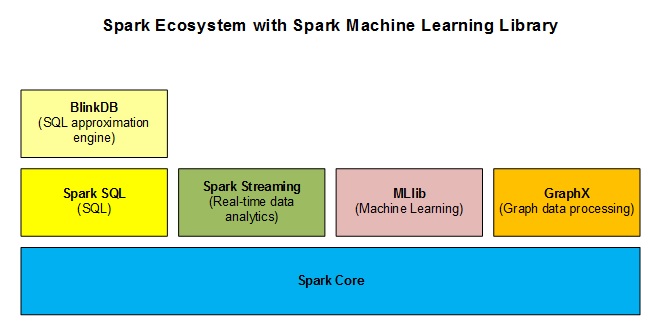
MLlib is Spark’s machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.

Like we learned earlier, there are two different ways of using Spark Machine Learning API, Spark MLlib and Spark ML.

In addition to various algorithms Spark MLlib supports, there are also data processing functions and data analytics [utilities and tools](http://spark.apache.org/mllib/) available in this library.

* Frequent itemset mining via FP-growth and association rules
* Sequential pattern mining via PrefixSpan
* Summary statistics and hypothesis testing
* Feature transformations
* Model evaluation and hyper-parameter tuning

Figure 3 below shows Apache Spark framework with Spark MLlib library.



**Figure 3. Spark Ecosystem with Spark Machine Learning Library**

Spark MLlib API is available in Scala, Java, and Python programming languages. Here are the links to API in each of these languages.

* Spark MLlib [Scala API](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.package)
* [Java API](http://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/mllib/)
* [Python API](http://spark.apache.org/docs/latest/api/python/pyspark.mllib.html)

Sample Application

We’ll develop a sample Machine Learning application using classification technique, specifically collaborative filtering method, to predict the movies to recommend to a user based on other users’ ratings on different movies.

Our recommendation engine solution will use Alternating Least Squares (ALS) machine learning algorithm.

Even though the data sets used in the code example in this article are not very large in size and complexity, Spark MLlib can be used for any type of real world problems that are complex in nature and deal with data containing several dimensions, and very complex predictor functions. Remember machine learning can be used to solve problems that cannot be solved by numerical means alone.

To keep the machine learning application simple so we can focus on Spark MLlib API, we’ll follow the Movie Recommendations example discussed in [Spark Summit](http://www.spark-summit.org/) workshop. This exercise is a very good resource to learn more about Spark MLlib library. For more details on the application, visit their [website](https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html).

Use Case

The use case we want to implement Spark based machine learning solution is a recommendation engine.

Recommendation engines are used to make predictions for unknown user-item associations (e.g. movie ratings) based on known user-item associations. They can make predictions based on based on user’s affinity to other items and other users’ affinity to this specific item. The engine builds a prediction model based on the known data (called Training Data) and then make predictions for unknown user-item associations (called Test Data).

Our program includes the following steps to arrive at the top movie recommendations for the user.

* Load movies data file
* Load the data file with ratings provided by a specific user (you)
* Load the ratings data provided by other users (community)
* Join the user ratings data with community ratings into a single RDD
* Train the model using ALS algorithm using ratings data
* Identify the movies not rated by a particular user (userId = 1)
* Predict the ratings of the items not rated by user
* Get top N recommendations (N=5 in our example)
* Display the recommendation data on the console

If you want to process or run further analysis on output data, you can store the results in a NoSQL database like [Cassandra](http://cassandra.apache.org/) or [MongoDB](http://www.mongodb.com/).

Data Sets

We will use the movie datasets provided by MovieLens [group](https://movielens.org/). There are few different data files we need for the sample application. These datasets are available for download from [GroupLens website](http://grouplens.org/datasets/movielens/). We will use one of the latest datasets (smaller version with 100K ratings). Download the [dataset zip file](http://files.grouplens.org/datasets/movielens/ml-latest.zip) from the website.

Following table shows the different datasets used in the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Dataset** | **File Name** | **Description** | **Data Fields** |
| 1 | Movies Data | movies.csv | Movie details. | movieId,title,genres |
| 2 | User Ratings Data | user-ratings.csv | Ratings by a specific user. | userId,movieId,rating,timestamp |
| 3 | Community Ratings Data | ratings.csv | Ratings by other users. | userId,movieId,rating,timestamp |

User ratings file is for the user in context. You can update the ratings in this file based on your movie preferences. We’ll assign a user id called “User Id 0” to represent these ratings.

When we run the recommendation engine program, we’ll join the specific user ratings data with ratings from the community (other users).

Technologies

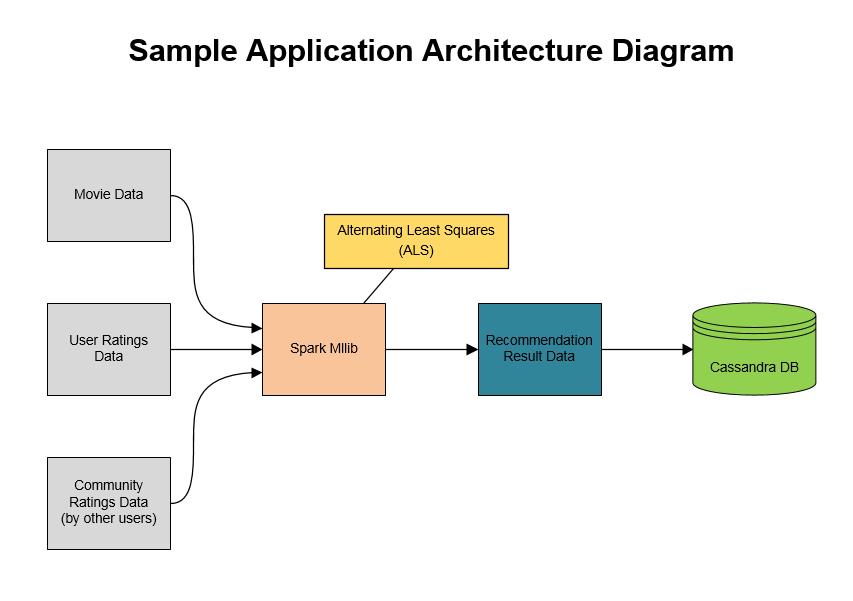
We will use Java to write the Spark MLlib program which can be executed as a stand-alone application. The program uses the following MLlib Java classes (all are located in org.apache.spark.mllib.recommendation package):

* ALS
* MatrixFactorizationModel
* Rating

We will use the following technologies in the sample application to illustrate how Spark MLlib API is used for performing predictive analytics.

* [Apache Spark 1.6.1](http://spark.apache.org/downloads.html)
* Spark Java API
* [JDK version 1.8.0\_65](http://www.oracle.com/technetwork/java/javase/downloads/index.html)
* [Apache Maven 3.3.3](https://maven.apache.org/download.cgi)
* [Eclipse IDE](https://eclipse.org/downloads/)

Architecture components of the sample application are shown in Figure 4 below.



**Figure 4. Spark Machine Learning Sample Application Architecture**

There are several implementations of movie recommendation example available in different languages supported by Spark, like Scala ([Databricks](https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html) and [MapR](https://www.mapr.com/ebooks/spark/08-recommendation-engine-spark.html)), Java ([Spark Examples](https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples/mllib)and [Java based Recommendation Engine](http://blogs.quovantis.com/recommendation-engine-a-machine-learning-approach/)), and [Python](https://www.codementor.io/spark/tutorial/building-a-recommender-with-apache-spark-python-example-app-part1). We’ll use the Java solution in our sample application. Download the [Java program](https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/mllib/JavaRecommendationExample.java) from [Spark examples website](https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples) to run the example on your local machine. Create a new Maven based Java project called spark-mllib-sample-app and copy the Java class into the project. Modify the Java class to pass in the data sets discussed in the previous section.

Make sure you include the required Spark Java libraries in the dependencies section of Maven pom.xml file. To do a clean build and download Spark library JAR files, you can run the following commands.

Set environment parameters for JDK (JAVA\_HOME), Maven (MAVEN\_HOME), and Spark (SPARK\_HOME)

For Windows Operating System:

set JAVA\_HOME=[JDK\_INSTALL\_DIRECTORY]

set PATH=%PATH%;%JAVA\_HOME%\bin

set MAVEN\_HOME=[MAVEN\_INSTALL\_DIRECTORY]

set PATH=%PATH%;%MAVEN\_HOME%\bin

set SPARK\_HOME=[SPARK\_INSTALL\_DIRECTORY]

set PATH=%PATH%;%SPARK\_HOME%\bin

cd c:\dev\projects\spark-mllib-sample-app

mvn clean install

mvn eclipse:clean eclipse:eclipse

If you are using a Linux or Mac OSX system, you can run the following commands:

export JAVA\_HOME=[JDK\_INSTALL\_DIRECTORY]

export PATH=$PATH:$JAVA\_HOME/bin

export MAVEN\_HOME=[MAVEN\_INSTALL\_DIRECTORY]

export PATH=$PATH:$MAVEN\_HOME/bin

export SPARK\_HOME=[SPARK\_INSTALL\_DIRECTORY]

export PATH=$PATH:$SPARK\_HOME/bin

cd /Users/USER\_NAME/spark-mllib-sample-app

mvn clean install

mvn eclipse:clean eclipse:eclipse

If application build is successful, the packaged JAR file will be created in target directory.

We will use spark-submit command to execute the Spark program. Here are the commands for running the program in Windows and Linux/Mac respectively.

Windows:

%SPARK\_HOME%\bin\spark-submit --class "org.apache.spark.examples.mllib.JavaRecommendationExample" --master local[\*] target\spark-mllib-sample-1.0.jar

Linux/Mac:

$SPARK\_HOME/bin/spark-submit --class "org.apache.spark.examples.mllib.JavaRecommendationExample" --master local[\*] target/spark-mllib-sample-1.0.jar

Monitoring of Spark MLlib Application

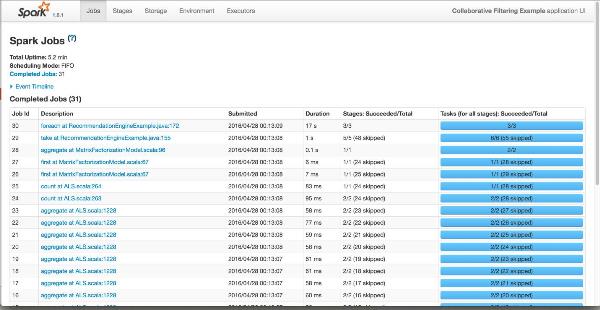
We can monitor the Spark program status on the web console which is available at [URL](http://localhost:4040/).

Let’s look at some of these visualization tools showing the Spark machine learning program statistics.

We can view the details of all jobs in the sample machine learning program. Click on “Jobs” tab on the web console screen to navigate the Spark Jobs web page that shows these job details.

Figure 5 below shows the status of the jobs from the sample program.

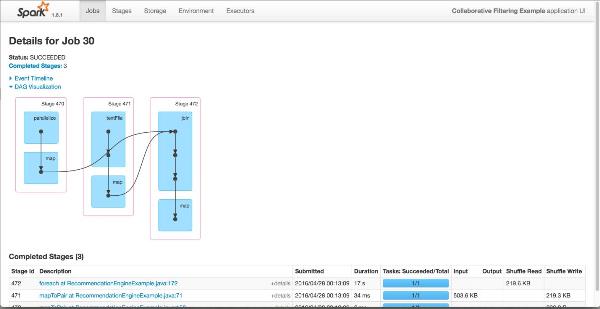
**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-machine-learning/en/resources/fig5.jpg)

**Figure 5. Spark jobs statistics for the machine learning program**

Direct Acyclic Graph (DAG) shows the dependency graph of different RDD operations we ran in the program. Figure 6 below shows the screenshot of this visualization of Spark machine learning job.

**(Click on the image to enlarge it)**

[](https://cdn.infoq.com/statics_s1_20171010-0642/resource/articles/apache-spark-machine-learning/en/resources/fig6.jpg)

**Figure 6. DAG visualization graph of Spark machine learning program**

Conclusions

Spark MLlib is one of the two machine learning libraries from Apache Spark. It's used for implementing predictive analytics solutions for business use cases like recommendation engine and fraud detection system. In this article, we learned about how to use [Spark MLlib](http://spark.apache.org/mllib/) to create a recommendation engine application to predict the movies that a user might like.

Make sure you perform sufficient testing to assess the effectiveness as well as performance of different machine learning techniques to find out the best solution for your requirements and use cases.

What's Next

In the next article, we will focus on the other Machine Learning API from Spark, called [Spark ML](http://spark.apache.org/docs/latest/ml-guide.html), which is the recommended solution for big data projects developed on data pipelines.

In the previous articles in the “Big Data Processing with Apache Spark” [series](https://www.infoq.com/apache_spark/?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal), we looked at Apache Spark framework and its different libraries for big data processing with Spark Introduction ([Part 1](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), Spark SQL library ([Part 2](http://www.infoq.com/articles/apache-spark-sql?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), Spark Streaming ([Part 3](http://www.infoq.com/articles/apache-spark-streaming?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)) and Spark MLlib Machine Learning library ([Part 4](https://www.infoq.com/articles/apache-spark-machine-learning?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)).

**Related Vendor Content**

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

[**Java Performance: Top 10 Problems**](https://www.infoq.com/vendorcontent/show.action?vcr=4502&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-sparkml-data-pipelines%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**4 Things to Consider When Delivering Apps in Hybrid or Multi-Cloud Environments**](https://www.infoq.com/vendorcontent/show.action?vcr=4535&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-sparkml-data-pipelines%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Production-Ready Microservices (By O'Reilly) - Download Now**](https://www.infoq.com/vendorcontent/show.action?vcr=4577&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-sparkml-data-pipelines%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Modernize your Enterprise Linux Data with SQL Server 2017**](https://www.infoq.com/vendorcontent/show.action?vcr=4561&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-sparkml-data-pipelines%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

**Related Sponsor**

[](https://www.infoq.com/infoq/url.action?i=16688&t=f)

Fastly’s edge cloud platform powers secure, fast and reliable online experiences for the world’s most popular digital businesses. [**See for yourself.**](https://www.infoq.com/infoq/url.action?i=16687&t=f)

In this article, we will focus on the other Machine Learning API from Spark, called [Spark ML](http://spark.apache.org/docs/latest/ml-guide.html), which is the recommended solution for big data applications developed using data pipelines.

Spark ML ([spark.ml](http://spark.apache.org/docs/latest/ml-guide.html)) package provides machine learning API built on the [DataFrames](http://spark.apache.org/docs/latest/sql-programming-guide.html" \l "dataframes" \t "_blank) which are becoming the core part of Spark SQL library. This package can be used for developing and managing the machine learning pipelines. It also provides feature extractors, transformers, selectors and supports machine learning techniques like classification, regression, and clustering. All of these are critical for developing machine learning solutions.

We’ll look at how we can use Apache Spark to perform exploratory data analysis (EDA), develop machine learning pipelines, and use the APIs and algorithms available in Spark ML package.

With support for building Machine Learning data pipelines, Apache Spark framework is a great choice for building a unified use case that combines ETL, batch analytics, real­-time stream analysis, machine learning, graph processing and visualizations.

Machine Learning Data Pipelines

Machine learning pipelines are used for the creation, tuning, and inspection of machine learning workflow programs. ML pipelines help us focus more on the big data requirements and machine learning tasks in our projects instead of spending time and effort on the infrastructure and distributed computing areas. They also help us with the exploratory stages of machine learning problems where we need to develop iterations of features and model combinations.

Machine Learning (ML) workflows often involve a sequence of processing and learning stages. Machine learning data pipeline is specified as a sequence of stages where each stage is either a Transformer or an Estimator component. These stages are executed in order, and the input data is transformed as it passes through each stage in the pipeline.

ML development frameworks need to support distributed computation as well as utilities to help with assembling the pipeline components. Other requirements for building data pipelines include fault tolerance, resource management, scalability and maintainability.

The machine learning workflow solutions in real world projects also include utilities like model import/export, cross-validation to choose parameters, and aggregate data from multiple data sources. They provide data utilities like feature extraction, selection and statistics. These frameworks support machine learning pipeline persistence to save and load ML models and pipelines for future use.

The concept of machine learning workflows and the composition of dataflow operators is becoming popular in several different systems. Big data processing frameworks like [scikit-learn](http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html" \t "_blank)and [GraphLab](http://graphlab.com/learn/userguide/index.html" \l "Deployment" \t "_blank) use the concept of pipelines built into the system.

A typical [data value chain](http://conferences.oreilly.com/strata/big-data-conference-ny-2015/public/schedule/detail/43278) process includes the following steps:

* Discover
* Ingest
* Process
* Persist
* Integrate
* Analyze
* Expose

A machine learning data pipeline follows a similar approach. The following table shows the different steps involved in a machine learning pipeline process.

|  |  |  |
| --- | --- | --- |
| **Step #** | **Name** | **Description** |
| ML1 | Data Ingestion | Loading the data from different data sources. |
| ML2 | Data Cleaning | Data is pre-processed to get it ready for the machine learning data analysis. |
| ML3 | Feature Extraction | Also known as [Feature Engineering](https://en.wikipedia.org/wiki/Feature_engineering), this step is about extracting the features from the data sets. |
| ML4 | Model Training | The machine learning model is trained in the next step using the training data sets. |
| ML5 | Model Validation | Next, the machine learning model is evaluated based on different prediction parameters, for its effectiveness. We also tune the model during the validation step. This step is used to pick the best model. |
| ML6 | Model Testing | The next step is to test the mode before it is deployed. |
| ML7 | Model deployment | Final step is to deploy the selected model to execute in production environment. |

Table 1. Machine learning pipeline process steps

These steps are illustrated in Figure 1 below.

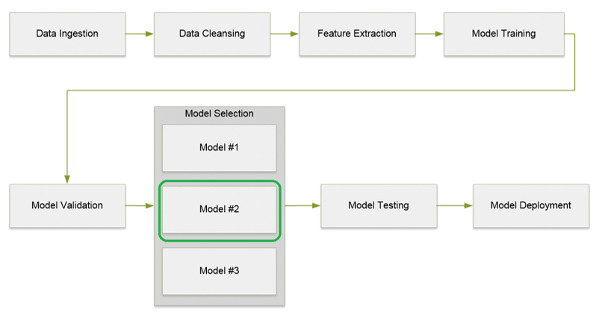


Figure 1. Machine learning data pipeline process flow diagram

Let’s look at each of these steps in more detail.

**Data Ingestion:** The data we collect for a typical machine learning pipeline application can come from multiple data sources and can range from few hundred gigabytes to a terabyte. Also, one of the characteristics of big data applications is ingesting data in different formats.

**Data Cleaning:** Data cleaning is the first and critical step in the overall data analytics pipeline. Also known as data cleansing, data scrubbing, or [data wrangling](https://en.wikipedia.org/wiki/Data_wrangling), it's used to structure the data to facilitate data processing and predictive analytics. Depending on the quality of data coming into the system, [60-70% of the overall time is spent in data cleaning](http://blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/) to bring data to a suitable format so machine learning models can be applied on the data.

Data can have various quality issues like missing data or the data elements with incorrect or irrelevant values. Data cleaning process typically uses various approaches [including custom transformers](https://developer.ibm.com/spark/blog/2016/02/22/predictive-model-for-online-advertising-using-spark-machine-learning-pipelines/) where the data cleansing actions are executed with custom transformers included in the pipeline.

Sparse or coarse-grained data is another challenge in data analytics. This is where lot of corner cases occur so we have to apply data cleaning techniques to make sure the data is of decent quality before feeding it to the data pipeline.

Data cleaning is usually an iterative process as we understand the problem deeper on each successive attempt and update the model iteratively. Data wrangling tools like [Trifacta](https://www.trifacta.com/" \t "_blank), [OpenRefine](http://openrefine.org/" \t "_blank) or [ActiveClean](https://activeclean.github.io/" \t "_blank) are used for data cleaning needs.

**Feature Extraction:** In Feature Extraction (sometimes called Feature Engineering) step, we extract specific features from the raw data using techniques like [Feature Hashing](https://en.wikipedia.org/wiki/Feature_hashing) (Hashing Term Frequency) and [Word2Vec](https://en.wikipedia.org/wiki/Word2vec). The results from this step are usually combined using an Assembler component and are passed to next step in the process.

**Model Training:** Machine learning [model training](http://docs.aws.amazon.com/machine-learning/latest/dg/training-ml-models.html) involves providing an algorithm and some training data that the model can learn from. The learning algorithm finds patterns in the training data and generates an output model.

**Model Validation:** This step involves evaluating and tuning the ML model to assess how effective it is with the predictions. As described in this [article](http://blog.cloudera.com/blog/2016/02/how-to-predict-telco-churn-with-apache-spark-mllib/), for binary classification models the evaluation metric could be the area under the Receiver Operating Characteristic ([ROC](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)) curve. ROC curve illustrates the performance of a binary classifier system. It's created by plotting the true positive rate ([TPR](https://en.wikipedia.org/wiki/Sensitivity_and_specificity)) against the false positive rate ([FPR](https://en.wikipedia.org/wiki/False_positive_rate)) at various threshold settings.

**Model Selection:** Model selection is done by using data to choose parameters for Transformers and Estimators. This is a critical step in the machine learning pipeline process. Classes like ParamGridBuilder and CrossValidator provide APIs for selecting the ML model.

**Model Deployment:** Once we select the right model, we can deploy it and start feeding new data and receive the predictive analytics results. We can also deploy machine learning models as [web services](https://azure.microsoft.com/en-us/documentation/videos/deploying-a-predictive-model-as-a-service-part-i-/).

Spark ML

The machine learning pipeline API was introduced in Apache Spark framework version 1.2. It provides the API for developers to create and execute complex ML workflows. The goal of the Pipeline API is to let users quickly and easily assemble and configure practical distributed machine learning pipelines by standardizing the APIs for different machine learning concepts. The Pipeline API is available in org.apache.spark.ml package.

Spark ML also helps with combining multiple machine learning algorithms into a single pipeline.

Spark machine learning API is divided into two packages called spark.mllib and spark.ml. The spark.mllib package contains the original API built on top of RDDs. On the other hand, the spark.ml package provides higher-level API built on top of DataFrames for constructing ML pipelines.

The MLlib library API which is based on RDDs [is now in maintenance mode](https://www.infoq.com/news/2016/05/Apache-Spark-2.0-Tech-Preview).

Spark ML is an important big data analytics library in the Apache Spark ecosystem as shown in Figure 2 below.

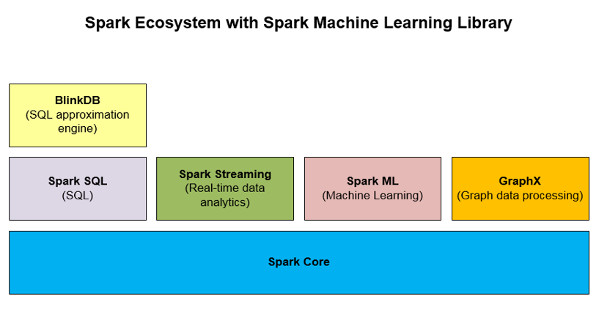


Figure 2. Spark Ecosystem with Spark ML

ML Pipeline Components

Machine learning data pipeline consists of several components to perform the data analytics. The key components of a data pipeline are listed below:

* Datasets
* Pipelines
* Pipeline Stages
* Transformers
* Estimators
* Evaluators
* Params (and ParamMaps)

Let’s take a quick look at where each of these components fit into the overall process.

**Datasets:** DataFrame is used for representing datasets in ML pipeline. It allows storing structured data into named columns. The columns can be used for storing text, feature vectors, true labels, and predictions.

**Pipelines:** ML workflows are modeled as Pipelines, which consist of a sequence of stages. Each stage transforms input data to produce output for succeeding stages. A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

**Pipeline Stages:** We define two kinds of stages: Transformers and Estimators.

**Transformer:** An algorithm which can transform one DataFrame into another DataFrame (example: ML model is a transformer that transforms a DF with features into a DF with predictions).

A Transformer converts a DataFrame into another DataFrame with one or more added features to it. For example in Spark ML package, [OneHotEncoder](https://spark.apache.org/docs/latest/ml-features.html" \l "onehotencoder" \t "_blank) transforms a column with a label index into a column of vectored features. Every Transformer has a method transform() which is called to transform a DataFrame into another.

**Estimator:** Estimator is a machine learning algorithm which learns from the data provided. The input to an estimator is a DataFrame and output is a Transformer. An Estimator is used to train the model. It produces a Transformer. For example, a LogisticRegression estimator produces a LogisticRegressionModel transformer. Another example is K Means as an estimator accepts a training DataFrame and produces a K Means model which is a transformer.

**Parameter:** Machine learning components use a common API for specifying parameters. An example of a parameter is the maximum number of iterations that the model should use.

The components of the data pipeline for text classification use case are shown in the following diagram.

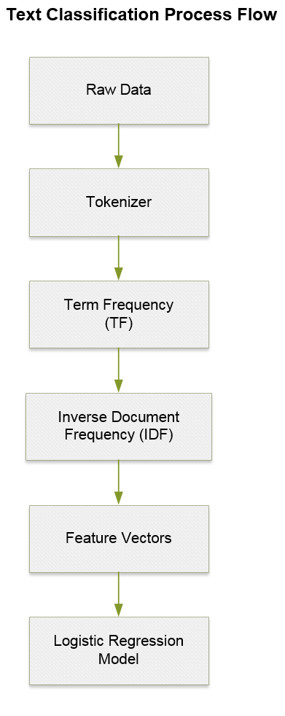


Figure 3. Data Pipelines using Spark ML

Use Cases

One of the use cases for using machine learning pipelines is text categorization. This use case typically includes the following steps:

* clean the text data
* transform data into feature vectors, and
* train the classification model

In text categorization or classification, data preprocessing steps like n-gram extraction and TF-IDF feature weighting are used before the training of a classification model (like [SVM](https://en.wikipedia.org/wiki/Support_vector_machine)).

Another machine learning pipeline use case is the image classification as described in this [article](https://amplab.cs.berkeley.edu/ml-pipelines/).

There are several other machine learning use cases that include fraud detection (uses classification model which is part of supervised learning), user segmentation (clustering model which is part of unsupervised learning).

TF-IDF

Term Frequency - Inverse Document Frequency ([TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf)), is a statistical measure to evaluate how important a word is to a document in a given corpus. It’s an information retrieval algorithm used to rank how important a word is to a collection of documents.

TF: If a word appears frequently in a doc, it’s important. This is calculated as:

TF = (# of times word X appears in a document) / (Total # of

words in the document)

IDF: But if a word appears in many docs (like “the”, “and”, “of”), the word is not meaningful, so lower its score.

Sample Application

Let’s look at a sample application to see how the Spark ML package can be used in a big data processing system. We’ll develop a document classification application to identify spam content in the datasets provided to the application. The dataset examples are Documents, Email messages, or other content received from external systems that can contain spam content.

We’ll use the [Spam Detection](http://conferences.oreilly.com/strata/hadoop-big-data-eu/public/schedule/detail/49475) example discussed in “Building machine-learning apps with Spark” at [Strata Hadoop World Conference](http://conferences.oreilly.com/strata/strata-eu-2016) workshop to build our sample application.

Use Case

This use case includes analyzing different messages sent to our system. Some of these messages contain spam whereas the messages we get without any spam are called ham data. Our goal is to find the messages that contain spam using Spark ML API.

Algorithm

We’ll use Logistic Regression algorithm in our machine learning program. [Logistic Regression](https://en.wikipedia.org/wiki/Logistic_regression) is a regression analysis model and is used to predict the probability of a binary response (yes or no) based on one or more independent variables.

Solution Details

Let’s look at the details of the sample application and the steps we will be running as part of the Spark ML program.

**Data Ingestion:** We’ll load the datasets (text files) for the content that has the spam data as well as the data that doesn’t contain any spam (called ham data).

**Data Cleansing:** In our sample application, we don’t perform any specific data cleaning. We just aggregate all the data into a single DataFrame object.

We create an array object by randomly selecting the data from both training and test datasets. In our example we divide the data sets into 70% of training data and 30% of test data.

We use these two data objects later in the pipeline process to train the model and to make predictions respectively.

Our ML data pipeline includes four steps:

* Tokenizer
* HashingTF
* IDF
* LR

Create a pipeline object and set the above stages in the pipeline. Then we create a Logistic Regression model based on training data in our example.

Now, we can make predictions on the model using the Test data (new datasets).

Figure 4 below shows the architecture diagram of the sample application.

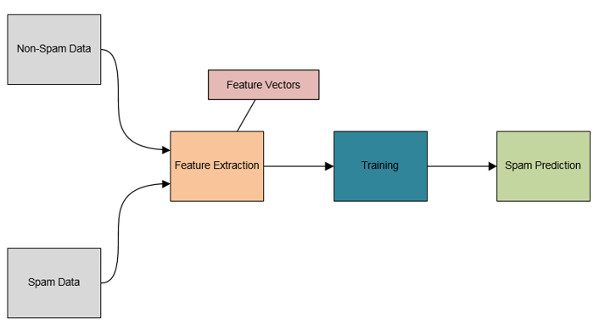


Figure 4. Data classification application architecture diagram

Technologies

We’ll use the following technologies in implementing the machine learning pipeline solution.

|  |  |
| --- | --- |
| **Technology** | **Version** |
| Apache Spark | 2.0.0 |
| JDK | 1.8 |
| Maven | 3.3 |

Table 2. Technologies and tools used in machine learning sample application.

Spark ML Program

The sample machine learning [code](https://github.com/jayantshekhar/strata-2016/blob/master/src/main/scala/com/cloudera/spark/spamdetection/Spam.scala), from the workshop example, is written in the Scala programming language and we can run the program using Spark Shell console.

Spam detection Scala Code snippets:

**Step 1:**Create a custom class to store the details of spam content

case class SpamDocument(file: String, text: String, label:

Double)

**Step 2:** Initialize SQLContext and import the [implicits methods](https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/SQLContext.implicits$.html) to convert Scala objects into DataFrames. Then load the datasets from the specified directory where the files are located, which returns the RDD objects. Create DataFrame objects from the RDD's for both datasets.

val sqlContext = new SQLContext(sc)

import sqlContext.implicits.\_

//

// Load the data files with spam

//

val rddSData = sc.wholeTextFiles("SPAM\_DATA\_FILE\_DIR", 1)

val dfSData = rddSData.map(d => SpamDocument(d.\_1, d.\_2,

1)).toDF()

dfSData.show()

//

// Load the data files with no spam

//

val rddNSData = sc.wholeTextFiles("NO\_SPAM\_DATA\_FILE\_DIR",

1)

val dfNSData = rddNSData.map(d => SpamDocument(d.\_1,

d.\_2, 0)).toDF()

dfNSData.show()

**Step 3:** Now, aggregate the datasets and split the whole data into training and test datasets (with 70% to 30% ratio)

//

// Aggregate both data frames

//

val dfAllData = dfSData.unionAll(dfNSData)

dfAllData.show()

//

// Split the data into 70% training data and 30% test data

//

val Array(trainingData, testData) =

dfAllData.randomSplit(Array(0.7, 0.3))

**Step 4:** We can configure the machine learning data pipeline now which includes creating the components that we discussed earlier in the article, Tokenizer, HashingTF, and IDF. Then create the regression model, in this case, LogisticRegression, using the training data.

//

// Configure the ML data pipeline

//

//

// Create the Tokenizer step

//

val tokenizer = new Tokenizer()

.setInputCol("text")

.setOutputCol("words")

//

// Create the TF and IDF steps

//

val hashingTF = new HashingTF()

.setInputCol(tokenizer.getOutputCol)

.setOutputCol("rawFeatures")

val idf = new

IDF().setInputCol("rawFeatures").setOutputCol("features")

//

// Create the Logistic Regression step

//

val lr = new LogisticRegression()

.setMaxIter(5)

lr.setLabelCol("label")

lr.setFeaturesCol("features")

//

// Create the pipeline

//

val pipeline = new Pipeline()

.setStages(Array(tokenizer, hashingTF, idf, lr))

val lrModel = pipeline.fit(trainingData)

println(lrModel.toString())

**Step 5:** Finally, we can call the transform method in logistic regression model to make the predictions on the test data.

//

// Make predictions.

//

val predictions = lrModel.transform(testData)

//

// Display prediction results

//

predictions.select("file", "text", "label", "features",

"prediction").show(300)

**Conclusions**

Spark Machine Learning library is one of the critical libraries in Apache Spark framework. It's used for implementing data pipelines. In this article, we learned about how to use [Spark ML](https://spark.apache.org/docs/latest/ml-guide.html) package API and how to use it in a text classification use case.

What's Next

Graph data models are about the connected data and relationships between different entities in the data model. Graph data processing techniques are getting a lot of attention lately because they can solve problems like fraud detection and develop recommendation engines.

Spark framework provides a library specialized for graph data analytics. We’ll learn about this library, called [Spark GraphX](https://spark.apache.org/docs/latest/graphx-programming-guide.html), in the next article in this series. We’ll develop a sample application to perform graph data processing and analytics using Spark GraphX.

Key Takeaways

* Learn about graph data processing and analytics
* Apache Spark GraphX library as a solution to perform graph data analytics
* Graph algorithms like PageRank, Connected Components and Triangle Counting
* Spark GraphX components and API
* Sample application using Spark GraphX

*This is the sixth article of the "Big Data Processing with Apache Spark” series. Please see also:*[*Part 1: Introduction*](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)*,*[*Part 2: Spark SQL*](http://www.infoq.com/articles/apache-spark-sql?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)*,*[*Part 3: Spark Streaming*](http://www.infoq.com/articles/apache-spark-streaming?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)*,*[*Part 4: Spark Machine Learning,*](https://www.infoq.com/articles/apache-spark-machine-learning?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)[*Part 5: Spark ML Data Pipelines*](https://www.infoq.com/articles/apache-sparkml-data-pipelines?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)*.*

Big data comes in different shapes and sizes. It can be batch data that needs to be processed offline, processing large set of records and generating the results and insights at a later time. Or the data can be real-time streams which needs to be processed on the fly and create the data insights almost instantaneously.

**Related Vendor Content**

[**AWS Security Fitness Guide: 11 Exercises to Get CloudFit**](https://www.infoq.com/vendorcontent/show.action?vcr=4539&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-graphx%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Site24x7: Powerful, Agile, Cost-Effective IT Management from the Cloud**](https://www.infoq.com/vendorcontent/show.action?vcr=4559&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-graphx%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Top 10 AWS Cloud Security Risks**](https://www.infoq.com/vendorcontent/show.action?vcr=4538&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-graphx%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Microservice Databases: Migrating from Relational Monolith to Distributed Data (By O'Reilly)**](https://www.infoq.com/vendorcontent/show.action?vcr=4348&primaryTopicId=4523&vcrPlace=EMBEDDED&pageType=ARTICLE_PAGE&vcrReferrer=https%3A%2F%2Fwww.infoq.com%2Farticles%2Fapache-spark-graphx%3Futm_source%3Dapachesparkseries%26utm_medium%3Dlink%26utm_campaign%3Dinternal&utm_source=infoq&utm_medium=VCR&utm_campaign=vcr_articles_click&utm_content=embedded)

[**Best practices for using the Vary Header**](https://www.infoq.com/infoq/url.action?i=16805&t=f)

**Related Sponsor**

[](https://www.infoq.com/infoq/url.action?i=16688&t=f)

Fastly’s edge cloud platform powers secure, fast and reliable online experiences for the world’s most popular digital businesses. [**See for yourself.**](https://www.infoq.com/infoq/url.action?i=16687&t=f)

We have seen how Apache Spark can be used for processing batch (Spark Core) as well as real-time data (Spark Streaming).

Sometimes the data we need to deal with is connected in nature. For example, in a social media application, we have entities like Users, Articles, Likes etc. that need to be managed and processed as a single logical unit of data. This type of data is called [Graph data](https://en.wikipedia.org/wiki/Graph_theory), and requires a different type of techniques and approaches to run analytics on this data, compared to traditional data processing.

In the previous articles in this article [series](https://www.infoq.com/apache_spark/?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)titled “Big Data Processing with Apache Spark”, we learned about the Apache Spark framework and its different libraries for big data processing starting with the first article on Spark Introduction ([Part 1](http://www.infoq.com/articles/apache-spark-introduction?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), then we looked at the specific libraries like Spark SQL library ([Part 2](http://www.infoq.com/articles/apache-spark-sql?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), Spark Streaming ([Part 3](http://www.infoq.com/articles/apache-spark-streaming?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)), and both Machine Learning packages: Spark MLlib ([Part 4](https://www.infoq.com/articles/apache-spark-machine-learning?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)) and Spark ML ([Part 5](https://www.infoq.com/articles/apache-sparkml-data-pipelines?utm_source=apachesparkseries&utm_medium=link&utm_campaign=internal)).

In this final installment, we will focus on how to process graph data and Spark’s graph data analytics library called [GraphX](http://spark.apache.org/graphx/).

First, let’s look at what graph data is and why it’s critical to process this type of data in enterprise big data applications.

Graph Data

There are three different topics to cover when we discuss graph data related technologies:

* Graph Databases
* Graph Data Analytics
* Graph Data Visualization

Let’s discuss these topics briefly to learn how they are different from each other and how they complement each other to help us develop a comprehensive graph based big data processing and analytics architecture.

Graph Databases

Unlike traditional data models, data entities as well as the relationships between those entities are the core elements in graph data models. When working on graph data, we are interested in the entities and the connections between the entities.

For example, if we are working on a social network application, we would be interested in the details of a particular user (let’s say John) but we would also want to model, store and retrieve the associations between this user and other users in the network. Examples of these associations are “John is a friend of Mike” or “John read the book authored by Bob.”

It's important to remember that the graph data we use in the real world applications is dynamic in nature and changes over time.

The advantage of graph databases is to uncover patterns that are usually difficult to detect using traditional data models and analytics approaches.

Without Graph databases, implementing a use case like finding common friends is an expensive query as described in this [post](https://gist.github.com/DmitrySandalov/1354852) using data from all the tables with complex joins and query criteria.

Graph database examples include [Neo4j](https://neo4j.com/product/), [DataStax Enterprise Graph](https://www.datastax.com/products/datastax-enterprise-graph), [AllegroGraph](http://allegrograph.com/), [InfiniteGraph](http://www.objectivity.com/products/infinitegraph/), and [OrientDB](http://orientdb.com/orientdb/).

Graph Data Modeling

Graph data modeling effort includes defining the nodes (also known as vertices), relationships (also known as edges), and labels to those nodes and relationships.

Graph databases are modeled based on what Jim Webber calls [Query-driven Modeling](https://www.infoq.com/articles/data-modeling-graph-databases) which means the data model is open to domain experts rather than just database specialists and supports team collaboration for modeling and evolution.

Graph database products typically include a query language ([Cypher](https://neo4j.com/developer/cypher-query-language/) if you are using Neo4j as the database) to manage the graph data stored in the database.

Graph Data Processing

Graph data processing mainly includes graph traversal to find specific nodes in the graph data set that match the specified patterns and then locate the associated nodes and relationships in the data so we can see the patterns of connections between different entities.

The data processing pipeline typically includes the following steps:

* pre-processing of data (which includes loading, transformation, and filtering)
* graph creation
* analysis
* post-processing

A typical graph analytics tool should provide the flexibility to work with both graphs and collections so we can combine data analytics tasks like ETL, exploratory analysis, and iterative graph computation within a single system without having to use several different frameworks and tools.

There are several frameworks that we can use for processing graph data and running predictive analytics on the data. These frameworks include [Spark GraphX](https://spark.apache.org/graphx/), Apache Flink's [Gelly](https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html), and [GraphLab](http://www.select.cs.cmu.edu/code/graphlab/).

In this article, we’ll focus on Spark GraphX for analyzing the graph data.

There are also several different [graph generators](https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/libs/gelly/graph_generators.html) as discussed in Gelly framework documentation like Cycle Graph, Grid Graph, Hypercube Graph, Path Graph and Star Graph.

Graph Data Visualization

Once we start storing connected data in a graph database and run analytics on the graph data, we need tools to visualize the patterns behind the relationships between the data entities.

Graph data visualization tools include [D3.js](https://d3js.org/), [Linkurious](http://linkurio.us/) and [GraphLab Canvas](https://turi.com/products/create/docs/graphlab.canvas.html). Data analytics efforts are not complete without data visualization tools.

Graph Use Cases

There are a variety of use cases where graph databases are better fit to manage the data than other solutions like relational databases or other NoSQL data stores. Some of these use cases include the following:

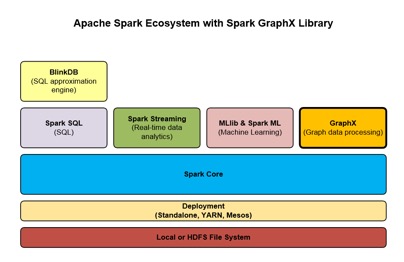
* **Recommendations and Personalization:**Graph analysis can be used to generate [recommendation and personalization](https://en.wikipedia.org/wiki/Recommender_system) models on their customers and to make key decisions from the insights found in the data analysis. This helps the enterprises to effectively influence customers to purchase their product. This analysis also helps with marketing strategy and customer service behavior.
* **Fraud Detection:**Graph data solutions also help to find [fraudulent transactions](http://www.slideshare.net/JenAman/credit-fraud-prevention-with-spark-and-graph-analysis) in a payment processing application based on the connected data that include the entities like users, products, transactions, and events. This [article](https://www.phdata.io/getting-started-with-apache-spark-graphx-part-1/) describes a test application about how to use Spark GraphX for fraud detection using PageRank algorithm on metadata about phone communication.
* **Topic Modeling:**This includes techniques to cluster documents and extract topical representations from the data in those documents.
* **Community Detection:**Alibaba website uses graph data analytics techniques like [community detection](https://spark-summit.org/2015/events/hybrid-community-detection-for-web-scale-e-commerce-using-spark-streaming-and-graphx/) to solve ecommerce problems.
* **Flight Performance:**Other use cases include on-time flight performance as discussed in this [article](https://databricks.com/blog/2016/03/16/on-time-flight-performance-with-graphframes-for-apache-spark.html), to analyze flight performance data organized in graph structures and find out statistics like airport ranking and shortest paths between cities.
* **Shortest Distance:**[Shortest distances and paths](https://blog.insightdatascience.com/computing-shortest-distances-incrementally-with-spark-1a280064a0b9#.106in1z43) are also useful in social network applications. They can be used for measuring the relevance of a particular user in the network. Users with smaller shortest distances are more relevant than users farther away.

Spark GraphX

[GraphX](http://spark.apache.org/graphx/) is Apache Spark's API for graphs and graph-parallel computation. It extends the Spark RDD by introducing a new Graph abstraction: a [directed multigraph](https://en.wikipedia.org/wiki/Multigraph#Directed_multigraph_.28edges_without_own_identity.29) with properties attached to each vertex and edge.

GraphX library provides graph operators like subgraph, joinVertices, and aggregateMessages to transform the graph data. It provides several ways of building a graph from a collection of vertices and edges in an RDD or on disk. GraphX also includes a number of graph algorithms and builders to perform graph analytics tasks. We’ll discuss graph algorithms later in this article.

Figure 1 below shows Apache Spark ecosystem and where GraphX fits in with other libraries in the framework.



**Figure 1. Spark Ecosystem and GraphX library**

GraphX makes it easier to run analytics on graph data with the built-in operators and algorithms. It also allows us to cache and uncache the graph data to avoid recomputation when we need to call a graph multiple times.

Some of the graph operators available in GraphX are listed in Table 1 below.

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Operators** | **Description** |
| Basic Operators | * numEdges * numVertices * inDegrees * outDegrees * degrees |  |
| Property Operators | * mapVertices * mapEdges * mapTriplets |  |
| Structural Operators | * reverse * subgraph * mask * groupEdges |  |
| Join Operators | * joinVertices * outerJoinVertices |  |

**Table 1: Spark GraphX’s graph operators**

We’ll look at more details of these operators in the Sample Application section when we run GraphX algorithms on different social network data sets.

**GraphFrames**

[GraphFrames](https://databricks.com/blog/2016/03/03/introducing-graphframes.html), a new addition to Spark graph data processing toolset, integrates the features like pattern matching and graph algorithms with Spark SQL. Vertices and edges are represented as DataFrames instead of RDD objects.

GraphFrames simplify the graph data analytics pipeline and optimize the queries across both graph and relational data. It provides some advantages as shown below compared to the RDD based graph data processing:

* Support for Python and Java in addition to Scala APIs. Now we can use GraphX algorithms in all three languages.
* Advanced query capability using the Spark SQL and DataFrames API. Graph-aware query planner uses materialized views to improve the query performance.
* We can also save and load graphs using formats like Parquet, JSON, and CSV.

GraphFrames is available as an add-on component to GraphX from [spark-apache.org website](http://spark-packages.org/package/graphframes/graphframes). This [article](http://blog.cloudera.com/blog/2016/10/how-to-do-scalable-graph-analytics-with-apache-spark/) shows how to use GraphFrames to calculate the PageRank for each node in the graph data set.

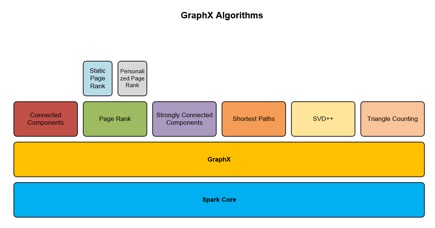
Graph Analytics Algorithms

Graph algorithms help with executing the analytics on graph data sets without having to write our own implementations of those algorithms. Below is a list of various algorithms that help with finding patterns in the graphs.

* PageRank
* Connected components
* Label propagation
* SVD++
* Strongly connected components
* Triangle count
* Single-Source-Shortest-Paths
* Community Detection

Spark GraphX comes with a set of pre-built graph algorithms to help with graph data processing and analytics tasks. These algorithms are available in the org.apache.spark.graphx.lib package. It’s as simple as calling these algorithms as methods in Graph class.

Figure 2 below shows how the different graph algorithms are built on top of the base GraphX API.



**Figure 2. Graph algorithms in Spark GraphX library**

In this article, we’ll look into more details of the PageRank, Connected Components, and Triangle Count algorithms.

PageRank

PageRank algorithm is used to determine the relative importance of an object inside a graph data set. It measures the importance of each node in a graph, assuming an edge from another node to this node represents an endorsement.

Google's search engine is a classic example of PageRank. Google uses PageRank as one of the measures to determine the importance of a web page based on the how many other web pages reference it.

Another example is social network website like Twitter. If a Twitter user is followed by lot of other users, then that user has a higher influence in the network. This metric can be used for ad selection/placement to the users that follow the first user (100,000 users follow a chef=> probably food lovers)

GraphX provides two implementations of PageRank: Static and Dynamic.

**Static PageRank:** This algorithm runs for a fixed number of iterations to generate PageRank values for a given set of nodes in a graph data set.

**Dynamic PageRank:** On the other hand, Dynamic PageRank algorithm runs until PageRank values converge based on a pre-defined tolerance value.

Connected Components

A Connected Component in a graph is a connected subgraph where two vertices are connected to each other by an edge and there are no additional vertices in the main graph. This means the two nodes belong to the same connected component when there is a relationship between them. The lowest numbered vertex number of ID in the subgraph is used to label the connected components in a graph. Connected components can be used to create clusters in the graph for example in a social network.

There are two ways of traversing the graph for computing connected components:

* [Breadth-first Search](https://en.wikipedia.org/wiki/Breadth-first_search) (BFS)
* [Depth-first Search](https://en.wikipedia.org/wiki/Depth-first_search) (DFS)

There is another algorithm called [Strongly Connected Components](http://theory.stanford.edu/~tim/w11/l/scc.pdf) (SCC) in graph data processing. If all nodes in a graph are reachable from every single node, then the graph is considered to be strongly connected.

Triangle Counting

Triangle counting is a community detection graph algorithm which is used to determine the number of triangles passing through each vertex in the graph data set. A vertex is part of a triangle when it has two adjacent vertices with an edge between. The triangle is a three-node subgraph, where every two nodes are connected. This algorithm returns a Graph object and we extract vertices from this triangle counting graph.

Triangle counting is used heavily in social network analysis. It provides a measure of clustering in the graph data which is useful for finding communities and measuring the cohesiveness of local communities in social network websites like LinkedIn or Facebook. [Clustering Coefficient](https://arxiv.org/pdf/1301.5887v3.pdf), an important metric in a social network, shows how much community around one node is tightly connected.

Other use cases where Triangle Counting algorithm is used are spam detection and link recommendations.

Triangle counting is a message heavy and computationally expensive algorithm compared to other graph algorithms. So, make sure you run the Spark program on a decent computer when you test Triangle Count algorithm. Note that PageRank is a measure of relevancy whereas Triangle Count is a measure of clustering.

Sample Application

We have seen so far in this article what graph data is and why graph analytics is an important part of data processing projects in different organizations. Let’s now look at a sample application that uses some of the graph algorithms.

We’ll use data sets from different social network websites like Facebook, LiveJournal, and YouTube. All these applications contain the connected data and are excellent resources for graph data analytics programs.

The examples we use in this article are based on the GraphX samples discussed in this [article](https://github.com/keiraqz/dato-vs-graphx) on comparison of graph processing tools.

Use Case

The main goal of the use cases in our sample application is to determine graph data statistics such as:

* How popular different users in the social network are (PageRank)
* Clusters of users based on how the users in the network are connected (Connected Components)
* Community detection and cohesiveness of the communities of users in the social network (Triangle Counting)

Datasets

In our code examples on Spark GraphX, we will use few different data sets for running Spark GraphX programs. These datasets are available from SNAP (Stanford Network Analysis Project (SNAP) [website](https://snap.stanford.edu/) hosted by Stanford University. If you want to download these datasets, copy them to data folder in the sample application main directory.

**Algorithm**

We’ll use the following three algorithms in our sample application:

* PageRank on YouTube
* Connected Components on LiveJournal
* Triangle Counting on Facebook

The following table shows the use cases, data sets, and algorithms used in the graph data processing programs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case** | **Dataset Source** | **Link** | **File Name** | **Rename File** |
| PageRank | YouTube | https://snap.stanford.edu/data/com-Youtube.html | com-youtube.ungraph.txt | page-rank-yt-data.txt |
| Connected Components | LiveJournal | <https://snap.stanford.edu/data/com-LiveJournal.html> | com-lj.ungraph.txt | connected-components-lj-data.txt |
| Triangle Count | Facebook | <https://snap.stanford.edu/data/egonets-Facebook.html> | facebook\_combined.txt | triangle-count-fb-data.txt |

**Table 2: Data sets and algorithms used in Spark GraphX sample application**

Once you rename the files, copy them to a subdirectory called “data” in the project’s main directory.

Technologies

We’ll use the following technologies in graph analytics sample code:

|  |  |
| --- | --- |
| **Technology** | **Version** |
| Apache Spark | 2.1.0 |
| Scala | 2.11 |
| JDK | 1.8 |
| Maven | 3.3 |

**Table 3. Technologies and tools used in sample application**.

Example Code

We'll write Spark GraphX code using [Scala](https://www.scala-lang.org/) programming language. We'll use [Spark Shell](http://spark.apache.org/docs/latest/quick-start.html)command line tool to run these programs. This is the fastest way to verify the results of the program. No additional code compilation and build steps are needed.

Before we look at the specific code for each of the use cases, these programs will be available as a zip file along with this article that you can download and try out in your own development environment.

Let’s look at the details of each of the sample GraphX programs.

First, we will run PageRank on [YouTube online social network data](https://snap.stanford.edu/data/com-Youtube.html). This dataset includes the ground-truth communities which are basically user defined groups that other users can join in.

**PageRank:**

import org.apache.spark.\_

import org.apache.spark.graphx.\_

import org.apache.spark.rdd.RDD

import java.util.Calendar

// Load the edges first

val graph = GraphLoader.edgeListFile(sc, "data/page-rank-yt-data.txt")

// Compute the graph details like edges, vertices etc.

val vertexCount = graph.numVertices

val vertices = graph.vertices

vertices.count()

val edgeCount = graph.numEdges

val edges = graph.edges

edges.count()

//

// Let's look at some of the Spark GraphX API like triplets, indegrees, and outdegrees.

//

val triplets = graph.triplets

triplets.count()

triplets.take(5)

val inDegrees = graph.inDegrees

inDegrees.collect()

val outDegrees = graph.outDegrees

outDegrees.collect()

val degrees = graph.degrees

degrees.collect()

// Number of iterations as the argument

val staticPageRank = graph.staticPageRank(10)

staticPageRank.vertices.collect()

Calendar.getInstance().getTime()

val pageRank = graph.pageRank(0.001).vertices

Calendar.getInstance().getTime()

// Print top 5 items from the result

println(pageRank.top(5).mkString("\n"))

The variable “sc” in the above code is the SparkContext which is already available when you run programs from Spark Shell.

Let’s now look at the code for how to run Connected Components on [LiveJournal’s social network data](https://snap.stanford.edu/data/soc-LiveJournal1.html). This dataset includes the users who are registered on the website and maintain individual and group blog posts. The website also allows users to identify other users who are their friends.

**Connected Components:**

import org.apache.spark.\_

import org.apache.spark.graphx.\_

import org.apache.spark.rdd.RDD

import java.util.Calendar

// Connected Components

val graph = GraphLoader.edgeListFile(sc, "data/connected-components-lj-data.txt")

Calendar.getInstance().getTime()

val cc = graph.connectedComponents()

Calendar.getInstance().getTime()

cc.vertices.collect()

// Print top 5 items from the result

println(cc.vertices.take(5).mkString("\n"))

val scc = graph.stronglyConnectedComponents()

scc.vertices.collect()

Finally, here is the Spark program, again in Scala, for calculating Triangle Counting on [Facebook's social circles](https://snap.stanford.edu/data/egonets-Facebook.html) data. The dataset includes the lists of friends from Facebook with user profiles, circles, and ego networks.

**Triangle Counting:**

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.graphx.\_

import org.apache.spark.rdd.RDD

val graph = GraphLoader.edgeListFile(sc,"data/triangle-count-fb-data.txt")

println("Number of vertices : " + graph.vertices.count())

println("Number of edges : " + graph.edges.count())

graph.vertices.foreach(v => println(v))

val tc = graph.triangleCount()

tc.vertices.collect

println("tc: " + tc.vertices.take(5).mkString("\n"));

// println("Triangle counts: " + graph.connectedComponents.triangleCount().vertices.collect().mkString("\n"));

println("Triangle counts: " + graph.connectedComponents.triangleCount().vertices.top(5).mkString("\n"));

val sum = tc.vertices.map(a => a.\_2).reduce((a, b) => a + b)

Conclusions

With the increasing growth of connected data in commercial organizations, government agencies, and social media networking companies, graph data processing and analytics are only going to become more critical in predictive analytics and recommendation engine solutions to gain insights and provide service for employees, customers and users.

As we learned in this article, [Spark GraphX](http://spark.apache.org/graphx/) is a very good choice for graph data processing requirements. It provides a unified data processing algorithm and solution toolset for delivering valuable insights and prediction models on the connected data generated by various business processes in organizations.

What’s Next

As we have seen in the articles published in this series, [Apache Spark](http://spark.apache.org/) framework provides the necessary libraries, utilities and tools for unified big data processing application architectures. Whether the data needs to be processed in real time or as a batch, or if the dataset has connections and relationships, Spark makes it easier to work with different types of data. We no longer need to depend on several different frameworks to process and analyze different types of data created and managed in organizations.

If you are looking for a big data solution for applications in your organization, or if you are interested in transitioning to big data and data science areas, Apache Spark is an excellent choice.

## OVERVIEW

Spark adds in-Memory Compute for ETL, Machine Learning and Data Science Workloads to Hadoop

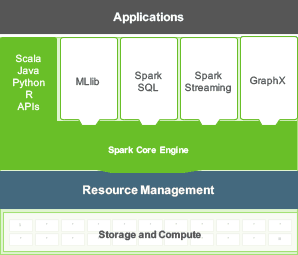
### WHAT APACHE SPARK DOES

Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets. With Spark running on Apache Hadoop YARN, developers everywhere can now create applications to exploit Spark’s power, derive insights, and enrich their data science workloads within a single, shared dataset in Hadoop.

The Hadoop YARN-based architecture provides the foundation that enables Spark and other applications to share a common cluster and dataset while ensuring consistent levels of service and response. Spark is now one of many data access engines that work with YARN in HDP.

Arun Murthy : [Hadoop & Spark : Perfect Together](http://www.slideshare.net/SparkSummit/spark-and-hadoop-perfect-togeher-by-arun-murthy) : Spark Summit 2015

Apache Spark consists of Spark Core and a set of libraries. The core is the distributed execution engine and the Java, Scala, and Python APIs offer a platform for distributed ETL application development.



Additional libraries, built atop the core, allow diverse workloads for streaming, SQL, and machine learning.

Spark is designed for data science and its abstraction makes data science easier.   Data scientists commonly use machine learning – a set of techniques and algorithms that can learn from data. These algorithms are often iterative, and Spark’s ability to cache the dataset in memory greatly speeds up such iterative data processing, making Spark an ideal processing engine for implementing such algorithms.

Spark also includes MLlib, a library that provides a growing set of machine algorithms for common data science techniques: Classification, Regression, Collaborative Filtering, Clustering and Dimensionality Reduction.

Spark’s ML Pipeline API is a high level abstraction to model an entire data science workflow.   The ML pipeline package in Spark models a typical machine learning workflow and provides abstractions like Transformer, Estimator, Pipeline & Parameters.  This is an abstraction layer that makes data scientists more productive.

### SPARK USE CASES

As [Apache Spark](https://es.hortonworks.com/apache/spark)’s momentum continues to grow we are seeing customers across all industries get real value from using it with the Hortonworks Data Platform (HDP).  Customers are using Spark to improve their businesses by detecting patterns and providing actionable insight which is driving organizational change and also starting to change some facets of life.  The following table provides a few examples, from insurance to internet companies, of how Spark is being used:

|  |  |
| --- | --- |
| Insurance | Optimize their claims reimbursements process by using Spark’s machine learning capabilities to process and analyze all claims. |
| Healthcare | Build a Patient Care System using Spark Core, Streaming and SQL. |
| Al por menor | Use Spark to analyze point-of-sale data and coupon usage. |
| Internet | Use Spark’s ML capability to identify fake profiles and enhance products matches that they show their customers. |
| Banking | Use a machine learning model to predict the profile of retail banking customers for certain financial products. |
| Government | Analyze spending across geography, time and category. |
| Scientific Research | Analyze earthquake events by time, depth, geography to predict future events. |
| Investment Banking | Analyze intra-day stock prices to predict future price movements. |
| Geospatial Analysis | Analyze Uber trips by time and geography to predict future demand and pricing. |
| Twitter Sentiment Analysis | Analyze large volumes of Tweets to determine positive, negative or neutral sentiment for specific organizations and products. |
| Airlines | Build a model for predicting airline travel delays. |
| Devices | Predict likelihood of a building exceeding threshold temperatures. |

Many customers are using [Cloudbreak](https://es.hortonworks.com/apache/cloudbreak) and [Ambari](https://es.hortonworks.com/apache/ambari/) to spin up clusters in the cloud for ad-hoc, self-service data science.