

# BigData and Machine Learning with Hadoop and Spark Frameworks

Jean-Marc GRATIEN<sup>1</sup>

<sup>1</sup>Department of Computer Science  
IFP New Energy

January 25th 2022 / Master Data-AI

# Outline I

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX

## Outline II

- Spark : CLI

### 4 Dask

- Introduction
- Architecture
- Dask API
- Dask Ecosystem

### 5 TP

- TP0 : Hadoop Installation
- TP1 : Hadoop Word count
- TP2 : Start with Spark
- TP4 : Spark ML, Data processing
- TP5 : Spark ML, Machine learning
- TP6 : Spark ML, Image processing
- TP7 : Start with Dask

# Objectifs

## Objectifs

- General Overview on Hadoop and Spark
- Introduce to Hadoop
- Introduction to Spark Framework

# Audience and Prerequisites

- Audience : computer science and data scientist students
- Prerequisites :
  - sequential programming in java and python
  - elementary of machine learning, data analytics
  - image processing
- Material(Slide+TPs) available at :
  - git clone `https://github.com/jgratien/BigDataHadoopSparkDaskCourse.git`

# Motivation

## Introduction to Bigdata

### BigData

- What is Bigdata?
- What are the BigData issues?

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Hadoop Framework

## Introduction to Hadoop

- Hadoop definition
  - Java opensource software framework
  - Data storage management
  - Parallel data analysis
  - part of Apache project supported by the Apache Software Foundation



# Hadoop Framework

## Introduction to Hadoop

- Hadoop History
  - 1990 - 2000 : World Wide Web
  - Yahoo, AltaVisa, . . . : first search engines
  - Nutch open source project created by Doug Cutting and Mike Cafarella
  - 2006 : Nutch project is split:  
the distributed storage and computing framework -> Hadoop
  - 2008 : Hadoop 1.0 (Open Source Project proposed by Yahoo)
  - 2012 : Hadoop 2.0 release
  - 2017 : Hadoop 3.0

# Hadoop Framework

## Introduction to Hadoop

### Why Hadoop?

- BigData issues :
  - increasing amount of data amount
  - distributed storage facilities
  - parallel data processing management
  - fault tolerance management

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- **Architecture**
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

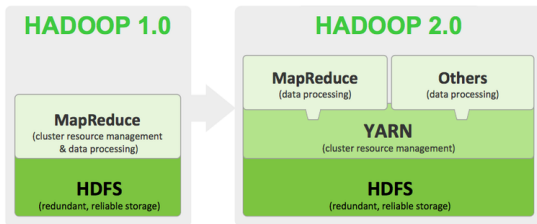
## 4 Dask

# Hadoop Framework

## Hadoop Framework Architecture

the basic Hadoop Framework based of 4 main modules :

- HDFS : Hadoop Distributed File System
- YARN : Yest Another Ressource Negotiator
- Map : parallel data processing
- Reduce : collecting data and producing results



# Hadoop Framework

## Hadoop Ecosystem

Hadoop ecosystem :

- Ambari : Hadoop component and services web interface management
- Cassandra : Distributed Data Base system
- Flume : Data Stream management layer
- HBase : NoSql distributed Data Base
- HCatalog : data storage management
- Hive : data storage with a SQL API
- Oozie : task framework
- Pig : HDFS data processing framework
- Solr : data indexing framework
- Sqoop : SQL DB and Hadoop data transfer framework
- Zookeeper : distributed data processing management

# Hadoop Framework

## Hadoop distributions

Hadoop Distributions :



# Distributions

- Hortonworks
- Cloudera
- MAPR



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- **HDFS**
- Yarn
- MapReduce

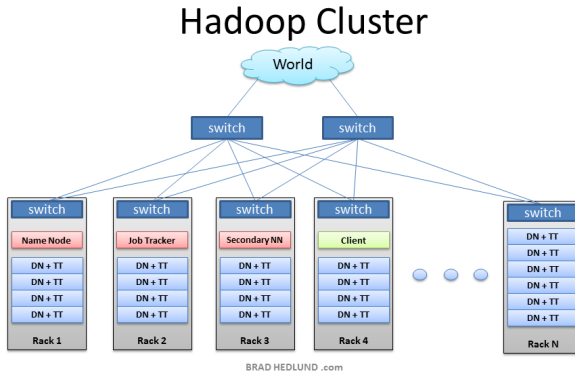
## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Hadoop Framework

## Hadoop Cluster



Purpose : Scalability, Fault tolerance and Reliability management

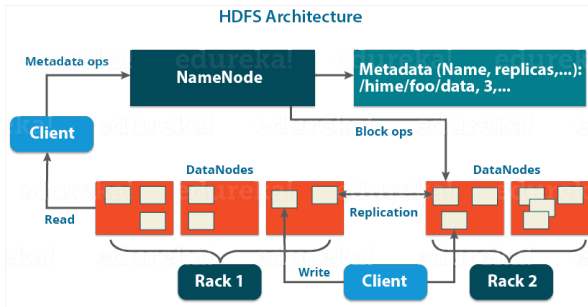


# Hadoop Framework

## Hadoop Cluster

### Concepts

- NameNode
- DataNode
- Replication
- Block, Blocksize



HDFS : Hadoop Distributed Filesystem

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Starting HDFS

```
1  # Format nodes
2  > hadoop namenode -format
3
4  # Starting HDFS services
5  > start-dfs.sh
```

### Shutting down HDFS

```
1  # Stopping HDFS services
2  > stop-dfs.sh
```

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Inserting Data into HDFS

```
1  # Step 1 : Create input directory
2  > $HADOOP_HOME/bin/hadoop fs -mkdir /usr/input
3
4  # Step 2 : copy data from local filesystem to hdfs
   filesystem
5  > $HADOOP_HOME/bin/hadoop fs -put /home/file.txt /
   user/input
6
7  # Step 3 : check results with ls cmd
8  > $HADOOP_HOME/bin/hadoop fs -ls /usr/input
```

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Retreiving Data from HDFS

```
1  # Step 1 : view data
2  > $HADOOP_HOME/bin/hadoop fs -cat /user/outputfile
3
4  # Step 2 : get data from hdfs filesystem to local
   # filesystem
5  > $HADOOP_HOME/bin/hadoop fs -get /user/output/ /
   home/hadoop_out
6
7  # Step 3 : check results with ls cmd
8  > $HADOOP_HOME/bin/hadoop fs -mkdir /usr/input
```

# Hadoop Framework: HDFS

## HDFS commands list

Commande name	Description
fs -help <cmd-name>	return cmd usage
fs -ls <path>	list <path> directory contents
fs -lsr <path>	ls ,recursively with sub dirs
fs -du <path>	show disk usage in bytes
fs -dus <path>	show disk usage in bytes and summary
fs -test [ezd] <path>	return 1 if path exists; has 0 length; or is a directory, otherwise 0
fs -cat <filename>	
fs -tail [-f] <filename>	

# Hadoop Framework: HDFS

## HDFS commands list

Commande name	Description
fs -mv <src><dest>	move file or directory within HDFS
fs -cp <src> <dest>	copy file or directory within HDFS
fs -rm <path>	remove file or directory within HDFS
fs -rmr <path>	rm recursively
fs -put <localSrc> <dest>	copy files or dirs from local FS to HDFS

# Hadoop Framework: HDFS

## HDFS commands list

Commande name	Description
fs -copyFromLocal <localSrc> <dest> fs -moveFromLocal <localSrc> <dest>	identical to put move file or dirs from local FS to HDFS
fs -get [-crc] <src> <localDest>	copy file or dirs from HDFS to local FS
fs -getmerge [-crc] <src> <localDest>	copy all files from HDFS and merge to a single file in FS
fs -copyToLocal <localSrc> <dest>	copy file or dirs from HDFS to local FS
fs -moveToLocal <localSrc> <dest>	move file or dirs from HDFS to local FS
fs -mkdir <path>	create directory in HDFS

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

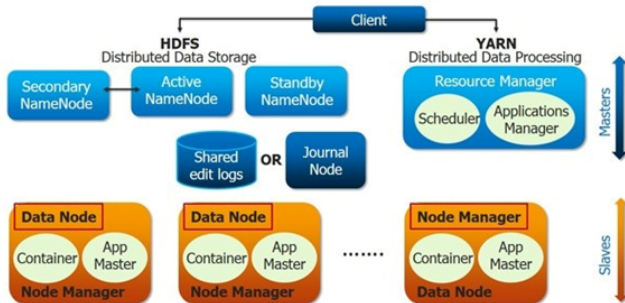


# Hadoop Framework

## YARN

prwatch.in

## Apache Hadoop 2.0 and YARN



# Hadoop Framework

## YARN

### Starting YARN

```
1 # Starting YARN services
2 > start-yarn.sh
```

### Shutting down YARN

```
1 # Stopping YARN services
2 > stop-yarn.sh
```

# Hadoop Web tools

## Hadoop Web Tools

Web tools on : `http://<hostname>:<port>`  
`<hostname>:<port>` (default localhost:50070) are defined in `hdfs-site.xml`  
Overview web page:

**Overview 'localhost:9000' (active)**

Started:	Tue Nov 26 11:52:59 CET 2019
Version:	2.7.7, rc1a0d84bd27cd79c3f3a7dd58202dc3ee1e53ac
Compiled:	2018-07-18T22:47Z by stevel from branch-2.7.7
Cluster ID:	CID:3c128403-a708-48e9-a733-f670fce0083b
Block Pool ID:	BP-1415240922-19.9.2.140-1574171722400

**Summary**

Security is off.  
Safemode is off.  
20 files and directories, 8 blocks — 26 total filesystem objects.  
Heap Memory used 57.92 MB of 245 MB Heap Memory. Max Heap Memory is 809 MB.  
Non-Heap Memory used 48.75 MB of 49.84 MB Committed Non-Heap Memory. Max Non-Heap Memory is 1 B.

Configured Capacity:	931.05 GB
DFS Used:	212 KB (0%)
Non DFS Used:	516.99 GB
DFS Remaining:	414.06 GB (44.47%)
Block Pool Used:	212 KB (0%)
DataNode usage% (Min/Median/Max,utilDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

# Hadoop Web tools

## Hadoop Web Tools

### File browser web page:

Browsing HDFS - Mozilla Firefox

localhost:50070/explorer.html#/

Hadoop Overview Resources Snapshot Startup Progress Utilities

### Browse Directory

/

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	graffesj	supergroup	0 B	11/19/2019, 2:59:00 PM	0	0 B	tmp
drwxr-xr-x	graffesj	supergroup	0 B	11/19/2019, 2:56:56 PM	0	0 B	user

Hadoop, 2016.

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn

### • MapReduce

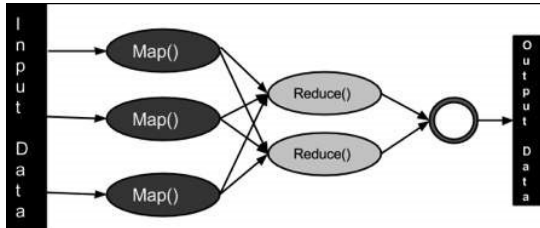
## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Hadoop Framework

## MapReduce Framework



MapReduce Framework

# Hadoop Framework

## MapReduce Framework

### MapReduce Algorithm :

- Programming model ;
- Two stages:
  - Map stage :
    - Mapper jobs;
    - data are processed in parallel by mapper jobs;
  - Reduce Stage :
    - Reducer jobs;
    - mapper output data are processed Reducer jobs;
    - Reducer jobs produce new set of output stored in HDFS.

# Hadoop Framework

## MapReduce Framework

Input and Output :

- Input : <key,value> pairs
  - key and values classes must implement Writable Interface;
  - key class have to implemente WritableComparable Interface;
- Job : (Input) → map → <k2,v2> → reduce → <k3,v3>(Output)

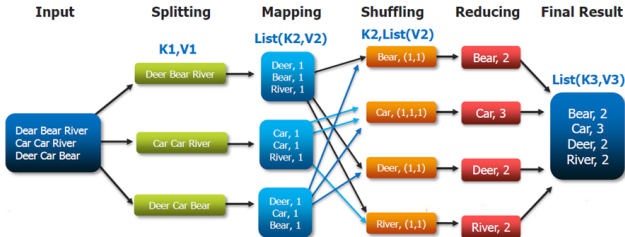
	Input	Output
Map	<k1,v1>	list(<k2,v2>)
Reduce	<k2,v2>	list(<k3,v3>)



# Hadoop Framework

## MapReduce Framework

### The Overall MapReduce Word Count Process



# Hadoop Framework

## Java exemple

### WordCount Java class

```
1  import org.apache.hadoop.*;
2  public class WordCount {
3      public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
4          public void map(Object key, Text value, Context context)
5              throws IOException, InterruptedException {
6              ... ;
7          }
8      }
9      public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
10         public void reduce(Text key, Iterable<IntWritable> values,Context context)
11             throws IOException, InterruptedException {
12             ... ;
13         }
14     }
15     public static void main(String[] args) throws Exception {
16         ...;
17     }
18 }
```

# Hadoop Framework

## Java exemple

### Word Count Mapper Java class

```
1 public class WordCount
2 {
3     public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
4     {
5         public void map(Object key, Text value, Context context)
6             throws IOException, InterruptedException
7         {
8             StringTokenizer itr = new StringTokenizer(value.toString());
9             while (itr.hasMoreTokens()) {
10                 word.set(itr.nextToken());
11                 context.write(word, one);
12             }
13         }
14     }
15 }
```

# Hadoop Framework

## Java exemple

### WordCount Reducer Java class

```
1 public class WordCount
2 {
3     public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>
4     {
5         private IntWritable result = new IntWritable();
6         void reduce(Text key, Iterable<IntWritable> values,Context context)
7             throws IOException, InterruptedException
8         {
9             int sum = 0;
10            for (IntWritable val : values) {
11                sum += val.get();
12            }
13            result.set(sum);
14            context.write(key, result);
15        }
16    }
17 }
```

# Hadoop Framework

## Java exemple

### Main test function

```
1 public class WordCount
2 {
3     public static void main(String[] args) throws Exception
4     {
5         Configuration conf = new Configuration();
6         Job job = Job.getInstance(conf, "word_count");
7         job.setJarByClass(WordCount.class);
8         job.setMapperClass(TokenizerMapper.class);
9         job.setCombinerClass(IntSumReducer.class);
10        job.setReducerClass(IntSumReducer.class);
11        job.setOutputKeyClass(Text.class);
12        job.setOutputValueClass(IntWritable.class);
13        FileInputFormat.addInputPath(job, new Path(args[0]));
14        FileOutputFormat.setOutputPath(job, new Path(args[1]));
15        System.exit(job.waitForCompletion(true) ? 0 : 1);
16    }
17 }
```

# Hadoop Framework

## Java exemple

Suppose we have two test files file01 and file02 in current directory

### Prepare Test Data

```
1 // Two test file file01 file02 in current directory
2 $ ls
3   file01
4   file02
5 $ hdfs dfs -put file01 /user/gratienj/input
6 $ hdfs dfs -cat /user/gratienj/input/file01
7   Hello World Bye World
8 $ hdfs dfs -put file02 /user/gratienj/input
9 $ hdfs dfs -cat /user/gratienj/input/file02
10  Hello Hadoop Goodbye Hadoop
```

# Hadoop Framework

## Java exemple

Suppose the Java Project is compiled and generates the jar file  
BigDataTP1.jar

### Run application

```
1 $ hadoop jar BigDataTP1.jar hadoop.WordCount /user/gratienj/input /user/gratienj/output
```

### Check results

```
1 $ hdfs dfs -cat /user/gratienj/output/part-r-00000
2 Bye 1
3 Goodbye 1
4 Hadoop 2
5 Hello 2
6 World 2
```

# Hadoop Framework

## Python example

### Python example with Hadoop Streaming

#### Mapper python script

```
1  #!/usr/bin/env python
2  """mapper.py"""
3  import sys
4  # input comes from STDIN (standard input)
5  for line in sys.stdin:
6      line = line.strip()
7      words = line.split()
8      for word in words:
9          print(' %s\t%s' % (word, 1))
```



# Hadoop Framework

## Python example

### Reducer python script Part 1

```
1 from operator import itemgetter
2 import sys
3 current_word = None
4 current_count = 0
5 word = None
```

# Hadoop Framework

## Python example

### Reducer python script Part 2

```
1 for line in sys.stdin: # input comes from STDIN
2     line = line.strip()
3     word, count = line.split('\t', 1)
4     try:
5         count = int(count)
6     except ValueError:
7         continue
8     if current_word == word:
9         current_count += count
10    else:
11        if current_word:
12            print('%s\t%s' % (current_word, current_count)) # write result to STDOUT
13            current_count = count
14            current_word = word
15    if current_word == word: # do not forget to output the last word if needed!
16        print('%s\t%s' % (current_word, current_count))
```

# Hadoop Framework

## Python example

### Python example with Hadoop Streaming : Part 1

#### Copy test files on HDFS

```
1 $ hdfs dfs -copyFromLocal /home/gratienj/test/books /user/gratienj/input/books
```

#### Run application

```
1 $ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.7.7.jar \  
2 -file /home/hduser/mapper.py -mapper /home/hduser/mapper.py \  
3 -file /home/hduser/reducer.py -reducer /home/hduser/reducer.py \  
4 -input /user/gratienj/input/books/* -output /user/gratienj/books-output
```

# Hadoop Framework

## Python example

### Python example with Hadoop Streaming : Part 2

#### Check results

```
1 $ hdfs dfs -ls /user/gratienj/books-ouput
2 Found 1 items
3 /user/gratienj/books-output/part-00000
4 $ hdfs dfs -cat /user/gratienj/books-output/part-00000
```

# Hadoop Web tools

## Ambari Server Tools

Ambari Server : tools to manage and monitor applications for Apache Hadoop

Web page : `http://<ambari-server-hostname>:8080`

The screenshot shows the Ambari Server web interface in a Mozilla Firefox browser. The page title is 'Overview localhost:9000 (active)'. The navigation bar includes 'Hadoop', 'Overview', 'Datanodes', 'Datanode Volume Failures', 'Snapshot', 'Startup Progress', and 'Utilities'. The 'Overview' tab is selected.

**Overview 'localhost:9000' (active)**

Started:	Tue Nov 26 11:52:59 CET 2019
Version:	2.7.7, rc1a0d84b027cd79c3d3a7dd58202dc3ee1e53ac
Compiled:	2018-07-18T22:47Z by stavel from branch-2.7.7
Cluster ID:	CID-3c128403-a708-48e9-a733-f670fce0883b
Block Pool ID:	BP-1415240922-19.9.2.140-1574171722400

**Summary**

Security is off.  
Safemode is off.  
20 Files and directories, 8 blocks - 26 total filesystem objects.  
Heap Memory used 57.92 MB of 245 MB Heap Memory. Max Heap Memory is 809 MB.  
Non-Heap Memory used 40.75 MB of 49.84 MB Committed Non-Heap Memory. Max Non-Heap Memory is 1 B.

Configured Capacity:	931.05 GB
DFS Used:	212 KB (0%)
Non DFS Used:	516.99 GB
DFS Remaining:	414.06 GB (44.47%)
Block Pool Used:	212 KB (0%)
DataNodes usage% (Min/Median/Max/StdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- **Introduction**
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Spark Framework

## Introduction to Spark

- Spark : Big Data framework for data processing
- History
  - 2009 : AMPLab, UC Berkeley University
  - 2010 : Open source as an Apache project
- Complete and Unified framework
  - Hadoop (MapReduce)
  - Storm (Streaming)
  - Languages : Java, Scala, Python
  - SQL

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

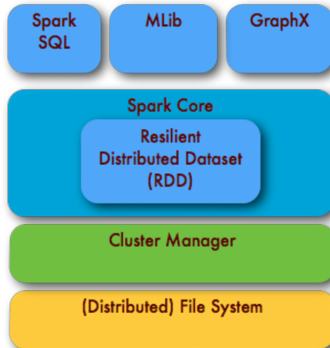
- Introduction
- **Architecture and Ecosystem**
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask



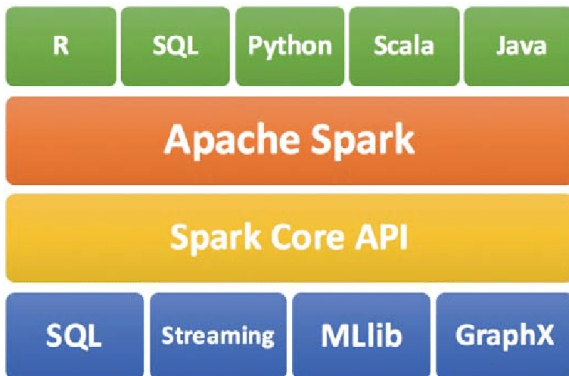
# Spark Framework

## Apache Spark Architecture



# Spark Framework

Apache Spark Ecosystem



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- **Spark Modules : Core, SQL, Mlib, Streaming and GraphX**
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Spark Framework

Spark Core : Spark configuration

Spark Cluster Configurations :

- Local mode
- Cluster mode
- Client mode

Spark parallel concepts:

- multiple executors (private JVM)
- multiple cores per executor

# Spark Framework

Spark Core : Spark configuration

## Configuring a SparkContext

```
1 import pyspark
2 from pyspark import SparkConf
3 sc_conf = SparkConf()
4 sc_conf.setAppName(app_name)
5 sc_conf.setMaster('local[*]')
6 sc_conf.set('spark.executor.memory', '4g')
7 sc_conf.set('spark.executor.cores', nb_cores)
8 sc_conf.set('spark.driver.memory', '16G')
9 sc_conf.set('spark.cores.max', '32')
10 sc_conf.set('spark.driver.maxResultSize', '10G')
11 sc_conf.set('spark.logConf', True)
```

# Spark Framework

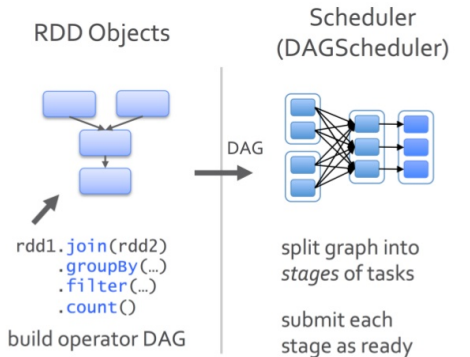
Spark Core : Spark context

## Create a SparkContext

```
1 import pyspark
2 from pyspark import SparkContext
3 sc =SparkContext()
```

# Spark Framework

## Spark Core : Data concepts



<https://databricks.training/s3.amazonaws.com/slides/advanced-spark/training.pdf>

# Spark Framework

## Spark Core : Data concepts

Spark Data concepts :

- **RDD** : Resilient Distributed Data, list of <key,value>
- **Transformations** : apply lambda to creating new RDDs
- **DAG** : pipeline of transformation
- **Actions** : operations on the RDD producing results
- **Scheduler** : perform actions on DAG
- **Stage** : parallel operations
- **Pipeline** : sequence of stages



# Spark Framework

Spark Core : RDD

## Create a Spark RDD

```
1 import pyspark
2 from pyspark import SparkContext
3 sc = SparkContext()
4
5 nums= sc.parallelize([1,2,3,4])
6
7 nums.take(1)
```

## Output

```
1 [1]
```

# Spark Framework

Spark Core : RDD Transformation and Actions

## Spark RDD transformations and Actions

```
1 sc = SparkContext()
2 nums= sc.parallelize([1,2,3,4])
3 squared = nums.map(lambda x: x*x).collect()
4 for num in squared:
5     print('%i_' % (num))
```

## Output

```
1 1
2 4
3 9
4 16
```

# Spark Framework

## Spark Core : RDD Transformations and Actions

Transformation :

- apply lambda function to RDD
- create a new RDD
- lazy evaluation
- create a DAG

# Spark Framework

## Spark Core : RDD Transformations and Actions

Examples :

Commande name	Description
map()	apply to each RDD line
flatMap()	apply to all RDD elements
mapPartition	apply per partition
filter()	apply to a selection of lines
groupByKey()	create new set of (key,value)
groupByKey()	
reduceByKey()	
sample()	selection of lines
union()	fusion of two RDDs
join()	union without duplicate keys

# Spark Framework

## Spark Core : RDD Transformations and Actions

### Actions :

- get results on a pipeline of transformations
- perform all the transformation
- real evaluation

### Examples :

Commande name	Description
getNumPartition() reduce() collect()	apply lambda to all elements create a collection
count() max(), min() sum()	count elements stats

# Spark Framework

Spark SQL : DataFrame

## Unified Data Abstraction



Image credit: <http://barrymieny.deviantart.com/>

 DATABRICKS

# Spark Framework

## Spark Core : RDD

### Create a Spark SQL context

```
1 from pyspark.sql import Row
2 from pyspark.sql import SQLContext
3 sqlContext = SQLContext(sc)
```

### Create a DataFrame

```
1 list_p=[('John',19),('Smith',29),('Adam',35)]
2 rdd = sc.parallelize(list_p)
3 ppl_rdd=rdd.map(lambda x: Row(name=x[0], age=int(x
    [1])))
4 ppl_df_rdd = sqlContext.createDataFrame(ppl_rdd)
```

# Spark Framework

Spark SQL : DataFrame

## Print DataFrame Schema

```
1 DF_ppl.printSchema()  
2 root  
3 |-- age: long (nullable = true)  
4 |-- name: string (nullable = true)
```



# Spark Framework

## Spark SQL : DataFrame

### Print DataFrame Schema

```
1 df = sqlContext.read.csv(SparkFiles.get("adult_data.csv"), header=True, inferSchema= True)
2 df_string.printSchema()
3 root
4   |-- age: string (nullable = true)
5   |-- workclass: string (nullable = true)
6   |-- fnlwgt: string (nullable = true)
7   |-- education: string (nullable = true)
8   |-- education_num: string (nullable = true)
9   |-- marital: string (nullable = true)
10  |-- occupation: string (nullable = true)
11  |-- relationship: string (nullable = true)
12  |-- race: string (nullable = true)
13  |-- sex: string (nullable = true)
14  |-- capital_gain: string (nullable = true)
15  |-- capital_loss: string (nullable = true)
16  |-- hours_week: string (nullable = true)
17  |-- native_country: string (nullable = true)
18  |-- label: string (nullable = true)
```

# Spark Framework

Spark SQL : DataFrame

## Select columns

```
1 df.select('age', 'fnlwgt')  
   .show(5)
```

## Select columns

```
1 +---+-----+  
2 |age|fnlwgt|  
3 +---+-----+  
4 | 39| 77516|  
5 | 50| 83311|  
6 | 38|215646|  
7 | 53|234721|  
8 | 28|338409|  
9 +---+-----+  
10 only showing top 5 rows
```

# Spark Framework

Spark SQL : DataFrame

## Select columns

```
1 df.groupBy("education").
2   count().sort("count",
3   ascending=True).show()
```

## Select columns

```
1 +-----+-----+
2 | education|count|
3 +-----+-----+
4 |  Preschool|   51|
5 |    1st-4th|  168|
6 |    5th-6th|  333|
7 | Doctorate|  413|
8 |     12th|  433|
9 |      9th|  514|
10 | Prof-school| 576|
11 |    7th-8th| 646|
12 |     10th|  933|
13 | Assoc-acdm|1067|
14 |     11th|1175|
15 | Assoc-voc|1382|
16 |   Masters|1723|
17 | Bachelors|5355|
18 |Some-college|7291|
19 |    HS-grad|10501|
20 +-----+-----+
```

# Spark Framework

## Spark SQL : DataFrame

Describe data: describe() functions give a summary of statistics :

- count,
- mean,min,max
- standarddeviation

### Describe

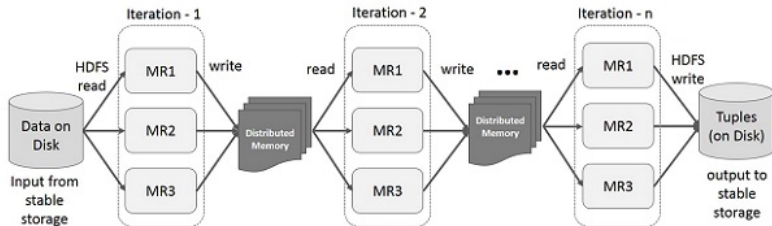
```
1 df.describe('capital_gain'  
    '').show()
```

### Describe

```
1 +-----+-----+  
2 |summary|    capital_gain|  
3 +-----+-----+  
4 |  count|           32561|  
5 |   mean|1077.6488437087312|  
6 | stddev| 7385.292084840354|  
7 |   min|              0|  
8 |   max|          99999|  
9 +-----+-----+
```

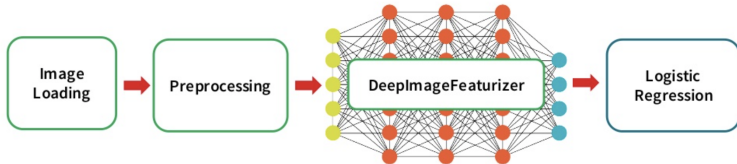
# Spark Framework

## Spark Mlib



# Spark Framework

## Spark Mlib



## MLlib Pipeline

# Spark Framework

## Spark Mlib

Mlib provides tools for Machine learning

- set of classifier and regression algorithms
- create models from Spark Dataframe
- set of tools to evaluate the predicting models
- concept of pipeline to process Data

# Spark Framework

## Spark Mlib Pipeline

Spark Pipeline : a sequence of stages (Transformer, Estimator)

- String Indexer : convert Categorical Data to numerics
- Standard Scaler on Continuous Values
- VectorAssembler : features must be a dense vector



# Spark Framework

Spark Mlib : Data processing

## StringIndexer

```
1 from pyspark.ml.feature import StringIndexer
2 df_rdd = ...
3 # Create String Indexer to convert "cat_key" to "
  ind_key"
4 string_indexer = StringIndexer(inputCol="cat_key",
  outputCol="encoded_key")
5 # Create model
6 model = string_indexer.fit(df_rdd)
7 # Transform RDD
8 encoded_df_rdd = model.transform(df_rdd)
```

# Spark Framework

Spark Mlib : Data processing

## OneHotEncoder

```
1 from pyspark.ml.feature import OneHotEncoder
2 encoder = OneHotEncoder(dropLast=False, inputCol="
    encoded_key", outputCol="vec_key")
3 vec_df_rdd = encoder.transform(encoded_df_rdd)
```

## VectorAssembler

```
1 from pyspark.ml.feature import VectorAssembler
2 assembler = VectorAssembler(inputCols=[key1, key2,
    key3], outputCol="features")
3 ass_df_rdd = assembler.transform(df_rdd)
```

# Spark Framework

Spark Mlib : Data processing

## Pipeline

```
1 from pyspark.ml import Pipeline
2 # DEFINE LIST OF STAGES
3 stages = [[label_indexer], [cat_key_indexer, encoder
4             ], [assembler]]
5
6 # DEFINE PIPELINE
7 pipeline = Pipeline(stages=stages)
8
9 # APPLY PIPELINE
10 pipelineModel = pipeline.fit(df_rdd)
11 model_df_rdd = pipelineModel.transform(rdd_df)
```

# Spark Framework

## Spark ML : ML pipeline Part 1

### create DataFrame

```
1 rdd.map(lambda x: (x["newlabel"], DenseVector(x["  
    features"])))  
2 sqlContext.createDataFrame(input_data, ["label", "  
    features"])
```

### Split data

```
1 randomSplit([.8, .2], seed=1234)
```

# Spark Framework

## Spark ML : ML pipeline Part 2

### Train model

```
1 lr=LogisticRegression(labelCol="label",featuresCol=
    "features",maxIter=10, regParam=0.3)
2 lr.fit()
```

### Make prediction

```
1 lr.transform()
```

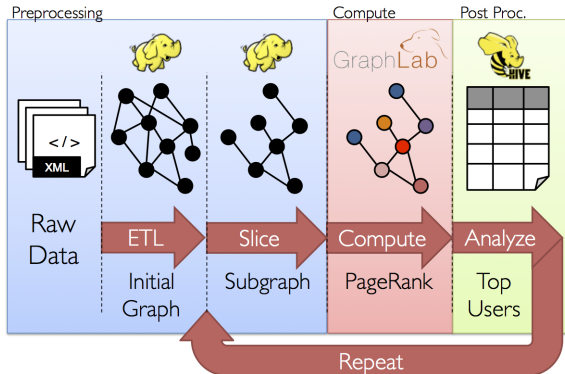
# Spark Framework

## Spark Streaming



# Spark Framework

## Spark GraphX



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- **Spark : CLI**

## 4 Dask



# Spark Ecosystem

## Spark Cluster : Command Line Set Up

### Spark Local Mode

```
1  #!/bin/sh
2  PYSPARK_DRIVER_PYTHON=python
3  PYSPARK_PYTHON=python
4  spark-submit \
5  --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=/python_root/bin/python \
6  --master local[*] \
7  --deploy-mode client \
8  myscript.py
```

# Spark Ecosystem

## Spark Cluster : Command Line Set Up

### Spark Client Mode

```
1  #!/bin/sh
2  spark-submit \
3  --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=/python_root/bin/python \
4  --master yarn \
5  --deploy-mode client \
6  script.py
```

# Spark Ecosystem

## Spark Cluster : Command Line Set Up

### Spark Cluster Mode

```
1 PYSPARK_DRIVER_PYTHON=/python_root/bin/python \  
2 PYSPARK_PYTHON=./environment/bin/python \  
3 spark-submit \  
4 --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=./environment/bin/python \  
5 --master yarn \  
6 --deploy-mode client \  
7 --principal gratienj@IFP.FR \  
8 --keytab /tmp/krb5cc_1109 \  
9 --archives environment.tar.gz#environment \  
10 script.py
```

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Dask Framework

## Introduction to Dask

- Dask : Open source python framework for data processing
- developped with community projects like : Numpy, Pandas, and Scikit-Learn
- supported by: Anaconda, CapitalOne, NSF, Nvidia,...
- High-level collections:
  - Array, Bag, and DataFrame collections
  - mimic NumPy, lists, and Pandas
  - operate datasets out of core memory
- Low-Level schedulers :
  - dynamic task schedulers
  - execute task graphs in parallel

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

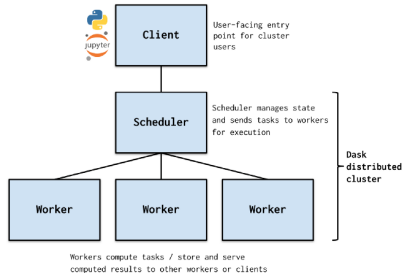
## 4 Dask

# Dask Framework

## Dask Architecture

Dask architecture:

- Dask Cluster
- Dask Scheduler
- Dask collections



# Dask Framework

## Dask Cluster

Various Dask cluster types:

- Hadoop/Spark clusters running YARN
- HPC clusters running job managers like SLURM, SGE, PBS, LSF, or others common in academic and scientific labs
- Kubernetes clusters

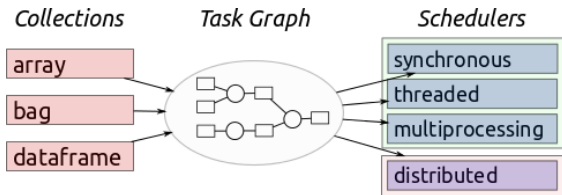


# Dask Framework

## Dask Scheduler

### Dask Scheduler:

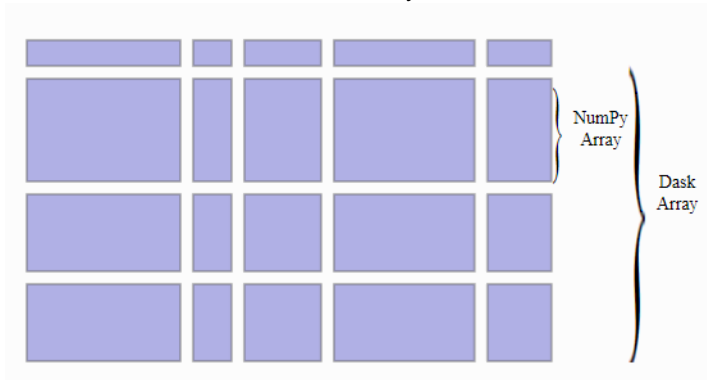
- Single machine scheduler  
: Optimized for larger-than-memory use. Simple, easy and cheap to use, but does not scale as it only runs on a single machine.
- Distributed scheduler :  
More sophisticated, fully asynchronous



# Dask Framework

## Dask collections

### Dask Array



## Dask DataFrame



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Dask framework

## Dask installation

### Installation

```
1 conda install dask
```

OR

### Installation

```
1 pip install dask[complete]
```

# Dask framework

## Dask starting cluster

### Lauching Dask cluster

```
1 from dask.distributed import LocalCluster, Client
2 cluster = LocalCluster()
3 client = Client(cluster)
```

Dashboard usually on `http://localhost:8787/status`

### Dashboard

```
1 #To see where the port of the dashboard is, use this command
2 print(client.scheduler_info()['services'])
3 # {'dashboard': 8787} --> means you can access it at localhost:8787
```

# Dask framework

## Dask Collections

### Dask Bag

```
1 import dask.bag as db
2 b = db.from_sequence([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], npartitions=2)
```

### Dask Array

```
1 import dask.array as da
2 x = da.random.random((10000, 10000), chunks=(1000, 1000))
```

### Dask DataFrame

```
1 from dask import datasets
2 import dask.dataframe as dd
3 df = datasets.timeseries()
```

# Dask framework

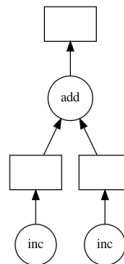
## Dask Distributed

### Dask Delayed function

```
1 from dask import delayed
2 @delayed
3 def inc(x):
4     return x + 1
5 @delayed
6 def add(x, y):
7     return x + y
```

### Dask Lazy evaluation

```
1 x = inc(15)
2 y = inc(30)
3 total = add(x, y)
4
5 #visualize DAG
6 total.visualize()
7
8 # execute all tasks
```





# Dask framework

## Dask Distributed

### Dask Future

```
1 from dask import delayed
2 def inc(x):
3     return x + 1
4 def dec(x):
5     return x - 1
6 def add(x, y):
7     return x + y
```

### Dask Lazy evaluation

```
1 from dask.distributed import Client
2 c = Client(n_workers=4)
3
4 x = c.submit(inc, 1)
5 y = c.submit(dec, 2)
6 total = c.submit(add, x, y)
```

# Dask framework

## Dask Distributed

### Dask evaluation

```
1 # execute all tasks
2 total.compute()
```

### Dask progress

```
1 from dask.distributed import progress
2 # to show progress bar
3 progress(f)
```

# Dask framework

## Dask Distributed

### Dask get results

```
1 # get result.  
2 c.gather (f)
```

### Dask persist

```
1 total.persist ()
```

# Dask framework

## Dask Distributed

### Dask asynchronism

```
1 from dask.distributed import as_completed
2 def func(x):
3     ...
4     return y
5 futures = [c.submit(func, x) for x in range(n)]
6 iterator = as_completed(futures)
7 for res in iterator:
8     print("RES_Y_:", res.result())
```

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# Dask ecosystem

## Ecosystem overview

Dask ecosystem overview :

- Dask tutorial :  
`https://github.com/dask/dask-tutorial.git`
- Collection : Bag, Array, DataFrame
- Data Storage : CSV, HDF5, ...
- Machine Learning : Scikit-learn, XGBoost, ...
- Cluster : Local, SSH, YARN, ...

# Dask Ecosystem

## Dask Bag

Dask bag

```
1 import dask.bag as db
2 b = db.from_sequence([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], npartitions=2)
```

Dask bag

```
1 import os
2 b = db.read_text(os.path.join('data', 'accounts.*.json.gz'))
```

Dask bag

```
1 import json
2 js = lines.map(json.loads)
```

# Dask Ecosystem

## Dask Array

### Dask bag

```
1 import h5py
2 import os
3 f = h5py.File(os.path.join('data', 'random.hdf5'), mode='r')
4 dset = f['x']
```

### Dask bag : lazy creation

```
1 import dask.array as da
2 x = da.from_array(dset, chunks=(1_000_000,))
```

### Dask bag : Numpy lazy API

```
1 result = x.sum()
2 print(result)
```

### Dask bag : evaluation



# Dask Ecosystem

## Dask DataFrame

### Dask DataFrame

```
1 import os
2 import dask
3 filename = os.path.join('data', 'accounts.*.csv')
```

### Dask DataFrame : lazy creation

```
1 import dask.dataframe as dd
2 df = dd.read_csv(filename)
3
4 df = dd.read_csv(os.path.join('data', 'nycflights', '*.csv'),
5                  parse_dates={'Date': [0, 1, 2]})
```

### Dask DataFrame : lazy API

```
1 df.DepDelay.max().visualize()
```

# Dask Ecosystem

## Dask Cluster

### Dask Local Cluster

```
1 from dask.distributed import Client, LocalCluster
2 cluster = LocalCluster()
3 client = Client(cluster)
```

### Dask YarnCluster

```
1 from dask_yarn import YarnCluster
2 from dask.distributed import Client
3
4 # Create a cluster where each worker has two cores and eight GiB of memory
5 cluster = YarnCluster(environment='environment.tar.gz',
6                       worker_vcores=2,
7                       worker_memory="8GiB")
8
9 # Scale out to ten such workers
10 cluster.scale(10)
11
12 # Connect to the cluster
13 client = Client(cluster)
```

# Dask Ecosystem

## Dask Cluster Command line SetUp

### Dask Scheduler and Workers SetUp

```
1 $ dask-scheduler
2 Scheduler at: tcp://192.0.0.100:8786
3
4 $ dask-worker tcp://192.0.0.100:8786
5 Start worker at: tcp://192.0.0.1:12345
6 Registered to: tcp://192.0.0.100:8786
7
8 $ dask-worker tcp://192.0.0.100:8786
9 Start worker at: tcp://192.0.0.2:40483
10 Registered to: tcp://192.0.0.100:8786
```

### Dask Client SetUp

```
1 from distributed import Client
2 client = Client('192.0.0.100:8786')
```

# TPs

TPs require docker usage.

A Docker Quick Sheet can be found in :

`https:`

`//github.com/jgratien/BigDataHadoopSparkDaskCourse/  
blob/main/TPs/doc/Docker.md`

- TP 0 : Hadoop Installation
- TP 1 : WorldCount
- TP 2 : DataBase request
- TP 3 : Spark Installation
- TP 4 : Spark Compute PI
- TP 5 : Spark Image Processing
- TP 6 : Spark ML
- TP 7 : Dask

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## TP0 : Manual Hadoop Installation

Hadoop prerequisites : Check ssh service and Java installation

### Check SSH

```
1 > ssh localhost
```

In case of error ssh: connect to host localhost port 22: Connection refused

### Installation SSH

```
1 > sudo apt remove openssh-server
2 > sudo apt install openssh-server
3 > sudo service ssh start
```

# TPs

## TP0 : Manual Hadoop Installation

Hadoop prerequisites : Check ssh service and Java installation

### Check SSH

```
1 > ssh localhost
```

### Add keys

```
1 > ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
2 > cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
3 > chmod 0600 ~/.ssh/authorized_keys
```

# TPs

## TP0 : Manual Hadoop Installation

Hadoop prerequisites : Check ssh service and Java installation

### Check java

```
1 > java -version
```

### Installation Java Jdk

```
1 > sudo apt-get install openjdk-8-jdk
```

### Check java version

```
1 > java -version
2 openjdk version "1.8.0_275"
3 OpenJDK Runtime Environment (build 1.8.0_275-8u275-b01-0ubuntu1~20.04-b01)
4 OpenJDK 64-Bit Server VM (build 25.275-b01, mixed mode)
```



# TPs

## TP0 : Manual Hadoop Installation

### Hadoop Installation: `hadoop-3.3.0.tar.gz`

#### Installation

```
1 > cd /home/hduser
2 > mkdir local ; cd local
3 > wget https://downloads.apache.org/hadoop/common/
   hadoop-3.3.0/hadoop-3.3.0.tar.gz
4 > tar xvfz hadoop-3.3.0.tar.gz
5 > mv hadoop-3.3.0 hadoop
6 > chown -R hduser:hadoop hadoop
```

# TPs

## TP0 : Installation Hadoop

### Env parameter settings

```
1  # Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
2  export JAVA_HOME=/usr/local/Java/1.8.0-xxx
3
4  # Set Hadoop-related environment variables
5  export HADOOP_HOME=/home/hduser/local/hadoop
6
7  export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
8  export HADOOP_MAPRED_HOME=${HADOOP_HOME}
9  export HADOOP_COMMON_HOME=${HADOOP_HOME}
10 export HADOOP_HDFS_HOME=${HADOOP_HOME}
11 export YARN_HOME=${HADOOP_HOME}
12
13 export PATH=${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin:$PATH
```

# TPs

## TP0 : Installation Hadoop

### Configuration files settings

```
1  # CREATE HADOOP TMP DIR
2  > mkdir -p /home/hduser/app/hadoop/tmp
3  > chown hduser:hadoop /home/hduser/app/hadoop/tmp
4  > chmod 750 /home/hduser/app/hadoop/tmp
5
6  # CREATE HDFS WORKINGDIR TO MNG HDFS File System
7  > mkdir -p /home/hduser/var/local/hadoop/hdfs/data
8  > chmod -R 777 /home/hduser/var/local/hadoop/hdfs
```

# TPs

## TP0 : Installation Hadoop

Configuration files in /home/hduser/local/hadoop/etc/hadoop

### hadoop-env.sh modification

```
1 JAVA_HOME="true_java_JOME_path"  
2 export JAVA_HOME=${JAVA_HOME}
```

### core-site.xml settings

```
1 <property>  
2     <name>hadoop.tmp.dir</name>  
3     <value>/home/hduser/app/hadoop/tmp</value>  
4     <description>A base for other temporary directories.</description>  
5 </property>  
6 <property>  
7     <name>fs.default.name</name>  
8     <value>hdfs://localhost:9000</value>  
9     <description>The name of the default file system.</description>  
10 </property>
```

# TPs

## TP0 : Installation Hadoop

Configuration files in `/home/hduser/local/hadoop/etc/hadoop`

### hdfs-site.xml settings

```
1 <property>
2   <name>dfs.data.dir</name>
3   <value>/home/hduser/var/local/hadoop/hdfs/data</value>
4   <final>true</final>
5 </property>
6
7 <property>
8   <name>dfs.replication</name>
9   <value>1</value>
10 </property>
```

# TPs

## TP0 : Installation Hadoop

Configuration files in /home/hduser/local/hadoop/etc/hadoop  
Copy mapred-site.xml.template mapred-site.xml

### mapred-site.xml settings

```
1 <property>
2     <name>mapred.job.tracker</name>
3     <value>localhost:9001</value>
4 </property>
```

# TPs

## TP0 : Installation Hadoop

Configuration files in /home/hduser/local/hadoop/etc/hadoop

### yarn-site.xml settings

```

1 <configuration>
2   <!-- Site specific YARN configuration properties -->
3   <property>
4     <name>yarn.nodemanager.aux-services</name>
5     <value>mapreduce_shuffle</value>
6   </property>
7   <property>
8     <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
9     <value>org.apache.hadoop.mapred.ShuffleHandler</value>
10  </property>
11 </configuration>

```

# TPs

## TP0 : Installation Hadoop

### Launch all services

```
1 > $HADOOP_HOME/sbin/start-hdfs.sh
2 > $HADOOP_HOME/sbin/start-yarn.sh
```

### Check lauched services

```
1 > jps
2
3 26867 DataNode
4 28228 Jps
5 27285 ResourceManager
6 26695 NameNode
7 27082 SecondaryNameNode
8 27420 NodeManager
```



# TPs

## TP0 : Hadoop cluster set up using docker

- Check Docker installation

### Docker : Set up

```
1 > docker --version
2 > docker-compose --version
```

- Test of NGINX server

### Docker : Check using nginx

```
1 > docker run -d -p 80:80 --name myserver nginx
```

Visit <http://localhost> and check nginx server homepage

# TPs

## TP0 : Hadoop cluster set up using docker

- Check Docker installation

### Docker : Set up

```
1 > cd TP0/Docker
2 > git clone https://github.com/big-data-europe/docker-hadoop.git
3 > cd docker-hadoop
4 > docker-compose up -d
5 > docker ps
```

- Test the cluster installation on `http://localhost:9870`
- Close the Hadoop cluster

### Docker : Safely close the hadoop cluster

```
1 > docker-compose down
```

Visit `http://localhost` and check nginx server homepage

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## TP1 : MapReduce with Hadoop

### Project MapReduce :

/home/hduser/BigDataHadoopSpark/TPs/TP1/MapReduce

Two projects, A java Project and a python project

```
1 MapReduce|
2     |--pom.xml
3     |--bin|
4     |--python|--mapper.py
5     |         |--reduce.py
6     |--src|--hadoop|--WordCount.java
7     |--target|
8     |--test|wordcount|--file01
9                     |--file02
10                    |books|--b0
11                    |--b1
```

# TPs

## TP1 : MapReduce with Hadoop

Java project:

- create directory in hdfs /user/hduser/**input**
- copy the files of MapReduce/test/wordcount in /user/hduser/**input**
- generate Java project BigDataTP1

```
1 cd BigDataHadoopSpark/TPs/TP1/MapReduce
2 mvn package
```

- apply Java WordCount application
- check results

# TPs

## TP1 : MapReduce with Hadoop

### Python project

- create directory in hdfs `/user/hduser/input/book`
- copy the files of `MapReduce/test/book` in `/user/hduser/input`
- apply Python WordCount application
- check results

# TPs

## TP1 : WorldCount and MapReduce using the containerized Hadoop cluster

Remind : Docker command Quick Sheet :

BigDataHadoopSparkDaskCourse/TPs/doc/Docker.md

### Docker : exec running container named namenode

```
1 > cd TPs/TP1/MapReduce
2 > docker exec -it namenode bash
```

### Docker : copy file in a container

```
1 > cd TPs/TP1/MapReduce
2 > docker cp test/wordcount/file01 namenode:file01
3 > docker cp test/wordcount/file02 namenode:file02
4 > docker cp jars/hadoop-mapreduce-examples-2.7.1-sources.jar namenode:hadoop-mapreduce-examples-2.7.1-sources.jar
```

# TPs

TP1 : WorldCount and MapReduce using the containerized Hadoop cluster

## Docker :Realize WordCount TP within the namenode container

```
1  /$ mkdir input
2  /$ mv file01 file02 input/.
3  /$ hadoop fs -mkdir -p input
4  /$ hdfs dfs -put ./input/* input
5  /$ hadoop jar hadoop-mapreduce-examples-2.7.1-sources.jar org.apache.hadoop.examples.WordCount input
    output
```

## Docker :Check results

```
1  /$ hdfs dfs -cat output/part-r-00000
```



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## TP2 : Manual Spark Framework Installation

### Spark Installation: spark-3.0.1-bin-hadoop3.2.tgz

#### Installation

```

1 > cd /home/hduser
2 > mkdir local ; cd local
3 > wget https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz
4 > tar xvfz spark-3.0.1-bin-hadoop3.2.tar.gz
5 > mv spark-3.0.1-bin-hadoop3.2 spark
6 > export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
7 > export SPARK_HOME=/home/hduser/local/spark
8 > export PATH=$SPARK_HOME/bin:$PATH
9 > export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH

```

# TPs

## TP2 : PySpark Installation

### Installation

```
1 > cd /home/hduser
2 > export SPARK_HOME=/home/hduser/local/spark
3 > export PATH=$SPARK_HOME/bin:$PATH
4 > export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
5 > export PYSARK_PYTHON="path_to_python"
6 > pip install pyspark
7 > pip install findspark
8 > sbin/start-master.sh
9 > sbin/start-slave.sh spark://localhost:7077
```

## TP2 : PySpark Installation

```

1 > spark-shell
2
3 Setting default log level to "WARN".
4 To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
5 Spark context Web UI available at http://localhost:4040
6 Spark context available as 'sc' (master = local[*], app id = local-1578948405576).
7 Spark session available as 'spark'.
8 Welcome to
9
10      _ _ _ _ _
11     / _/_ _ _ _/_/_
12    _ \ \_ _ \/_ \'_/_/'_/_
13   ___/_/_/_/_/_/_/_/_/_\___version_3.0.0-preview2
14   _____/_/
15
16 Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_92)
17 Type in expressions to have them evaluated.
18 Type :help for more information.
19 scala>
```



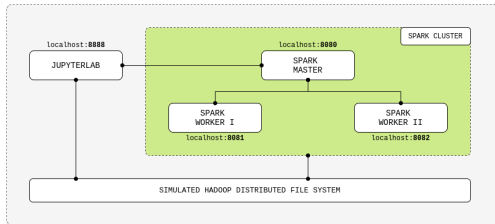
## TP2 : PySpark Installation

```

1 > pyspark
2
3 Python 2.7.5 (default, Apr 11 2018, 07:36:10)
4 [GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
5 Type "help", "copyright", "credits" or "license" for more information.
6 Setting default log level to "WARN".
7 To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
8 /work/irlin355_1/gratienj/BigData/local/spark/spark-3.0.0-preview2-bin-hadoop2.7/python/pyspark/
   context.py:219: DeprecationWarning: Support for Python 2 and Python 3 prior to version 3.6 is
   deprecated as of Spark 3.0. See also the plan for dropping Python 2 support at https://spark.
   apache.org/news/plan-for-dropping-python-2-support.html.
9     DeprecationWarning)
10 Welcome to
11
12      ____
13    _/ __/_  ___/_/___/_/___
14   _\ \/_  \/_  \/_  \/_  \/_
15  _/_/_/_/_/_/_/_/_/_/_/_/_version_3.0.0-preview2
16  _/_/_/_/_/_/_/_/_/_/_/_/_
17
18 Using_Python_version_2.7.5_(default,_Apr_11_2018_07:36:10)
   SparkSession.available_as_'spark'.
```

# TPs

## TP2 : Spark Framework Installation with docker



### Installation

- 1 > cd TP2/Docker
- 2 > git clone https://github.com/cluster-apps-on-docker/spark-standalone-cluster-on-docker.git
- 3 > curl -LO https://raw.githubusercontent.com/cluster-apps-on-docker/spark-standalone-cluster-on-docker/master/docker-compose.yml
- 4 > docker-compose up

# TPs

TP3 : Test0 Test1 Test2

Test0 :

- create spark context
- create liste of integer
- partition list with spark
- print num of partions

Test1 :

- compute square of integer list
- print square list

Test2 :

- compute PI

# TPs

TP3 : Test0 Test1 Test2

Realize Test0, Test 1 and Test2 with the JupiterLab Nodebook

## Deploy Spark Cluster

```
1 > cd TP2/Docker
2 > docker-compose up
```

Connect to JupiterLab NoteBook at <http://localhost:8888>



# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## TP4 : Spark ML, Data processing

### Test0 :

- load `TPs/data/iris.csv` file in Panda DataFrame
- create Spark DataFrame
- show 5 first lines
- select two columns
- print some statistics on Spark Data frame

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## TP5 : Spark ML, Data processing

### Spark ML :

- load `TPs/data/iris.csv` file in Panda DataFrame
- create Spark DataFrame
- create Pipeline to prepare data for machine learning
- compute a predicting model
- evaluate the predicting model

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

Project : Spark ML, Image processing

project :

- load Lena.jpg file
- develop a parallel median Filter in python with Spark

# Outline

## 1 Introduction

## 2 Hadoop

- Introduction
- Architecture
- HDFS
- Yarn
- MapReduce

## 3 Spark

- Introduction
- Architecture and Ecosystem
- Spark Modules : Core, SQL, Mlib, Streaming and GraphX
  - Spark Core
  - Spark SQL
  - Spark Mlib
  - Spark Streaming
  - Spark GraphX
- Spark : CLI

## 4 Dask

# TPs

## Test0 Test1 Test2 with Dask

Test0 :

- create Dask client
- create liste of integer
- partition list with dask

Test1 :

- compute square of integer list
- print square list

Test2 :

- compute PI

Test3 :

- create Dask bags, Array and DataFrame form h5, csv and json files
- directories small\_weather account and nycflights