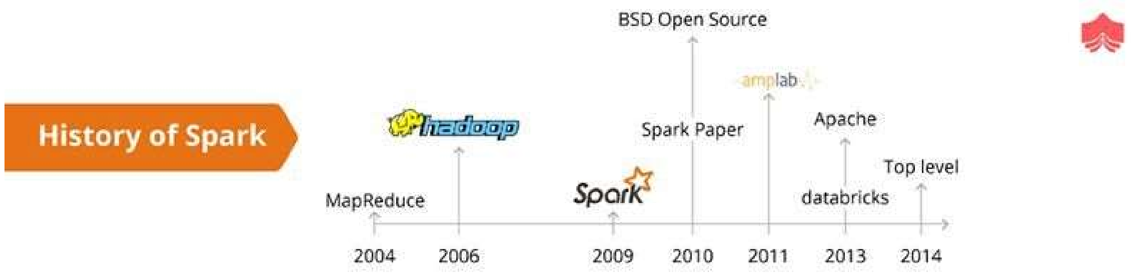
5.1 What is Spark?

So far we have learnt about Big Data analytics and some of the algorithms and techniques that allows us to solve problems conventional analytics solutions cannot solve. With Hadoop, HDFS and MapReduce, many big data solutions have been developed and deployed in the last decade. However, MapReduce over the years has its issues with speed and options for more advanced analytics such as machine learning. Hence, Spark was created to overcome the inefficiencies and complexities found in MapReduce.

Back in 2009, researchers at University of California Berkeley, more specifically the AMPLab (now known as the RISELab), started working on a simpler framework to overcome MapReduce's weaknesses when it comes to interactive or iterative computing tasks. This gave birth to the Spark project that began with the demonstration of 10 to 20 times speed improvements over Hadoop MapReduce for some tasks. Over the years, the team and the open source community continue to make improvements to Spark until where it is today - a popular big data analytics tool that continues to gain increasing popularity and adoptions.



Source: Evolution of Apache Spark (upGrad KnowledgeHut, n.d.)

In 2013, Spark was donated by its original creators and researchers to the Apache Software Foundation, an American nonprofit corporation that supports a large number of open-source software projects.

Apache Spark is a unified engine designed for large-scale distributed data processing. It supports both on-premise and also cloud data processing. As opposed to Hadoop MapReduce, it does not rely on hard disk space but on memory storage for intermediate computation. This is attributed to its biggest advantage over Hadoop MapReduce - speed.

The design philosophy of Apache Spark can be described as the following:

- Speed
- Ease of use
- Modularity
- Extensibility

While many literatures may sound like Spark is way better than MapReduce, in reality, it all depends on what kind of applications you are working on. MapReduce can still work on a large amount of data in parallel without a requirement on memory resources. Spark, while being many times faster than MapReduce, will have a stricter requirement on the amount of memory in the cluster in order to achieve the speed and advantage it promises.

References

1. Apache Spark. (n.d.). Apache Spark's history. https://spark.apache.org/history.html
2. upGrad KnowledgeHut. (n.d.). Evolution of Apache Spark. https://www.knowledgehut.com/tutorials/apache-spark-tutorial/apache-spark-evolution

5.1.1 Design Philosophy of Spark

Four design philosophies of Apache Spark were mentioned in the previous slide. Let's go through them in detail now:

Speed

Spark has been created to overcome the speed issue faced by MapReduce. It achieved the speed advantage by performing the computation on memory instead of hard disk. With commodity servers becoming cheaper relatively, with huge amount of memories and multiple cores, Spark can be optimised to make full use of these with efficient multi-threading and parallel processing in memory.

Spark also utilises its query computations using directed acyclic graphs (DAG). The DAG scheduler and query optimiser allow Spark to keep track of operations applied. This also enables Spark to decompose the processing into tasks to be executed in parallel across workers on the cluster.

Ease of use

By providing a simple logical data structure called Resilient Distributed Dataset (RDD), Spark provides simplicity when it comes to data transformation and actions. Apart from that, Spark also offers a relatively simple programming model for one to build big data applications. One can code without thinking much in distributed or parallel processing and Spark will enable that within its engine.

Modularity

As Spark can be access via several popular programming languages (Scale, Java, Python, R and SQL), it makes coding operations in Spark more convenient through the unified libraries and the core components modules. These modules are such as Spark SQL, SparkMLlib (for machine learning), Spark Structured Streaming and GraphX. There is no need to separate one's code or learn separate APIs or work on distinct engines. The unified processing engine in Spark allows you to work on one single Spark application.

Extensibility

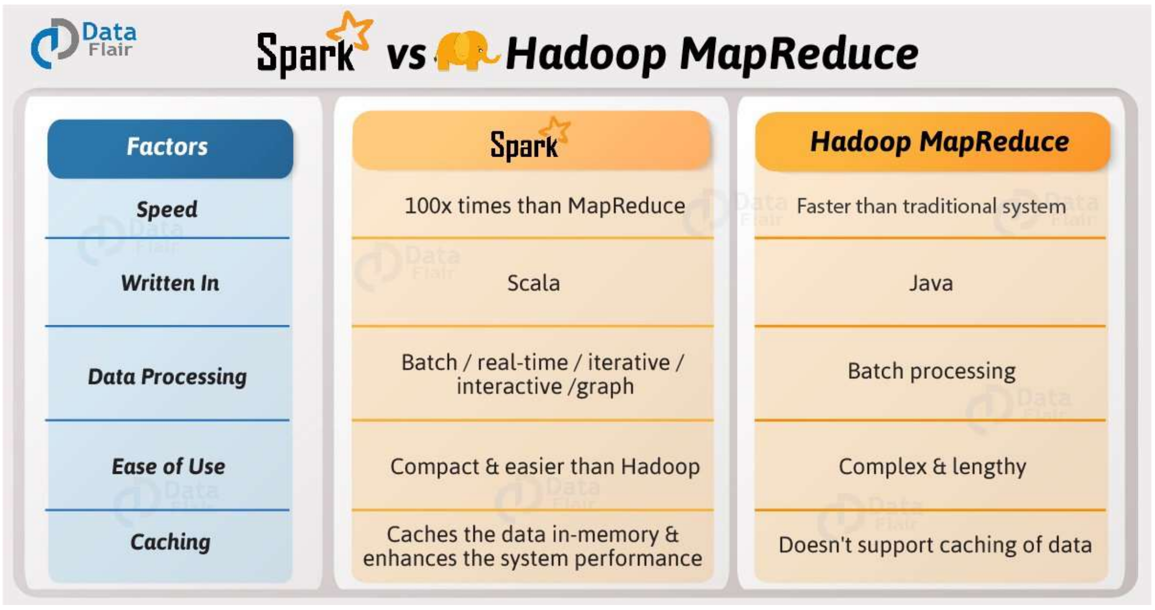
Spark is designed with extensive list of third-party packages to allow users to connect to external data sources and performance monitors. This enables users to have the flexibility of choosing a preferred data source such as Apache HBase, MongoDB, RDBMS and even Apache Hadoop.

Reflection moments

> *From what you already know about Hadoop and MapReduce, can you see the differences between Hadoop, MapReduce and Spark? Do you see the advantage of Spark over Hadoop/MapReduce?*

5.1.2 Hadoop MapReduce vs Apache Spark

The following diagram summarises some of the comparisons between Spark and Hadoop MapReduce:



Spark vs Hadoop MapReduce

Factors	Spark	Hadoop MapReduce
Speed	100x times than MapReduce	Faster than traditional system
Written In	Scala	Java
Data Processing	Batch / real-time / iterative / interactive /graph	Batch processing
Ease of Use	Compact & easier than Hadoop	Complex & lengthy
Caching	Caches the data in-memory & enhances the system performance	Doesn't support caching of data

Source: A comparison of Spark vs Hadoop MapReduce (Dataflair team, n.d.)

Let's dive a little deeper into each factor:

Speed

When it comes to speed, especially when we are dealing with Big Data (think of Volume in the 5Vs), MapReduce may still outperform conventional data analytics systems. However, if one also requires to work on real-time data (think of Velocity in the 5Vs), Spark would be a better candidate. The recent versions of Apache Spark is known to run applications up to 100x faster in memory and 10x faster on disk than Hadoop. This is largely due to Spark using the storage of intermediate data in-memory that reduce the reading from/writing to disk (MapReduce processes data by reading from and writing to disk).

Ease of Use/Difficulty

MapReduce requires developers to code both mapper and reducer steps, and this increase the level of difficulties especially for beginners. Every operation in the whole big data analytics process may again be broken down into several operations in MapReduce. Spark, on the other hand, will be relatively easier to code especially when one can put all operations into one single Spark application. Choices of languages will give Spark an edge over MapReduce since one is not restricted to Java or Python (via Streaming) only.

Data Processing

MapReduce is most appropriate when it comes to batch processing tasks, and the applications can be pretty limited to tasks like counting, indexing and log data analysis. Spark is designed to deal with real-time data processing. It can handle data from real-time event streams such as tweets or facebook postings. Spark also allows interactive data processing, where MapReduce does not allow at all.

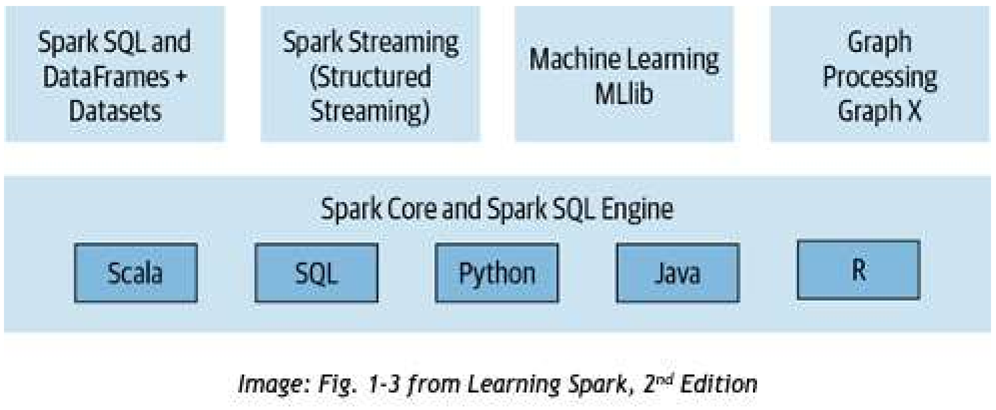
Caching of Data

Only Spark allows caching of data in memory for further iterations. This may bring speed advantages. On the contrary, MapReduce, which depends on hard disk for processing, has no support for data caching at all.
What other factors you may thought about? Take some time to read up and compare. This will help you to understand both systems better.

5.1.3 Unified Engine and Distributed Execution in Spark

Unified Engine

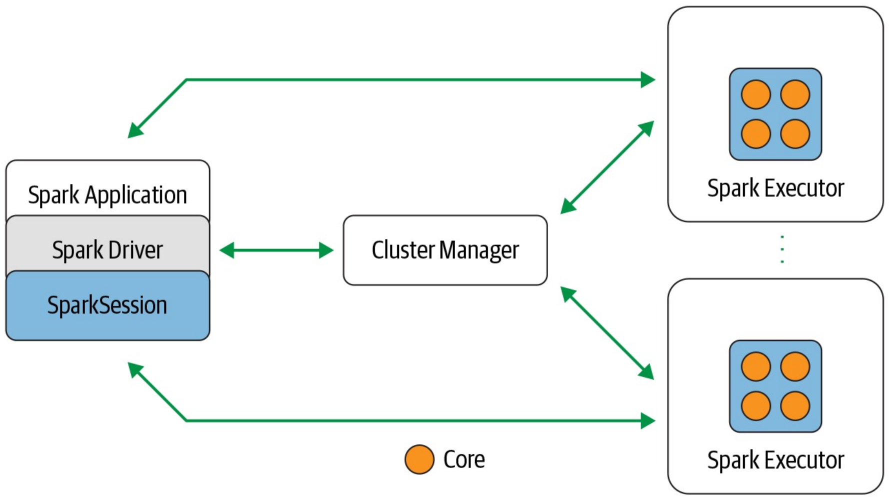
Spark has been designed to have a unified stack of components that is capable of addressing diverse workloads under one single distributed analytics engine. In other systems, it would have taken several separate engines to complete tasks like batch processing, stream and querying. Below is a depiction of the Spark unified engine:


Spark SQL and DataFrames + Datasets
Spark Streaming (Structured Streaming)
Machine Learning MLlib
Graph Processing Graph X
Spark Core and Spark SQL Engine
Scala
SQL
Python
Java
R
Image: Fig. 1-3 from Learning Spark, 2nd Edition

Source: Apache Spark components and API stack (Damji, Wenig, Das & Lee, 2020)

Take a look at the diagram and you will see different kinds of workload such as machine learning or streaming are on top of the Spark core engine. By using the structured APIs provided by Spark, you can use any of the supported languages to access the desired components. The beauty of this engine is how it is able to decompose one's code into highly compact bytecode. This bytecode will be executed in the workers Java Virtual Machines (JVMs) across the cluster.

Distributed Execution



Source: Apache Spark components and architecture (Damji, Wenig, Das & Lee, 2020)

You should noticed by now - Spark can work in standalone and in cluster mode. When it comes to Big Data processing, it is the distributed cluster mode that will give the users the advantages. This concept has been covered in this subject in Week 1 and Week 4.

In Spark's distributed execution environment, a Spark Driver will be responsible to orchestrate all the parallel processing or operations on the cluster. A Cluster Manager is responsible for managing and allocating resources for the Spark cluster. Sparks has own standalone cluster manager, but it also supports Apache Hadoop YARN, Apache Mesos and Kubernetes. Finally, Spark Executors run on worker nodes in the cluster. The Spark Driver communicates with the Spark Executors to instruct them which tasks they need to execute.

Putting the two together

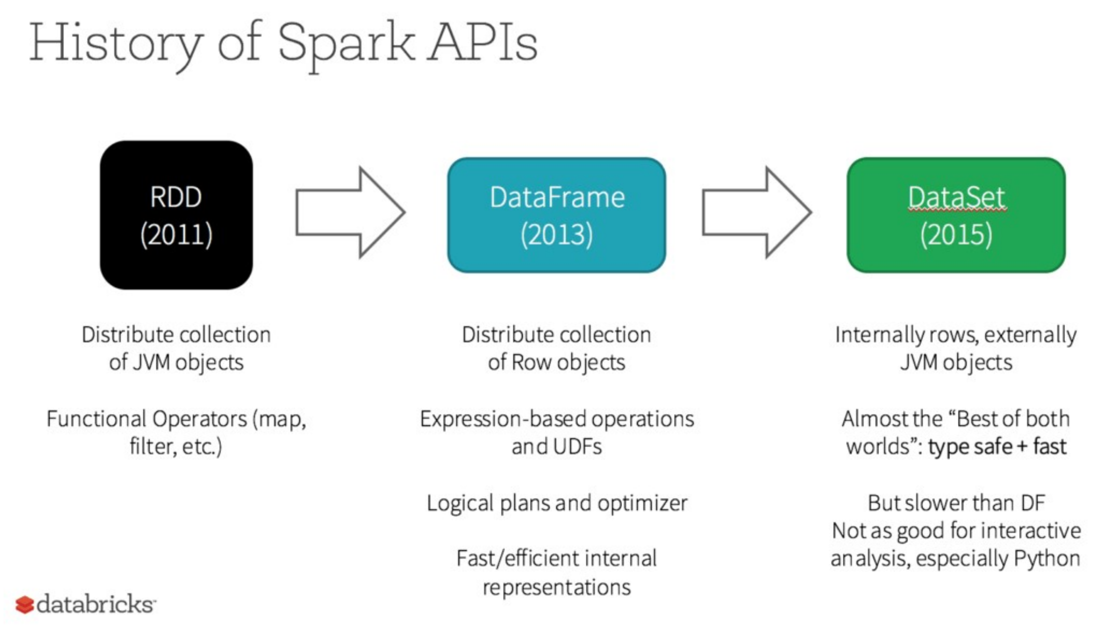
Having seen the above, you should be able to imagine that with Apache Spark, you can code your Big Data application using python. Spark will be able to convert it into the code Spark engine can execute on any available workers in a cluster, via the cluster manager to have your analytics tasks carried out in a cluster. All these can be achieved without you explicitly have to code differently, or to worry about how to manage and monitor your task across multiple nodes in a cluster.

Key reading

Damji, J., Wenig, B., Das, T., & Lee, D. (2020). Chapter 1: Introduction to Apache Spark: A unified analytics engine. In Learning Spark: Lightning-fast data analytics (2nd ed.). O'Reilly Media Inc. https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.pdf

5.1.4 Resilient Distributed Dataset (RDD), DataFrame and DataSet

Resilient Distributed Dataset (RDD) was proposed in a research paper in 2012 by the creators of Spark and it can be seen as the heart of every Spark application. It was designed as an immutable distributed collection of elements of your data. It is partitioned across the nodes in a cluster, and allows one to operate on this data partitions across nodes.



History of Spark APIs

RDD (2011)

Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

DataFrame (2013)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

DataSet (2015)

Internally rows, externally JVM objects

Almost the "Best of both worlds": type safe + fast

But slower than DF
Not as good for interactive analysis, especially Python

databricks

Source: History of Spark APIs (Databricks, n.d.)

As Spark evolves, RDD has also evolved. In 2013, DataFrame (not to be mistaken with DataFrame from Pandas) was introduced to offer better features. In 2015, DataSets was introduced. The two (DataFrame and Dataset) APIs were merged in Apache 2.0 as a unified API, allowing type safety to be practised in Spark.

To know a little more about all three types of APIs, take a look at the following 5-minute 15-second video:

RDD vs Dataframe vs Dataset

https://youtu.be/26zI50iNBp8

Key readings

- Damji, J. (2016). A tale of three Apache Spark APIs: RDDs vs DataFrames and Datasets. Databricks. https://www.databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html
- Databricks. (n.d.). Resilient Distributed Dataset (RDD). https://www.databricks.com/glossary/what-is-rdd
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., Franklin, M.J., Shenker, S., & Stoica, I. (2012). Resilient Distributed Datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI

12), (pp. 15-28).
https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

5.2 Applications of Spark and its importance

Spark is widely used by the industry and academia for big data analytics. Feel free to look at the list of companies and organisations on this Apache Spark webpage.

Watch the following videos for some example use cases of Spark:

- Large Scale In-production Machine Learning Models Monitoring (26 minutes 49 seconds)

How Intuit uses Apache Spark to Monitor In-Production Machine Learning Models at Large-Scale

https://youtu.be/woVFk1Imvu8

Source: (Databricks, 2020)

- Natural Language Processing using Spark NLP (you can stop at minute 12:15 before the live demo starts)

Advanced Natural Language Processing with Apache Spark NLP

https://youtu.be/iHjdqF6iCnU

Source: (Databricks, 2020)

- Autism Diagnostic (1 minute 27 seconds)

Cognoa uses Databricks to help diagnose autism in young children

https://youtu.be/0uR5M7VpJ-w

Source: (Databricks, 2022)

- Loan Qualification Check (1 minute 23 seconds)

PicPay is using Databricks to qualify loan applications across Brazil

https://youtu.be/IDRlDYWsL9Q

Source: (Databricks, 2022)

- Real Estate and Property Business analytics (1 minute 25 seconds)

Compass is using Databricks to grow its real estate agents' businesses year after year.

https://youtu.be/5hpAMigj7d4

Source: (Databricks, 2022)

There are many more use cases if you search around the Internet. From the types of analytics point of view, you will see many are using Spark mainly for its real-time analytics capabilities. The ability to support real-time processing on streaming data at scale makes Spark a real attractive technology. Secondly, the support for Machine Learning (ML) through MLLib gave Spark an edge over MapReduce. The MLlib APIs allow one to leverage on the scale and speed of Spark core and focus on data modelling problems instead of on the distributed processing issues.

If you have looked beyond what has been sampled above, what did you find? Share that with your classmates and instructor and discuss what you think about them?

5.3 Coding time using Python - PySpark

In this section, you will be learning how to code in Spark and Python. Focus on getting familiarised with the pyspark syntax, and ask yourself how different are they compared to your previous analytics experience with conventional techniques and also MapReduce.
Happy coding!

5.3.1 Word Frequency example using Spark

Welcome to your first ever Spark program - the familiar word frequency application. You have done this once using MapReduce in Week 4, didn't you? This time, you will see it live using Spark with python (via pyspark library).

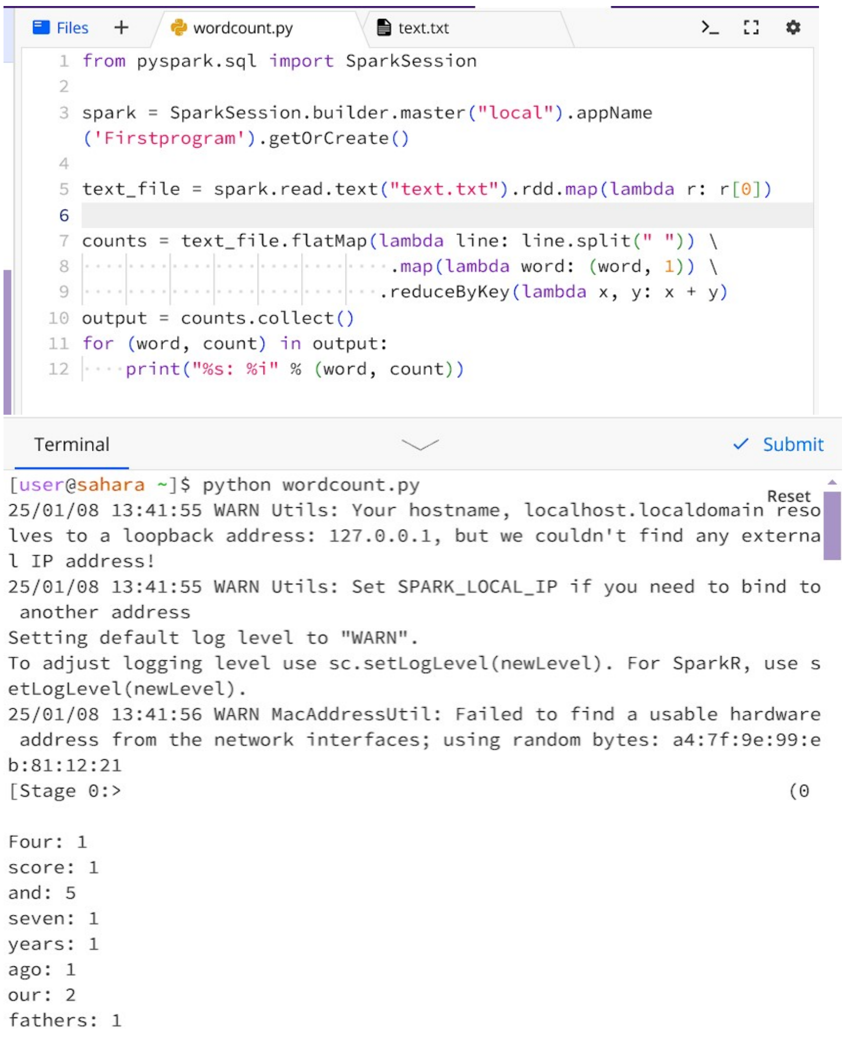
As oppose to the MapReduce codes, you only have/need one single python application. There is also no need to split the file into mapper and reducer. If you try to read the file on your right, you could roughly already guess what the code is doing...

- Line 1: import of the required libraries
- Line 3: initialisation of the spark session
- Line 5: read from the data source and perform some kind of mapping

- Line 7: some magical code (hint: they are called lambda function. To know more, look at bonus material)
- Line 10: Looks like some collection process is taking place
- Line 11-12: This looks familiar again - the results must be found in output and we will be looking the results!

Some vagueness are intentional here - in 5.3.2 we will have line by line explanations to help you understand. But the whole point is - you can generally better understand what the code is trying to do as compared to the MapReduce example.

For this slide, please execute the code wordcount.py and observe the output. Do scroll up and down, and try to interpret the output from the console and ask yourself - do you anticipate the outcome? What really happens there? Things will get clearer when you work on the next slide.



```
1  from pyspark.sql import SparkSession
2
3  spark = SparkSession.builder.master("local").appName
   ('Firstprogram').getOrCreate()
4
5  text_file = spark.read.text("text.txt").rdd.map(lambda r: r[0])
6
7  counts = text_file.flatMap(lambda line: line.split(" ")) \
8                      .map(lambda word: (word, 1)) \
9                     .reduceByKey(lambda x, y: x + y)
10 output = counts.collect()
11 for (word, count) in output:
12     print("%s: %i" % (word, count))
```

```
[user@sahara ~]$ python wordcount.py
25/01/08 13:41:55 WARN Utils: Your hostname, localhost.localdomain reso
lves to a loopback address: 127.0.0.1, but we couldn't find any externa
l IP address!
25/01/08 13:41:55 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to
 another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use s
etLogLevel(newLevel).
25/01/08 13:41:56 WARN MacAddressUtil: Failed to find a usable hardware
 address from the network interfaces; using random bytes: a4:7f:9e:99:e
b:81:12:21
[Stage 0:>                                                            (0

Four: 1
score: 1
and: 5
seven: 1
years: 1
ago: 1
our: 2
fathers: 1
```

5.3.2 Step by Step breakdown of the Word Frequency Spark example

In this slide, we will explain the code line by line to help you see the workings of PySpark and also how familiar it can be if you have some prior Python experience. Use the notebook function to run line by line, observe the outcome and feel free to edit if you wish to find out more. Explanations will be provided in-line in your notebook.

If you are not familiar with Jupyter Notebook, here is a crash course:

You can move your mouse over the numbers on the left (looks like [1], [2] and so on). The number will turn into a "Play" button. Click on that and that particular line of code will be executed.

Sometimes the standard output (things that will get "printed" in the console) will be displayed right below your code. Sometimes you will see nothing. Both are OK. What's not OK, is when you received an error output.

If you have assigned value(s) to a variable, you will not see any output in the console. To view, you can just type the name of the variable in the next line (some of the code example has already that done for you).

You can also select Run All to execute all codes at one go (and in sequence), but that's not so fun, is it? :)

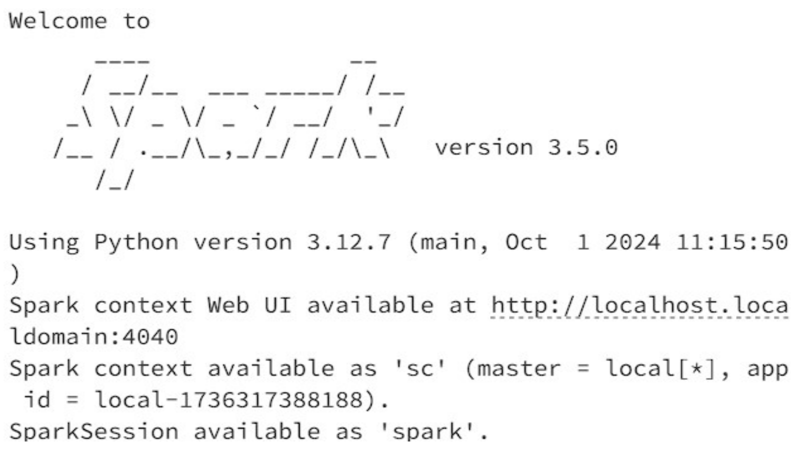
Enjoy and have fun learning at Google Colab

5.3.3 Using PySpark via the console

Now that you are more familiar with Spark using python, why not let's try to do it interactively? In other words, you can use PySpark directly from the console as a shell. It will be even more familiar, if you have worked on python shell before this. From the terminal on your right, at the cursor (by right it should read [user@sahara ~]$) type pyspark

[user@sahara ~]$ pyspark

and you should get something like this as output:



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.12.7 (main, Oct  1 2024 11:15:50
)
Spark context Web UI available at http://localhost.loca
ldomain:4040
Spark context available as 'sc' (master = local[*], app
 id = local-1736317388188).
SparkSession available as 'spark'.
```

If you have entered the PySpark shell, well done! now it is time to repeat the steps you have already done twice, this round, step by step into the shell and observe the output. The text.txt file has been saved in the directory so the instructions below should work.

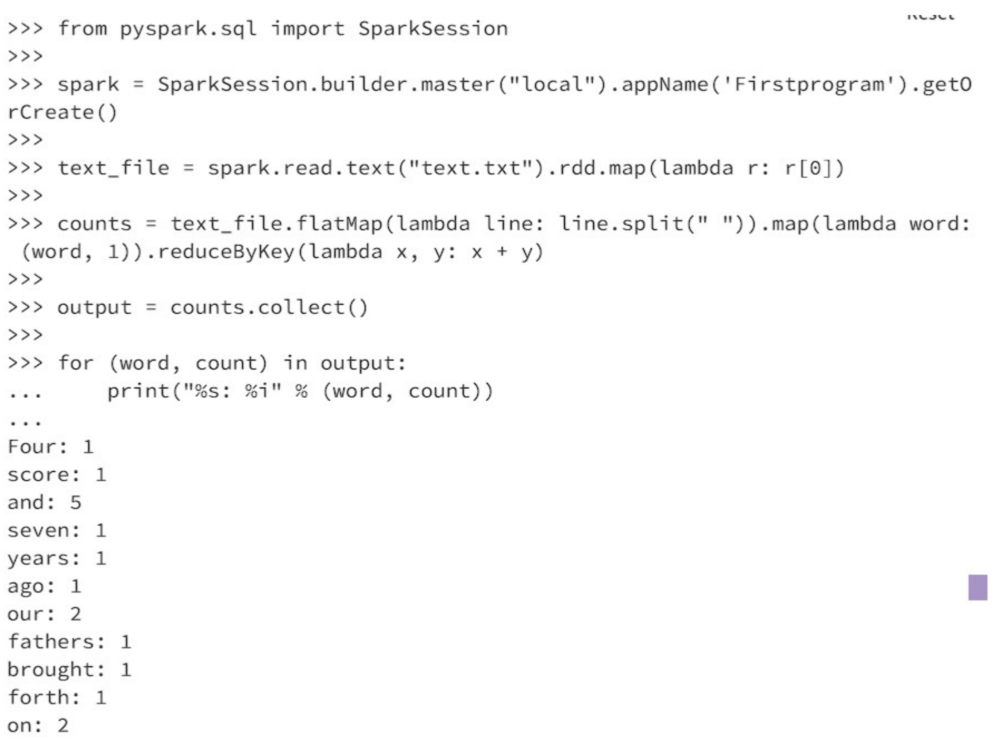
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local").appName('Firstprogram').getOrCreate()
text_file = spark.read.text("text.txt").rdd.map(lambda r: r[0])
counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word,
1)).reduceByKey(lambda x, y: x + y)
output = counts.collect() for (word, count) in output: print("%s: %i" % (word, count))
```

Note that the longer lines have been put into single lines so that it is easier to copy and paste into your terminal. Also, the final two lines (the for loop) can be copied together. You would get the following after pasting:

```
>>> for (word, count) in output:
... print("%s: %i" % (word, count))
...
```

Just press Enter again and the loop will be executed.

If you obtained the exact same output (no surprise right?), congratulations, you have just used PySpark shell to carry out the word frequency count application interactively. The effect is actually not too different from using a notebook, but once you are familiar with the terminal, you will realise this is one of the common ways you will access a master node of a cluster installed in the cloud. Through the terminal, you can remotely perform analytics interactively using Spark to solve your Big Data problems.



```
>>> from pyspark.sql import SparkSession
>>>
>>> spark = SparkSession.builder.master("local").appName('Firstprogram').getO
rCreate()
>>>
>>> text_file = spark.read.text("text.txt").rdd.map(lambda r: r[0])
>>>
>>> counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word:
 (word, 1)).reduceByKey(lambda x, y: x + y)
>>>
>>> output = counts.collect()
>>>
>>> for (word, count) in output:
...     print("%s: %i" % (word, count))
...
Four: 1
score: 1
and: 5
seven: 1
years: 1
ago: 1
our: 2
fathers: 1
brought: 1
forth: 1
on: 2
```

5.3.4 Sorting using Spark

In this example, you will see how besides performing word count, we can also sort by the frequency of the words extracted from the text file text.txt.

Can you think of other processes you may use Spark to perform?

<Google Colab Link>

5.3.5 Spark Challenge: Word length

Based on the previous examples, let's put your knowledge to test.

Time: 60 minutes

Purpose: To apply Spark for analytics

Task:

- Using the Jupyter Notebook, please create a Spark script with Python and PySpark to perform word length calculation.
- Imagine you are given a text, such as the text.txt file we have used so far or perhaps a book from Project Gutenberg, and you are interested in finding out the word lengths of the words in that text/book as well as the average length.

5.3.6 Discussion: Machine Learning using Spark and Python

Time: 30 minutes

Purpose: To apply Spark for analytics

Task:

Take some time to explore a few example MLlib's from the Apache Spark project. Go through each example carefully and refer to the MLlib main guide for a better understanding. Enjoy the learning process!

Github: https://github.com/winnie-2018/big_data_mgmt_from_dummy/blob/main/Learning%20Materials/week_5_files/536_pyspark_ml.ipynb

Google Collab: https://colab.research.google.com/github/winnie-2018/big_data_mgmt_from_dummy/blob/main/Learning%20Materials/week_5_files/536_pyspark_ml.ipynb

5.4 Bonus: Directed Acyclic Graph (DAG)

Watch the following 5-minute 21-second video to further understand Directed Acyclic Graph (DAG) used in Spark.

What is DAG?

https://youtu.be/1Yh5S-S6wsI

Source: (intricity101, 2021)

5.4.1 Bonus: Lambda Functions in Python

Watch the following 7-minute 8-second video to learn about Lambda Functions in Python.

Intermediate Python Tutorial #5 - Lambda Functions

https://youtu.be/BcbVe1r2CYc