Big Data Storage and Processing

MSc in Data Analytics

CCT College Dublin

Apache Spark

Week 4 - 5

Data Storage & Processing
Data Analysis
Data Warehouse
Big Data Tools
cassandra
No sequel Database
Spark
Realtime Processing
Messaging System
HIVE

Lecturer: Dr. Muhammad Iqbal*

Email: miqbal@cct.ie

Agenda

- Introduction to Apache Spark
- Apache Spark: Components
- Apache Spark Components: Spark SQL, Spark Streaming, MLib
- RDD Basics and Spark File Based Input Methods
- Comparison of Spark and Mapreduce
- Spark Programming Paradigm
- Spark and the Hadoop Environment
- Apache Spark Architecture
- Spark Programming Model
- Dataframes and Spark SQL
- Spark ML

Introduction to Apache Spark

- Jean-Georges Perrin (Author of a book Spark in Action) mentioned his experience in his book.

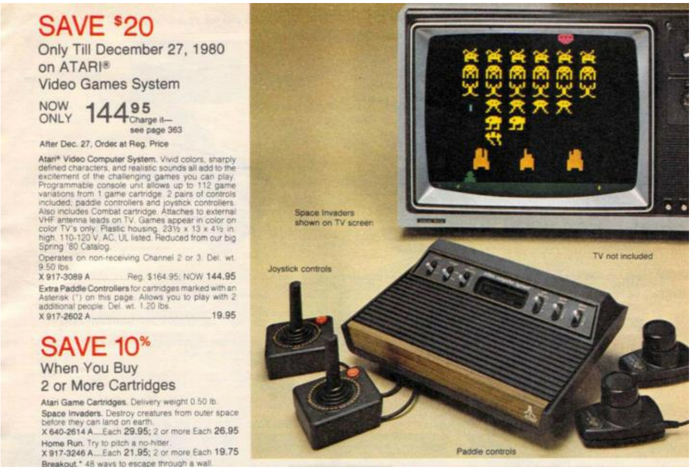
- During his childhood in the 1980s, when he first learned how to program through Basic and my Atari, he could not grasp why automated traffic control, traffic light enforcement, and parking meter enforcement could not be automated.

- From the age of 12, he was not clear about the volume of data. He created Monopoly and it fits in 64 KB of memory; however, the amount of data generated by this game is enormous if it is played by a large number of players.

- All those use cases are still accessible online 35 years later. Data has increased at a faster rate than hardware that supports it. A cluster of small computers will cost less than one large one.

- Networks are many times faster, and modern data centers offer speeds of up to 100 gigabits per second (Gbps), nearly 2,000 times faster than your home Wi-Fi from five years ago. These are some of the factors that drove people to ask this question:

- **How can we use distributed memory computing to analyze large quantities of data?**



SAVE $20

Only Till December 27, 1980
on ATARI®
Video Games System

NOW ONLY 144.95 Charge it— see page 363

After Dec. 27, Order at Reg. Price

Atari* Video Computer System. Vivid colors, sharply defined characters, and realistic sounds all add to the excitement of the challenging games you can play. Programmable console unit allows up to 112 game variations from 1 game cartridge. 2 pairs of controls included; paddle controllers and joystick controllers. Also includes Combat cartridge. Attaches to external VHF antenna leads on TV. Games appear in color on color TV's only. Plastic housing. 23½ x 13 x 4½ in. high. 110-120 V, AC, UL listed. Reduced from our big Spring '80 Catalog.

Operates on non-receiving Channel 2 or 3. Del. wt. 9.50 lbs.
X 917-3089 A .... Reg. $164.95; NOW **144.95**

Extra Paddle Controllers for cartridges marked with an Asterisk (*) on this page. Allows you to play with 2 additional people. Del. wt. 1.20 lbs.
X 917-2602 A 19.95

SAVE 10%

When You Buy
2 or More Cartridges

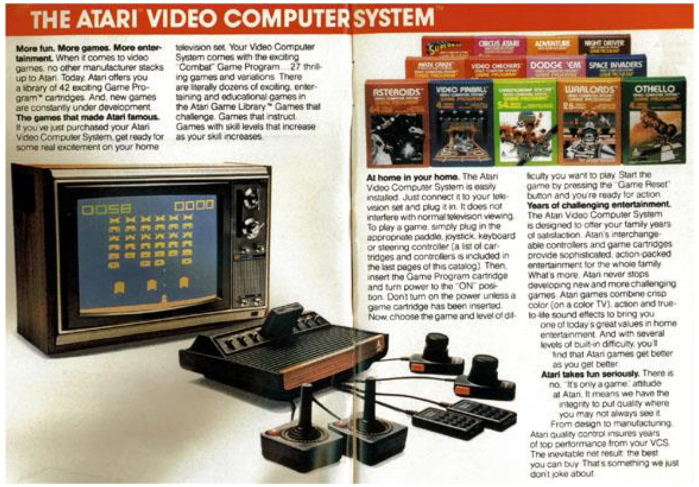
Atari Game Cartridges. Delivery weight 0.50 lb.

Space Invaders. Destroy creatures from outer space before they can land on earth.
X 640-2614 A....Each **29.95**; 2 or more Each **26.95**

Home Run. Try to pitch a no-hitter.
X 917-3246 A....Each **21.95**; 2 or more Each **19.75**

Breakout.* 48 ways to escape through a wall.

Space Invaders shown on TV screen
Joystick controls
TV not included
Paddle controls



THE ATARI VIDEO COMPUTER SYSTEM™

More fun. More games. More entertainment. When it comes to video games, no other manufacturer stacks up to Atari. Today, Atari offers you a library of 42 exciting Game Program™ cartridges. And, new games are constantly under development.

The games that made Atari famous. If you've just purchased your Atari Video Computer System, get ready for some real excitement on your home television set. Your Video Computer System comes with the exciting "Combat" Game Program... 27 thrilling games and variations. There are literally dozens of exciting, entertaining and educational games in the Atari Game Library.™ Games that challenge. Games that instruct. Games with skill levels that increase as your skill increases.

At home in your home. The Atari Video Computer System is easily installed. Just connect it to your television set and plug it in. It does not interfere with normal television viewing. To play a game, simply plug in the appropriate paddle, joystick, keyboard or steering controller (a list of cartridges and controllers is included in the last pages of this catalog). Then, insert the Game Program cartridge and turn power to the "ON" position. Don't turn on the power unless a game cartridge has been inserted. Now, choose the game and level of difficulty you want to play. Start the game by pressing the "Game Reset" button and you're ready for action.

Years of challenging entertainment. The Atari Video Computer System is designed to offer your family years of satisfaction. Atari's interchangeable controllers and game cartridges provide sophisticated, action-packed entertainment for the whole family. What's more, Atari never stops developing new and more challenging games. Atari games combine crisp color (on a color TV), action and true-to-life sound effects to bring you one of today's great values in home entertainment. And with several levels of built-in difficulty, you'll find that Atari games get better as you get better.

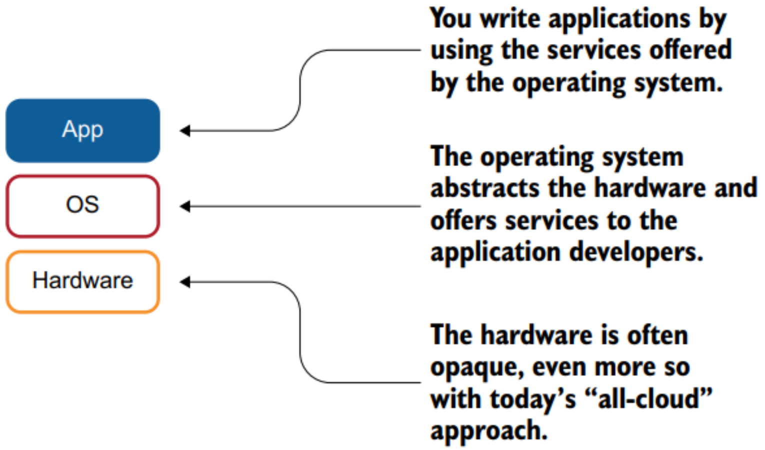
Atari takes fun seriously. There is no, "It's only a game," attitude at Atari. It means we have the integrity to put quality where you may not always see it. From design to manufacturing, Atari quality control insures years of top performance from your VCS. The inevitable net result: the best you can buy. That's something we just don't joke about.

Introduction to Apache Spark

- **Apache Spark** is more than a software stack for data scientists. When you build applications, you build them on top of an operating system, as illustrated in the figure.

- The operating system provides services to make your application development easier; in other words, you are not building a filesystem or network driver for each application you develop.

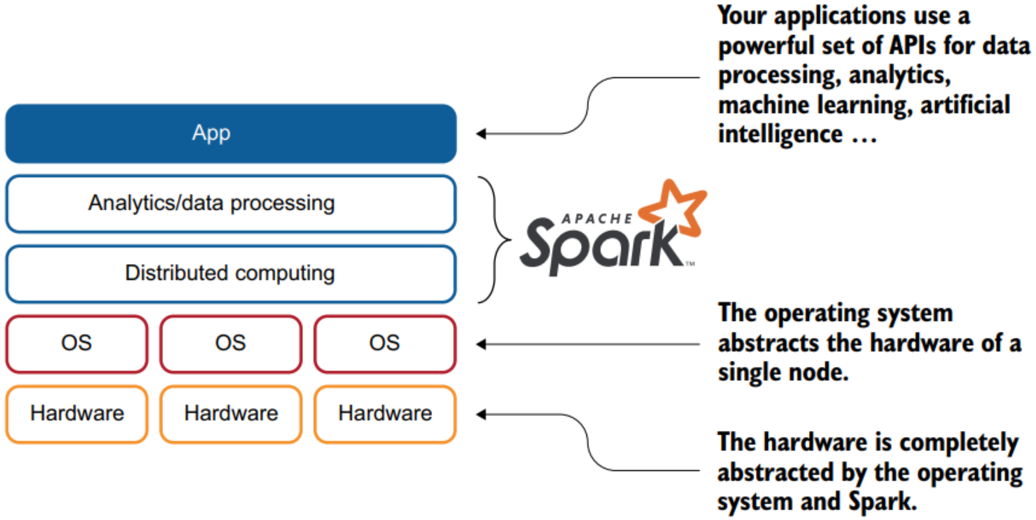
- **Apache Spark** simplifies the development of analytics-oriented applications by offering services to applications as an operating system does.



You write applications by using the services offered by the operating system.

The operating system abstracts the hardware and offers services to the application developers.

The hardware is often opaque, even more so with today's "all-cloud" approach.



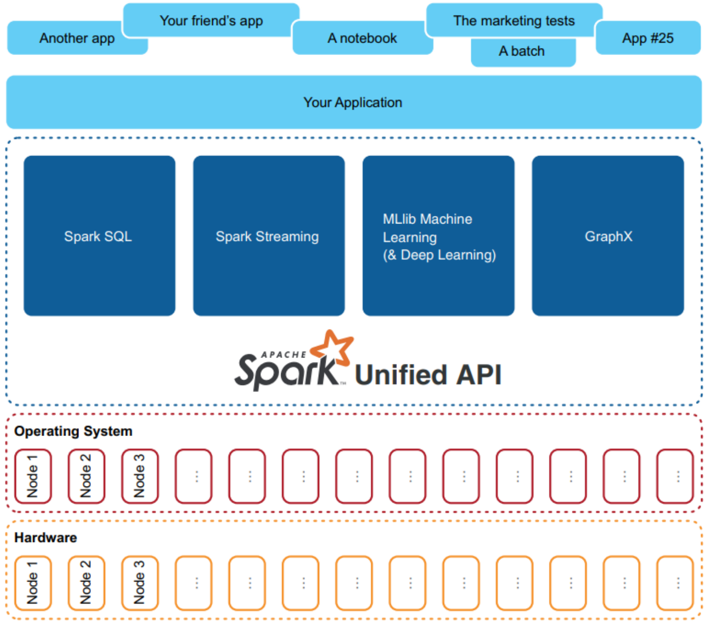
Your applications use a powerful set of APIs for data processing, analytics, machine learning, artificial intelligence …

The operating system abstracts the hardware of a single node.

The hardware is completely abstracted by the operating system and Spark.

Introduction to Apache Spark

- **Apache Spark** is a Java Virtual Machine (JVM) based on distributed data processing engine that scales, and it is fast compared to many other data processing frameworks.
  - **Spark™ is a fast and general engine for large-scale data processing.**
- The biggest claim from Spark regarding speed is that it is able to "**Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk**".
- According to Apache Spark, it reduces **disk I/O latency** by carrying out all processing in the main memory of the worker nodes.
- The other advantage of Spark offers is the ability to chain the tasks at an application programming level without writing onto the disks at all or minimizing the number of writes to the disks.



Spark was originated at the University of California Berkeley and later became one of the top projects in Apache.

Introduction to Apache Spark

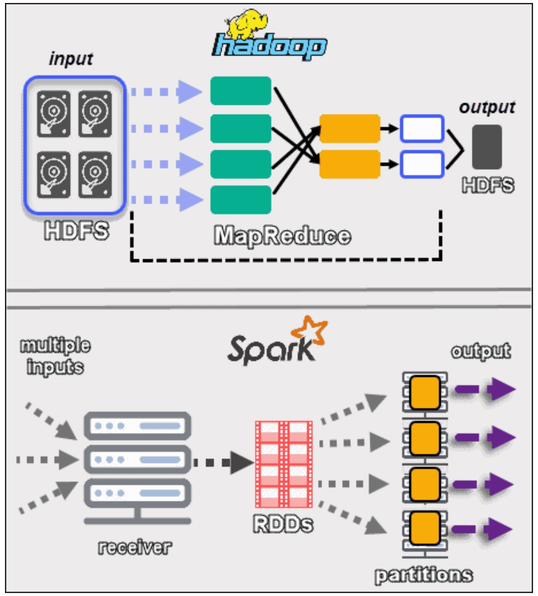
- **Apache Spark** is a general-purpose, distributed cluster computing, data processing framework that, like MapReduce in Apache Hadoop, offers powerful abstractions for the processing of a large dataset.

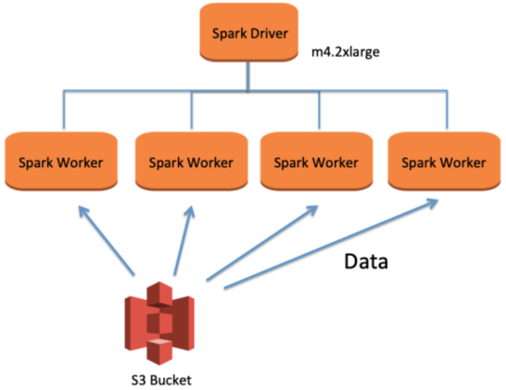
- **Apache Spark** is designed to work with Hadoop*, Amazon S3, Cassandra or as a standalone application.

- Support languages for Spark: Scala, Java, Python and R

- Rich set High level APIs and increases user productivity.

- Integration with new & existing system.




Spark Driver

m4.2xlarge

Spark Worker

Spark Worker

Spark Worker

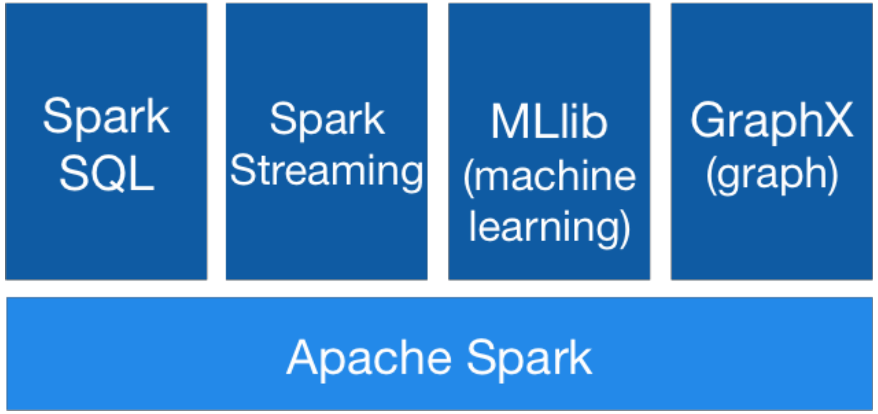
Spark Worker

Data

S3 Bucket

Introduction to Apache Spark

- Big Data analytics platform - written in Scala
- Supports wide range of analytics tasks
  - **Load Data**
  - **SQL**
  - **ML**
  - **Streaming**
- Same computing engine!
- Consistent set of APIs
- Tasks are easier and more efficient to write
  - **Spark** can be deployed in a standalone mode on a single node having a supported OS.
  - **Spark** can also be deployed in cluster node on Hadoop YARN as well as Apache Mesos.
  - **Spark** can be deployed in the Amazon EC2 cloud as well.



Spark SQL

Spark Streaming

MLlib (machine learning)

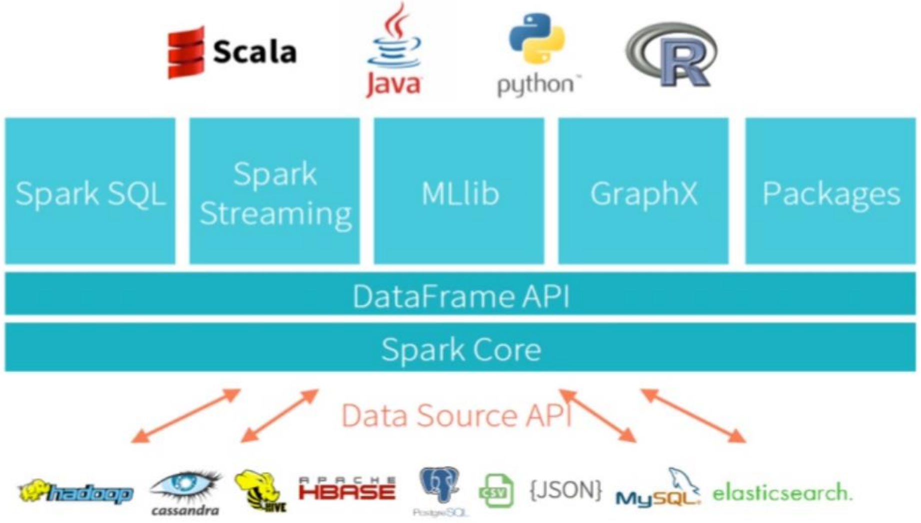
GraphX (graph)

Apache Spark

Apache Spark

Components

- The **Spark core** is complemented by a set of powerful and higher-level libraries
  - SparkSQL
  - Spark Streaming
  - MLlib (for machine learning)
  - GraphX
  - Packages (for example, SparkDL)
- Scala is the language in which Spark is written.

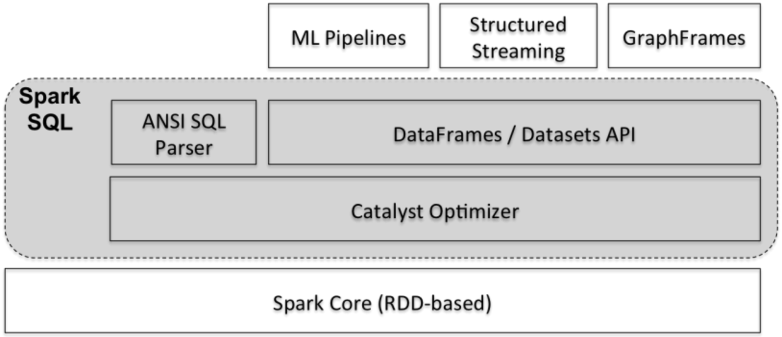


Dataframe is equivalent to a table in a relational database

- **Spark Core** is the base engine for large-scale parallel and distributed data processing. It is responsible for
  - memory management and fault recovery
  - scheduling, distributing and monitoring tasks, jobs on a cluster
  - interacting with storage systems

Apache Spark Component

Spark SQL

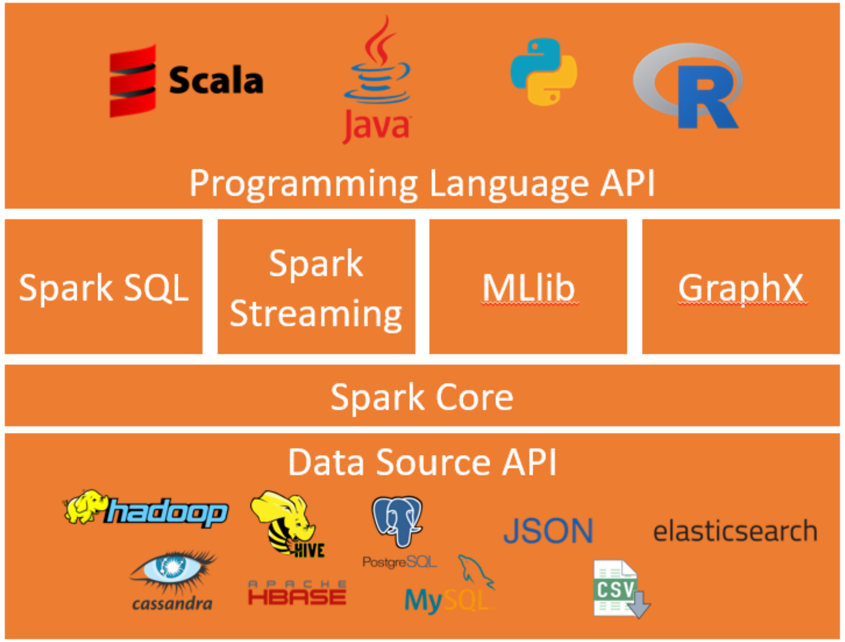


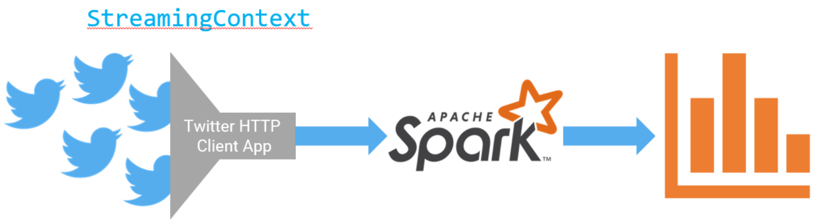
- **SparkSQL** is one of the most advanced components of **Apache Spark**.
- In is clear from the figure that Spark SQL components provide the foundation for Spark machine learning applications, streaming applications, graph applications, and many other types of application architectures.
- **SparkSQL** is a Spark component that supports querying data either via **SQL** or via the **Hive Query Language**.
- It originated as the **Apache Hive** port to run on top of **Spark** (in place of **MapReduce**) and is now integrated with the Spark stack.
- Below is an example of a Hive compatible query:
  - // sc is an existing
  - SparkContext.val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
  - sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
  - sqlContext.sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")
  - // Queries are expressed in HiveQL
  - sqlContext.sql("FROM src SELECT key, value").collect().foreach(println)

Apache Spark Component

Spark Streaming

- Data is generated continuously from one or many sources
- Sources typically send in data simultaneously
- Data comes in small packages (kilobyte scale) in succession
- A lot of applications use continuously-updated data
- **Examples:**
  - Sensors in vehicles, industrial equipment, and machinery send data to streaming for performance measurement.
  - Solar power company monitoring panel performance through streaming
  - Online gaming company collecting streaming data about player-game interactions
  - Spark streaming can use it to track most popular hashtags in 5 mins windows based on their counts in a Twitter stream, and by using the StreamingContext function.


Scala
Java
Programming Language API
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Source API


StreamingContext
Twitter HTTP Client App

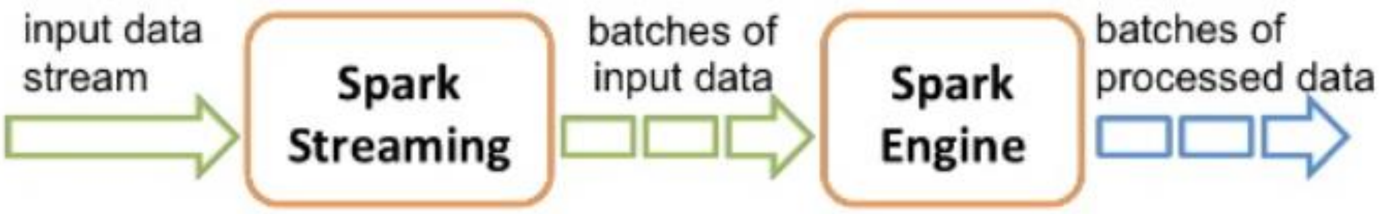
Apache Spark Component

Spark Streaming

- **Spark Streaming** supports real time processing of streaming data, **Spark Streaming** is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

- Data can be ingested: Kafka, Flume, Twitter, ZeroMQ, Kinesis or TCP sockets.

- Processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data streams.

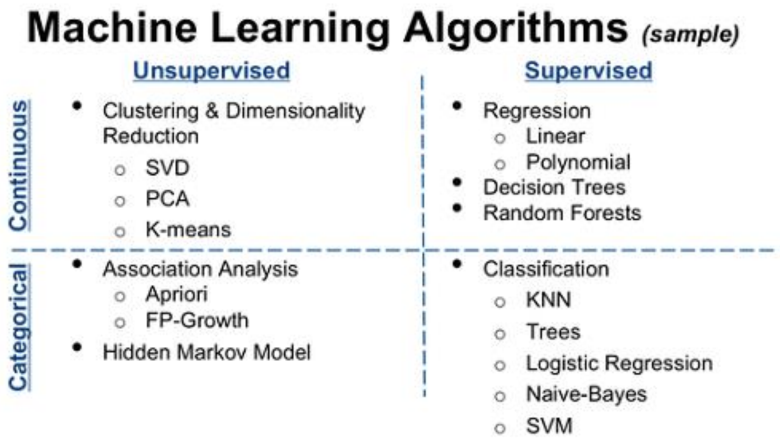


- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

Apache Spark Component

MLib

- **Apache Spark** comes with two flavours of the machine learning library. They are **spark.mllib** and **spark.ml**.
- The first one is developed on top of Spark's **RDD** abstraction, and the second one is developed on top of **Spark's DataFrame** abstraction.
- It is recommended to use the **spark.ml** library for any future machine learning application developments.
- **MLlib** is a machine learning library that provides various algorithms designed to scale out on a cluster for **classification, regression, clustering, collaborative filtering**, and so on.
- These algorithms also work with streaming data, such **as linear regression** using ordinary least squares or k-means clustering (and more on the way).



Machine Learning Algorithms *(sample)*

	Unsupervised	Supervised
Continuous	• Clustering & Dimensionality Reduction ○ SVD ○ PCA ○ K-means	• Regression ○ Linear ○ Polynomial • Decision Trees • Random Forests
Categorical	• Association Analysis ○ Apriori ○ FP-Growth • Hidden Markov Model	• Classification ○ KNN ○ Trees ○ Logistic Regression ○ Naive-Bayes ○ SVM

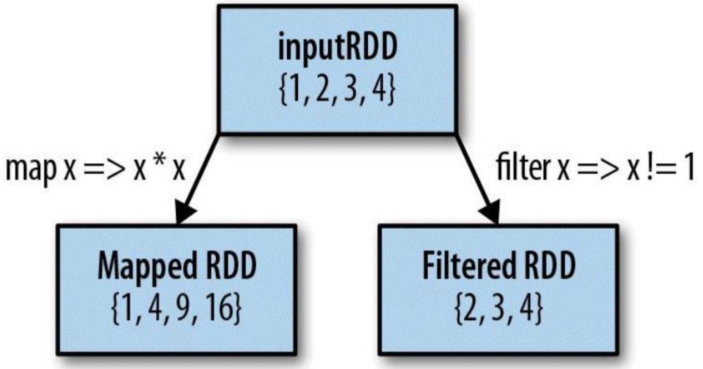
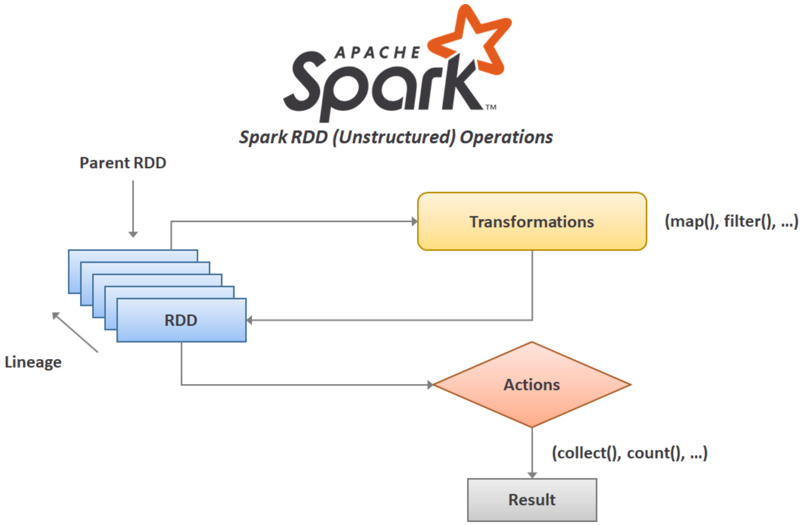
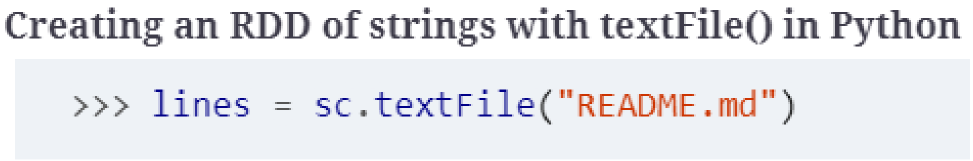


Figure 3-2. Mapped and filtered RDD from an input RDD

RDD Basics

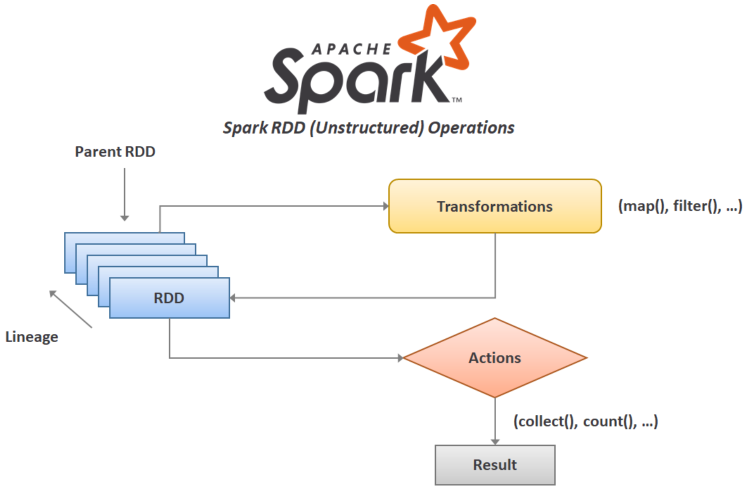
- A **RDD** in Spark is simply an immutable distributed collection of objects.
- Each **RDD** is split into multiple partitions, which may be computed on different nodes of the cluster. **RDDs** can contain any type of Python, Java, or Scala objects, including user-defined classes.
  - Users create **RDDs** in two ways
  - by loading an external dataset
  - by distributing a collection of objects (e.g., a list or set) in their driver program.
- We have already seen loading a text file as an **RDD** of strings using SparkContext.textFile(), as shown in Example.




Creating an RDD of strings with textFile() in Python

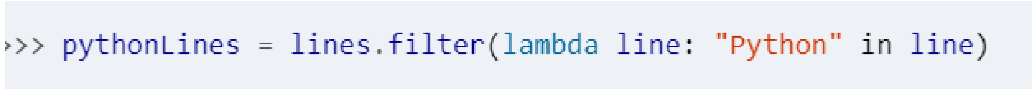
```
>>> lines = sc.textFile("README.md")
```

RDD Basics



- Once created, **RDDs** offer two types of operations:
  - **Transformations**
  - **Actions**
- **Transformations** construct a new **RDD** from a previous one. For example, one common transformation is filtering data that matches a predicate. In our text file example, we can use this to create a new **RDD** holding just the strings that contain the word Python as shown in Example.

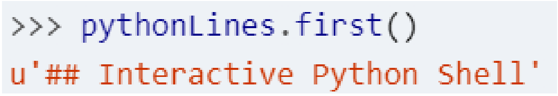
Calling the filter() transformation



```
>>> pythonLines = lines.filter(lambda line: "Python" in line)
```

- **Actions** compute a result based on an **RDD**, and either return it to the driver program or save it to an external storage system (e.g., **HDFS**). One example of an action we called earlier is first(), which returns the first element in an **RDD** and is demonstrated in Example.

Calling the first() action



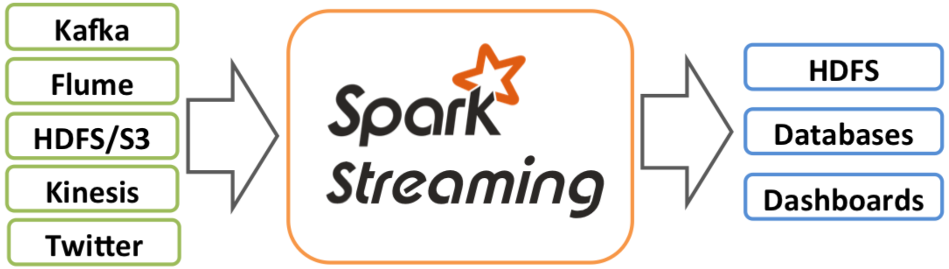
```
>>> pythonLines.first()
u'## Interactive Python Shell'
```

Properties of RDD

- **RDD** which is a fault-tolerant collection of elements/ partitions that can be operated in parallel across the nodes.
- **Properties of RDD**
  - Immutability
  - Cacheable – lineage – persist
  - Lazy evaluation (it different than execution)
  - Type Inferred
- Two ways to create **RDDs**: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, S3, Cassandra or any data source offering a Hadoop InputFormat.
- **Input for Spark**
  - **External Store:** HDFS, Hbase, Hive, S3, Cassandra, Ext3/ Ext4, NTFS
  - **Data Formats:** CSV, Tablimited, TXT, MD, Json, SquenceFile

Spark File Based Input Methods

- **Spark's file-based input methods**, including textFile, support running on directories, compressed files, and wildcards.
- You can use textFile("/my/directory"), textFile("/my/directory/*.txt"), and textFile("/my/directory/*.gz").
- The **textFile** method also takes an optional second argument for controlling the number of partitions of the file.
- Spark creates one partition for each **HDFS** block of the file, but you can also ask for a higher number of partitions by passing a larger value.
  - **JavaRDD<String> distFile = sc.textFile("data.txt");**
- This is in contrast with textFile, which would return one record per line in each file.



Comparison

Spark and Mapreduce

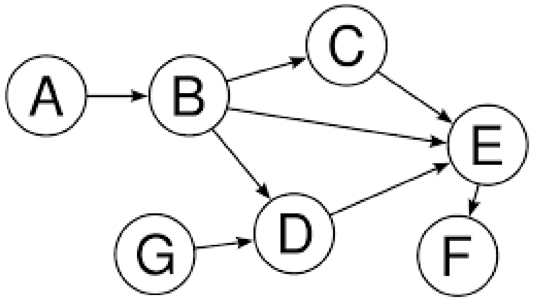
- Spark comes with a very **advanced Directed Acyclic Graph (DAG)** data processing engine.

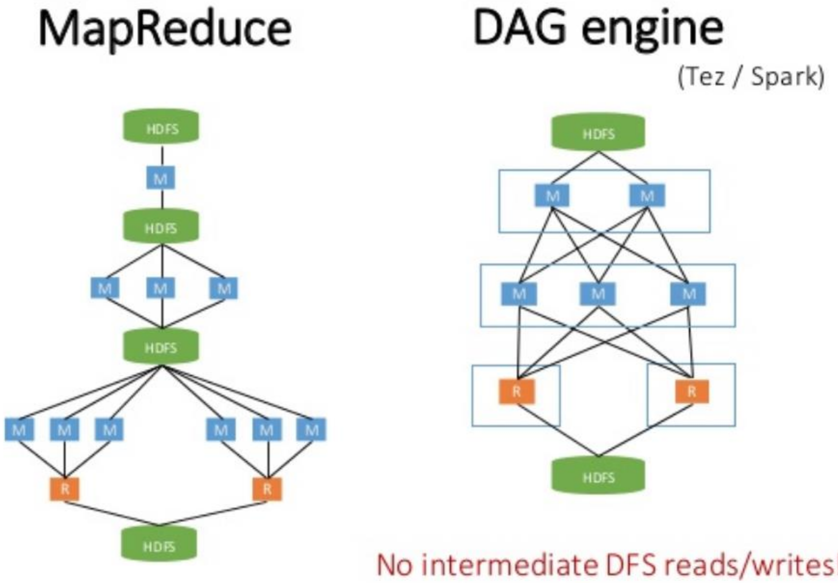
- A **DAG** of tasks is created to be executed by the engine and It consists of a set of vertices and directed edges connecting them.

- In the MapReduce case, the **DAG** consists of only two vertices, with **one vertex** for the **map task** and the other one for the **reduce task**. The edge is directed from the map vertex to the reduce vertex.

- The in-memory data processing combined with its DAG-based data processing engine makes Spark very efficient.

- Spark comes with utilities that can give excellent visualization of the **DAG** of any Spark job that is running.





No intermediate DFS reads/writes!

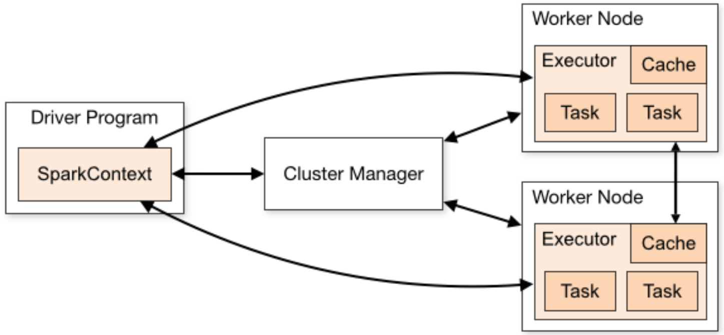
Spark Programming Paradigm

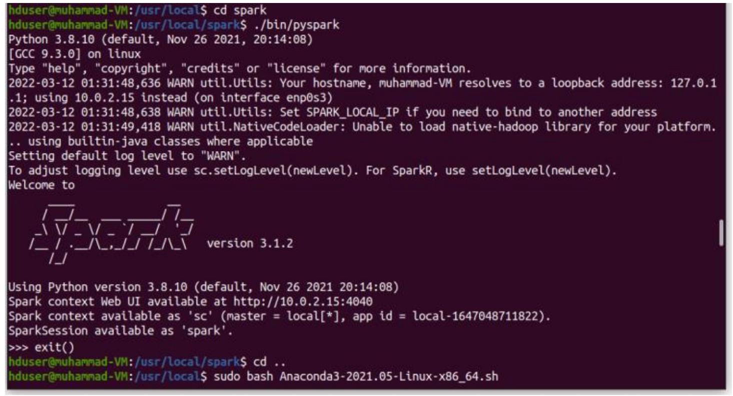
- The Spark programming paradigm is very powerful and exposes a uniform programming model supporting the application development in multiple programming languages.

- **Apache Spark** supports programming in Scala, Java, Python, and R even though there is no functional parity across all the programming languages supported.

- Apart from writing Spark applications in these programming languages, Spark has an **interactive shell with Read, Evaluate, Print, and Loop (REPL)** capabilities for the programming languages Scala, Python, and R.

- The Spark **REPL** is a very versatile tool that can be used to try and test Spark application code in an interactive fashion. The Spark **REPL** enables easy prototyping, debugging, and some other features.





```
hduser@muhammad-VM:/usr/local$ cd spark
hduser@muhammad-VM:/usr/local/spark$ ./bin/pyspark
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
2022-03-12 01:31:48,636 WARN util.Utils: Your hostname, muhammad-VM resolves to a loopback address: 127.0.1
.1; using 10.0.2.15 instead (on interface enp0s3)
2022-03-12 01:31:48,638 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
2022-03-12 01:31:49,418 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.8.10 (default, Nov 26 2021 20:14:08)
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1647048711822).
SparkSession available as 'spark'.
>>> exit()
hduser@muhammad-VM:/usr/local/spark$ cd ..
hduser@muhammad-VM:/usr/local$ sudo bash Anaconda3-2021.05-Linux-x86_64.sh
```

Spark Programming Paradigm

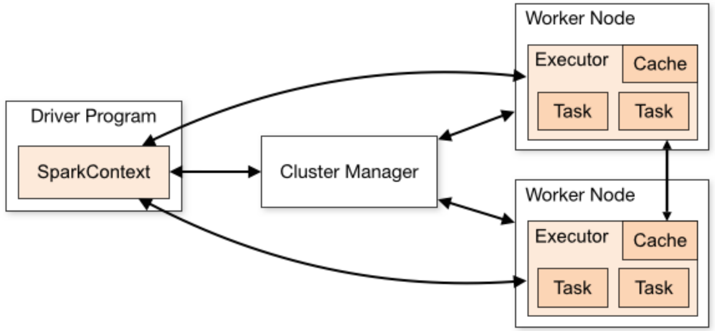
- In any distributed application, it is common to have a driver program that controls the execution and there will be one or more worker nodes.

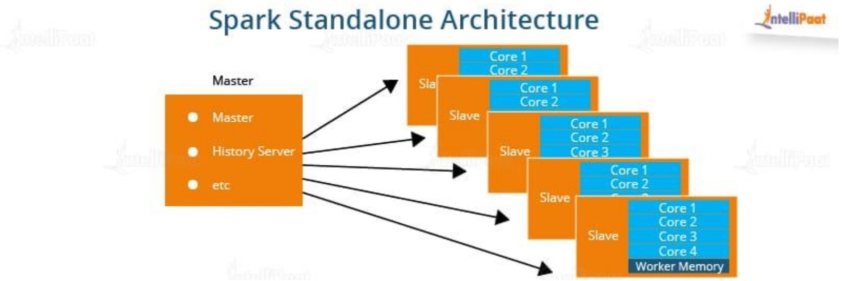
- The driver program allocates the tasks to the appropriate workers.

- In the case of a Spark application, its **SparkContext** object is the driver program and it communicates with the appropriate cluster manager to run the tasks.

- The **Spark master**, which is part of the Spark core library, the Mesos master, and the Hadoop YARN Resource Manager, are some of the cluster managers that Spark supports.

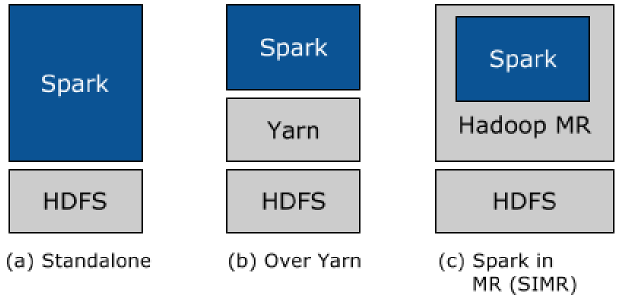
- In the case of a **Hadoop YARN** deployment of **Spark**, the **Spark driver** program runs inside the **Hadoop YARN** application master process or the Spark driver program runs as a client to the **Hadoop YARN**. Figure below describes the standalone deployment of Spark.





Spark and Hadoop Environment

- **Apache Spark** is a cluster computing framework for large-scale data processing.
- It does not use MapReduce as an execution engine and It uses its own distributed runtime for executing work on a cluster.
- However, **Spark** has many parallels with MapReduce, in terms of both API and runtime.
- **Spark** is closely integrated with Hadoop: **it can run on YARN and works with Hadoop file formats and storage backends like HDFS.**
- **Spark** is best known for its ability to keep large working datasets in memory between jobs.
- This capability allows **Spark** to outperform the equivalent MapReduce workflow.

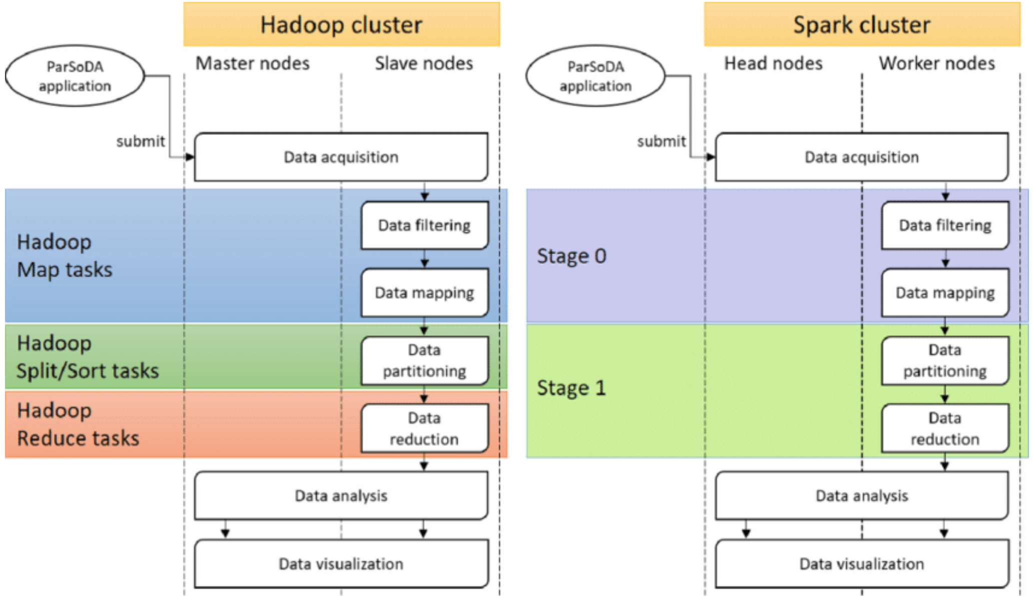


Spark and Hadoop Environment

- **Apache Spark** provides the **RDD** and it supports distributed processing on a cluster of nodes. In case of failure of nodes during processing, the spark framework can handle this by using the **RDD**.

- If there is a huge amount of data to be processed and there are nodes available in the cluster, the framework should have the capability to split the big dataset into smaller chunks and distribute them to be processed on more than one node in a cluster, in parallel.

- **Spark** is designed from the ground up to have its basic dataset abstraction capable of getting split into smaller pieces deterministically and distributed to more than one node in the cluster for parallel processing, while handling the failures in the nodes.

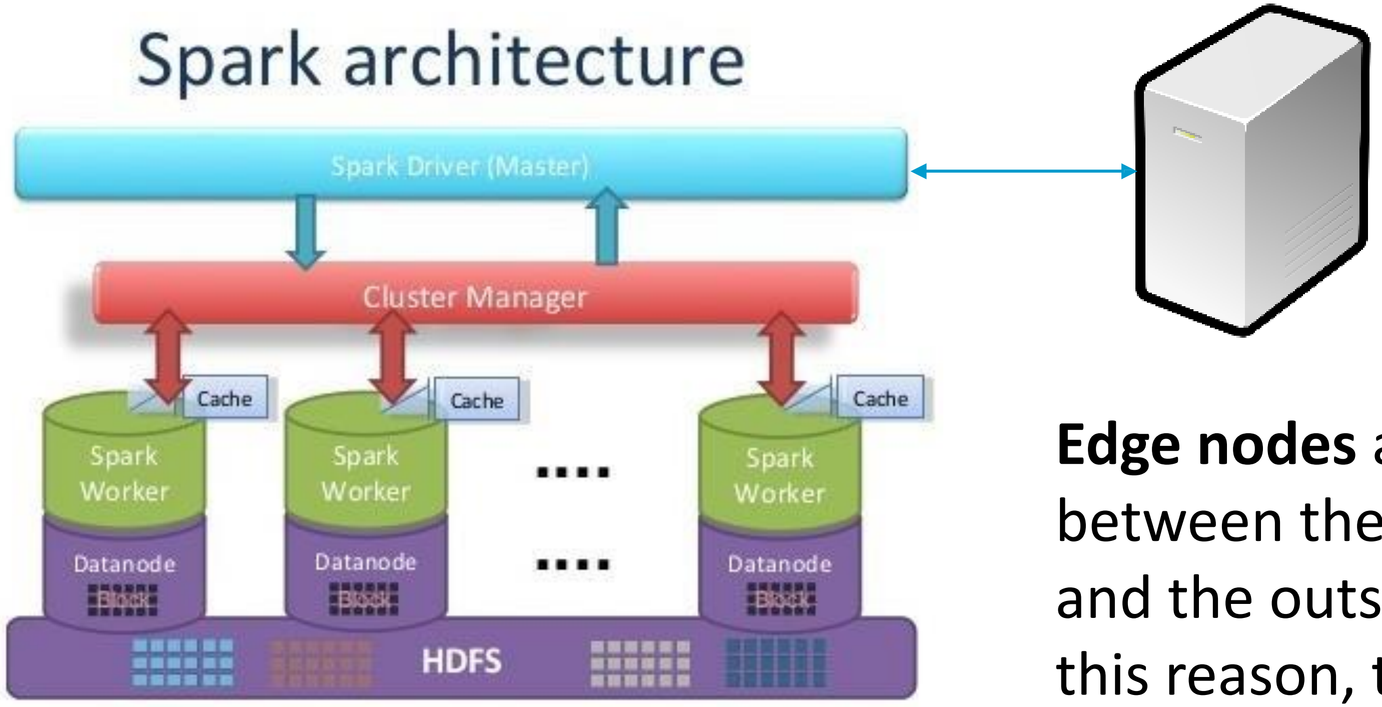


Criteria	MapReduce	Spark
Processing Speeds	Good	Excellent (up to 100 times faster)
Data caching	Hard disk	In-memory / Local Disk
Perform iterative jobs	Average	Excellent
Independent of Hadoop	No	Yes
Machine learning applications	Average	Excellent

Spark and the Hadoop Environment

- Data is expensive to move
  - e.g., loading to local R / Python / etc., engines
- It is always better to do the computation where the data resides due to the following reasons
  - Security
  - Resources and etc.
- Spark limits its scope to being a Computing Engine
- Spark can **read from** and **write to** a wide variety of sources
  - HDFS
  - Hive
  - Kafka
  - Cassandra
  - S3 (and other Cloud)

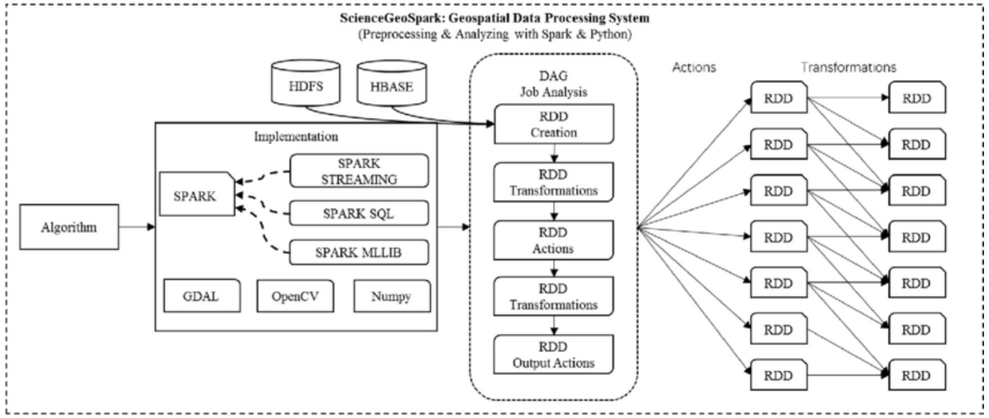
Apache Spark Architecture



Edge nodes are the interface between the Hadoop cluster and the outside network. For this reason, they are referred to as gateway **nodes**.

Spark Programming Model

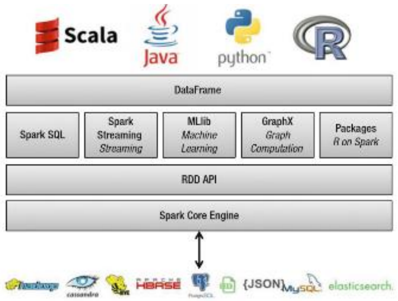
- In Functional Programming, the output of a function like mathematics function, only depends on the inputs to the function.
  - The same inputs will produce the same outputs in all circumstances
  - Declarative Programming Paradigm (like SQL)
- Scala is the scalable, object-oriented, functional and statically typed language.
  - Compiles to Java bytecode
  - Supports Java libraries
- Spark is written in Scala
  - Main programming language for Spark is Scala
  - Some APIs, e.g. GraphX, are only available in Scala



Spark Programming Model

- **How is SparkR going to help the data scientists to do better data processing?**
- The data processing occurs on the single computer on which the **R** installation is available.
- Moreover, if the data size is more than the main memory available on the computer, **R** will not be able to do the required processing.
- With the **SparkR package**, there is access to a whole new world of a cluster of nodes for data storage and for doing data processing.
- With the help of **SparkR** package, **R** can be used to access the **Spark DataFrames** as well as **R** DataFrames.
- It is very important to know the distinction between the two types of data frames, **R Dataframes and Spark Dataframes.**
- An **R DataFrame** is completely **local** and a data structure of the R language.
- A **Spark DataFrame** is a **parallel collection of structured data** managed by the Spark infrastructure.

Spark Programming Model



- When a **Spark DataFrame** is converted to an **R DataFrame**, it should fit in the available memory of the computer.

- By converting an **R DataFrame** to a **Spark DataFrame**, the data can be distributed and processed in parallel.

- By converting a **Spark DataFrame** to an **R DataFrame**, a lot of computations, charting, and plotting that is done by other R functions can be done.

- In a nutshell, the **SparkR** package brings the power of distributed and parallel computing capabilities to R.

- **Spark** with **R** can be used to process the entire raw data and finally, the aggregated and summarized data can be used to produce the reports, charts, or plots.

- Because of Spark's ability to process data at scale, **Spark** with **R** can replace the entire ETL pipeline and do the desired data analysis with **R**. So **Spark** with **R** is a good alternative to packages such as **dplyr**.

Spark Programming with R

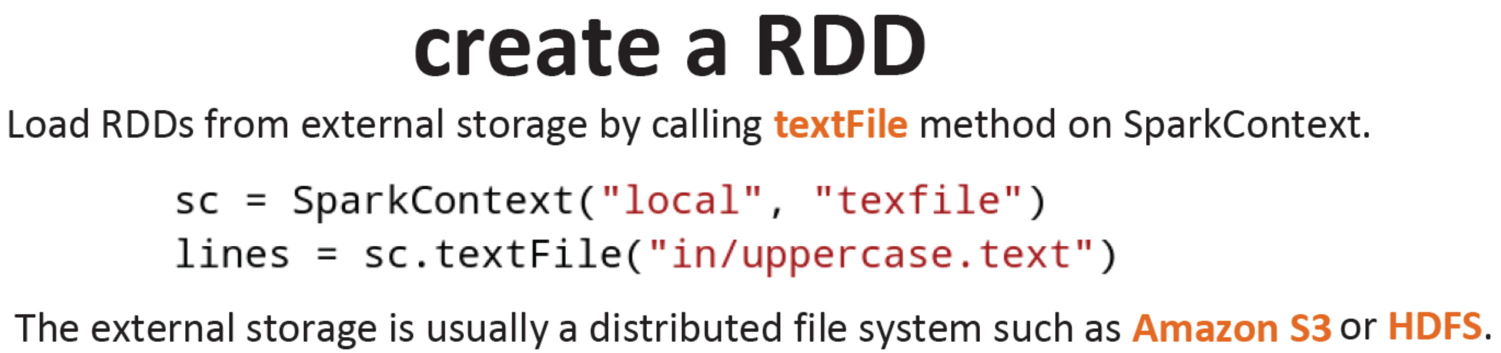
- Other languages are available to work with Spark
- **Java** (because it can use Scala libraries)
- **Python** – using **PySpark** libraries that implement FP concepts (Check Tutorial 4)
- **R** – two options: SparkR and SparklyR
- **PySpark** is used extensively, and default sets of libraries tend to be installed on all worker nodes (numpy, scipy, sklearn, etc.) – e.g., Anaconda
- **SparkR** is basically a tool for running R on Spark. In order to use SparkR, we simply import it in our environment and run our code. It is similar to Python except that it follows R's syntax rather than Python's. Almost everything available in Python is currently available in SparkR.
- **SparklyR** is from creators of Dplyr, Tidy, etc. The **SparklyR** package is a tool for working with large datasets in an interactive environment. In this case, it is an R-based interface to Apache Spark, filtering and aggregating Spark datasets and then bringing them into R for analysis and visualization.

Spark Programming Model

- **Spark RDDs** (Resilient Distributed Dataset)
- Like a Collection object in Scala
- Support functions as parameters
- Immutable – once created, you can't change it
- Distributable – parts of the RDD are divided onto different JVMs
- Lives in memory (unless it can't!)
- Strongly typed – using Java/Scala/Custom types

AccountNo	FirstName	LastName	AccountBalance
SB001	John	Mathew	250
SB002	Tracy	Mason	450
SB003	Paul	Thomson	560

This RDD would be of type RDD[(string, string, string, double)]



create a RDD

Load RDDs from external storage by calling **textFile** method on SparkContext.

```
sc = SparkContext("local", "texfile")
lines = sc.textFile("in/uppercase.text")
```

The external storage is usually a distributed file system such as **Amazon S3** or **HDFS**.

Spark Programming Model

- **Transformations**
- .filter(function)
- .distinct()
- .first()
- .union()
- .map(function)
- .flatMap(function)

- **Actions**
- .reduce()
- .count()
- .persist()
- .collect()
- .saveAsTextFile()
- .saveAsSequenceFile()

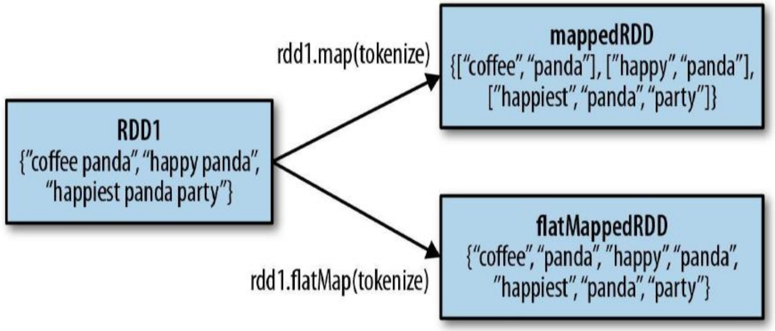

rdd1.map(tokenize)
mappedRDD
{["coffee", "panda"], ["happy", "panda"], ["happiest", "panda", "party"]}
RDD1
{"coffee panda", "happy panda", "happiest panda party"}
flatMappedRDD
{"coffee", "panda", "happy", "panda", "happiest", "panda", "party"}
rdd1.flatMap(tokenize)

Figure 3-3. Difference between flatMap() and map() on an RDD

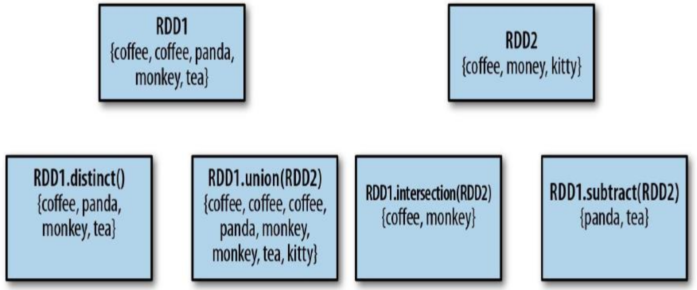
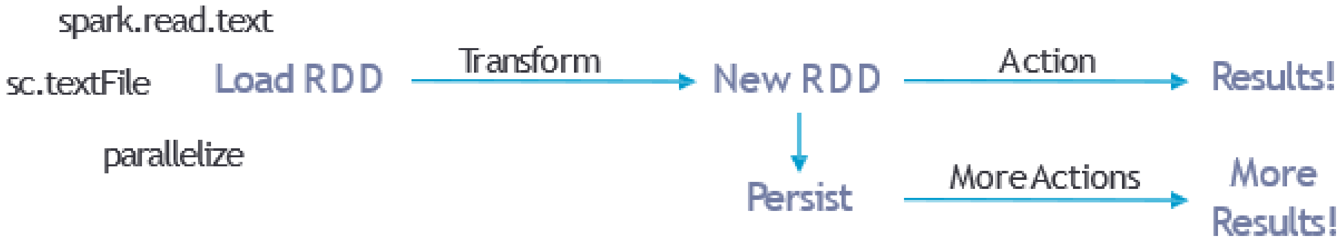


Figure 3-4. Some simple set operations



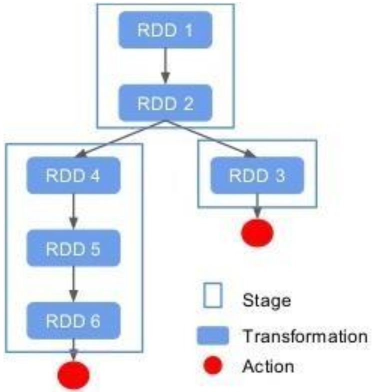
spark.read.text

sc.textFile

parallelize

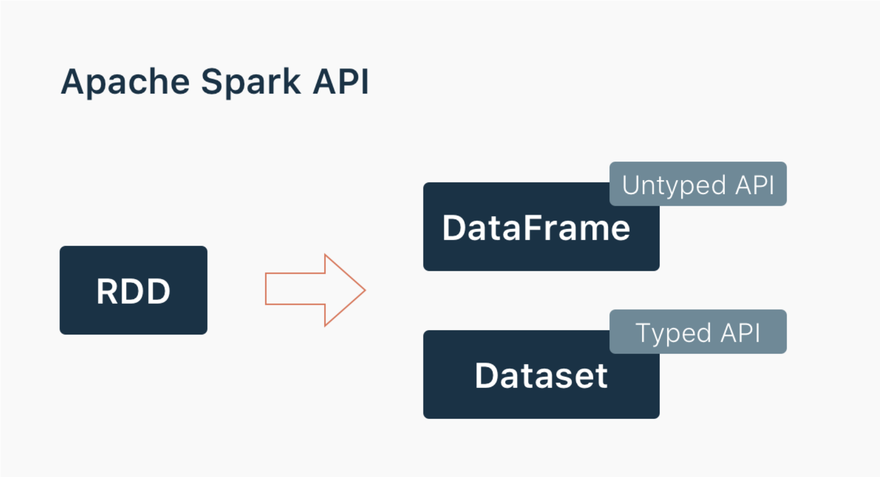
Load RDD → Transform → New RDD → Action → Results!

Persist → More Actions → More Results!

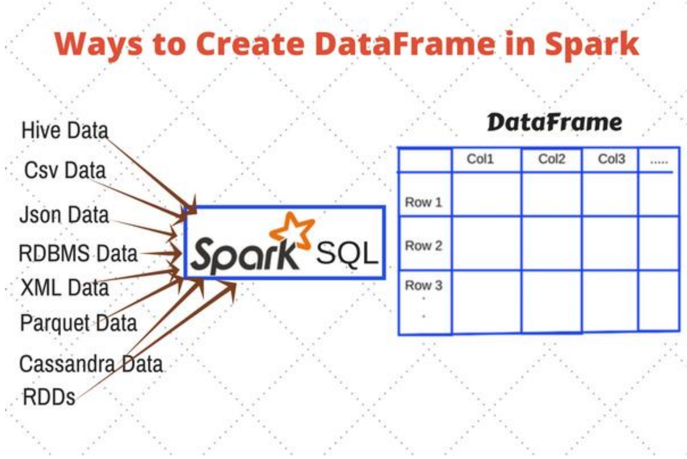


Dataframes and Spark SQL

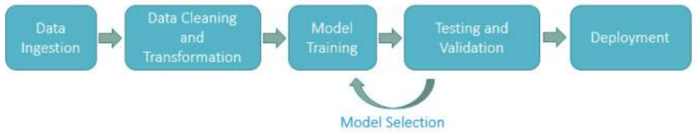
- **Dataframe**
  - Abstraction on top of **RDDs**
  - Basically, created **RDD** containing rows of columns
  - Has a schema
  - Rich, functional language for interaction (similar to Pandas/ Dplyr)



- **SparkSQL**
  - Great way to interact with Dataframes!
  - Also works with Hive metabase
  - `df = spark.sql("SELECT * FROM MYTABLE")`



Spark ML



Data Ingestion → Data Cleaning and Transformation → Model Training → Testing and Validation → Deployment

Model Selection

- Currently, Spark has two machine learning libraries, **Spark MLlib and Spark ML**. **Spark ML** is the newer, scikit-learn inspired machine learning library and is where active development is taking place.
- In addition to inheriting many of the performance considerations of the **RDD** and **Dataset APIs** that they are based on, these machine learning libraries have their own considerations as well.
- The Spark ML API is built around the concept of a “pipeline” consisting of the different stages. Each stage performs a separate task, and stages exist for tasks ranging from data cleaning, to feature selection, through applying a machine learning algorithm.
- For those familiar with scikit-learn much of the design will seem familiar. These pipeline stages are grouped into estimators and transformers.
- **Estimator:** An algorithm that trains on the data provided in a Spark DataFrame and creates a model called as **a Transformer**
- **Transformer:** Converts a Spark DataFrame containing features and transforms it to another Spark DataFrame containing predictions.
- **Parameter:** Used by the Estimators and Transformers. Usually specific to the machine learning algorithm. Spark ML comes with a uniform API for specifying the right parameters to the algorithms.
- **Pipeline:** This is a chain of Estimators and Transformers working together forming a machine learning workflow.

Resources/ References

- Learning Spark, Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia, O'Reilly Media, Inc., 2015.
- Spark in Action, Second Edition, Jean-Georges Perrin, Rob Thomas, Manning Publishers, 2020.
- Lublinsky B., Smith K. T. and Yakubovich A 2013, Professional Hadoop Solutions, Wrox [ISBN: 13:978-11186]
- Holmes A 2012, Hadoop in Practice, Manning Publications [ISBN: 13:978-16172]
- McKinney W. 2012, Python for Data Analysis, O'Reilly Media [ISBN: 13: 978-14493]
- https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- Some images are used from Google search repository (https://www.google.ie/search) to enhance the level of learning.