BigData and Machine Learning with Hadoop and Spark Frameworks

Jean-Marc GRATIEN[1]

[1] Department of Computer Science
IFP New Energy

January 22th 2024 / Master Data-AI

Outline I

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`

BigData

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

Outline

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

- 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

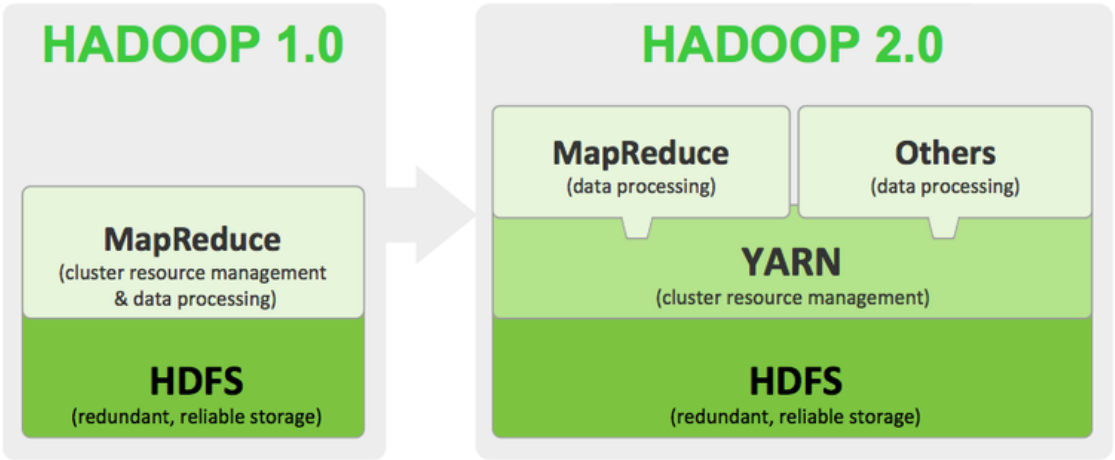
Why Hadoop?

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

Outline

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



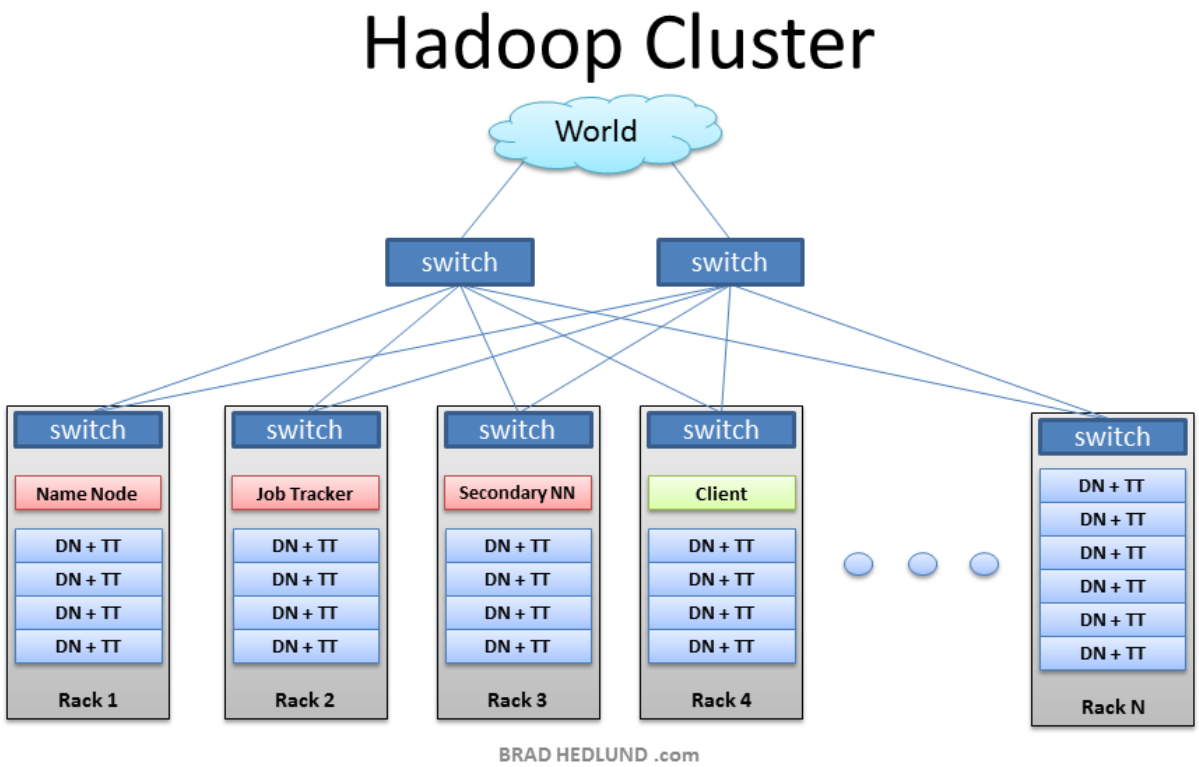
Hadoop ecosystem :

- 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 Distributions :

- Hortonworks
- Cloudera
- MAPR

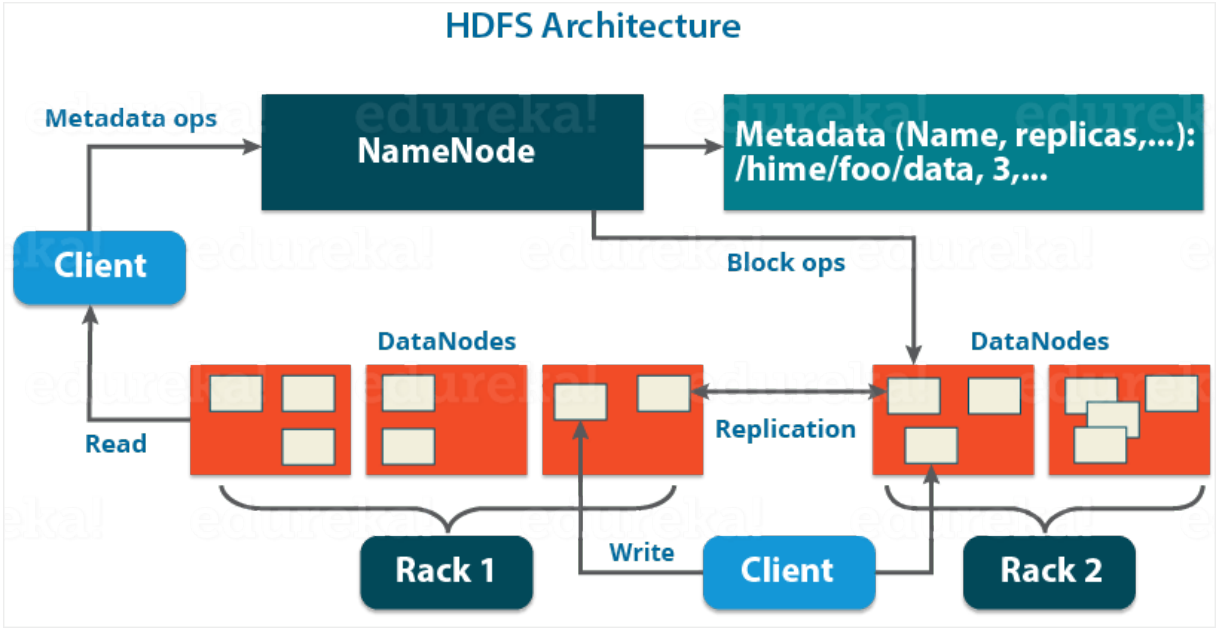
Outline



Purpose : Scalability, Fault tolerance and Reliality management

Concepts

- NameNode
- DataNode
- Replication
- Block, Blocksize



HDFS : Hadoop Distributed Filesystem

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
```

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
```

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
```

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, otherwize 0
fs -cat <filename>	
fs -tail [-f] <filename>	

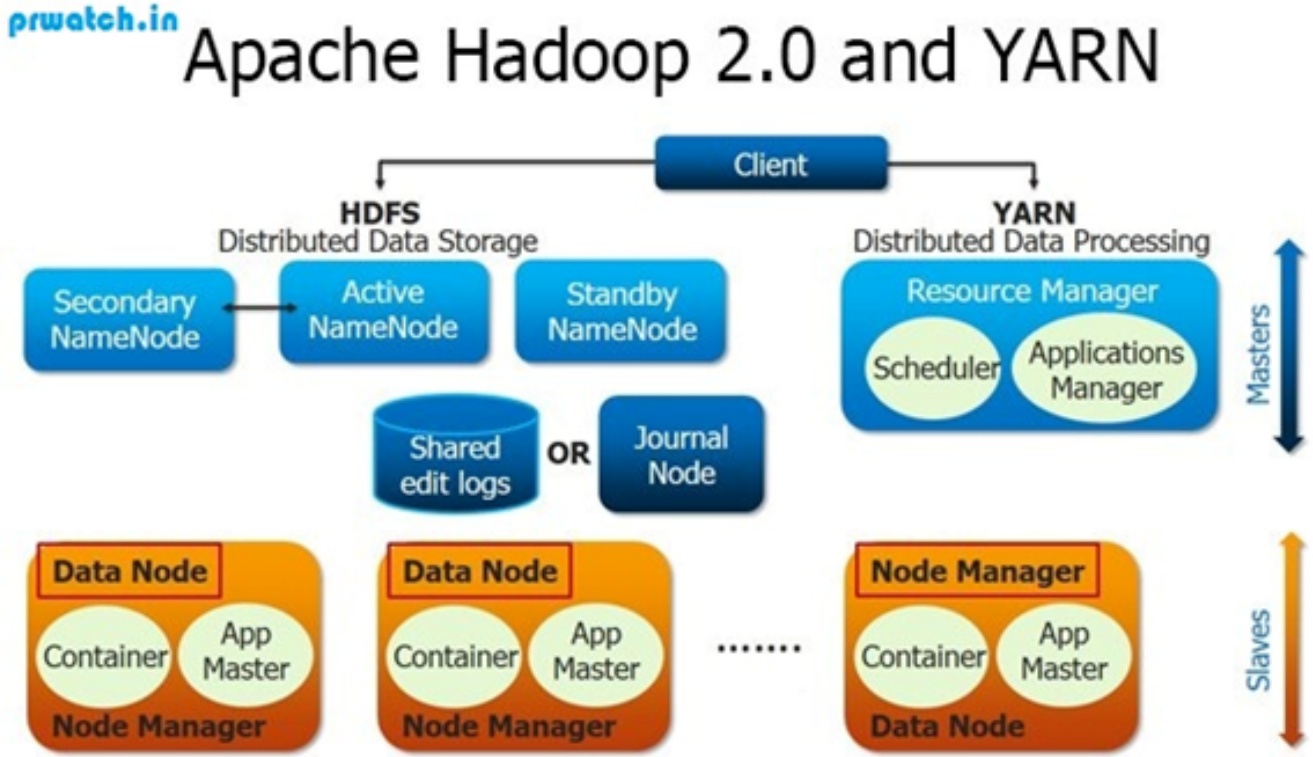
Commande name	Description
fs -mv <src><dest>	move file or directory within HDFS
fs -cp <src> <dest>	copy file or directory within HDFS
fs -rm <path>	remove file or directory within HDFS
fs -rmr <path>	rm recursively
fs -put <localSrc> <dest>	copy files or dirs from local FS to HDFS

Hadoop Framework: HDFS

HDFS commands list

Commande name	Description
fs -copyFromLocal <localSrc> <dest>	identical to put
fs -moveFromLocal <localSrc> <dest>	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


prwatch.in
Apache Hadoop 2.0 and YARN
Client
HDFS
Distributed Data Storage
YARN
Distributed Data Processing
Secondary NameNode
Active NameNode
Standby NameNode
Resource Manager
Scheduler
Applications Manager
Masters
Shared edit logs
OR
Journal Node
Data Node
Container
App Master
Node Manager
Data Node
Container
App Master
Node Manager
.....
Node Manager
Container
App Master
Data Node
Slaves

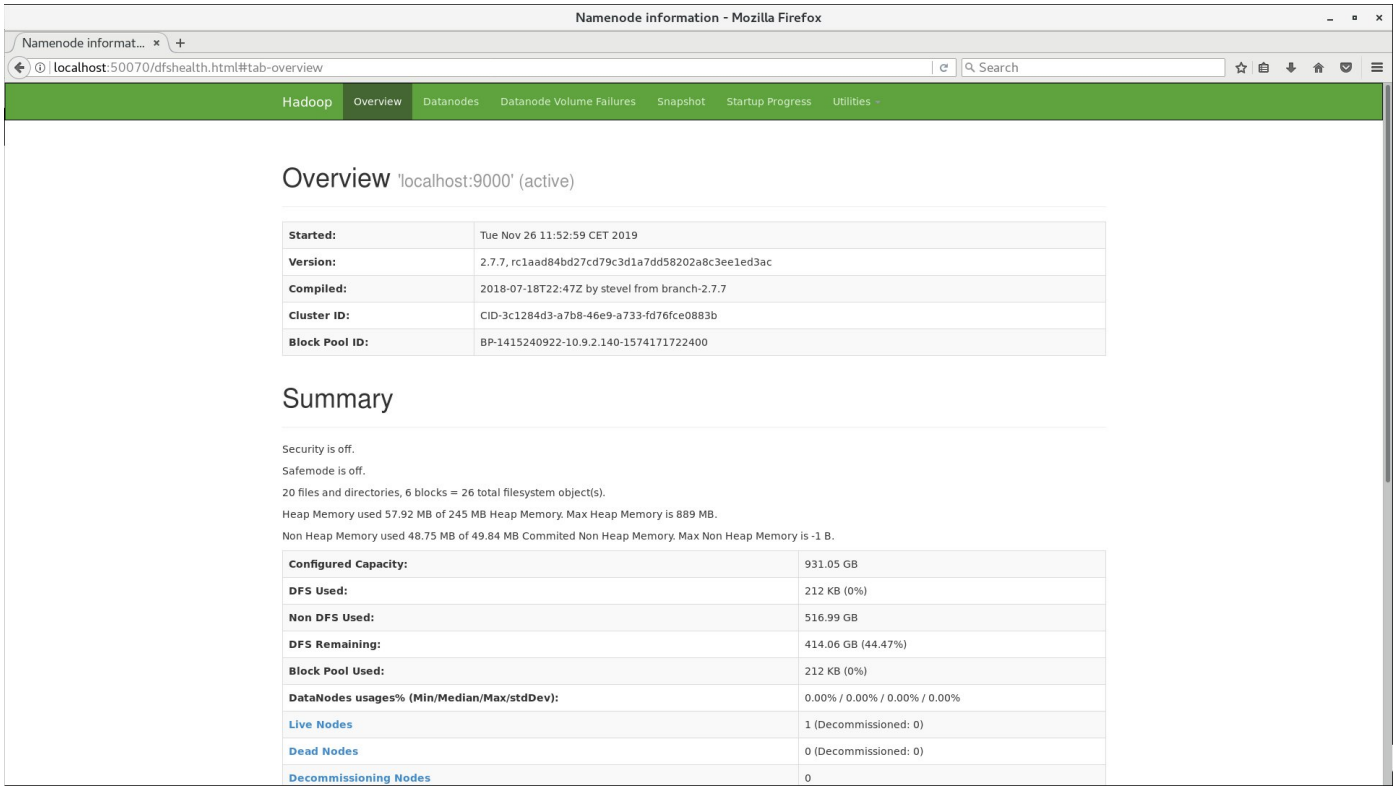
Starting YARN

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

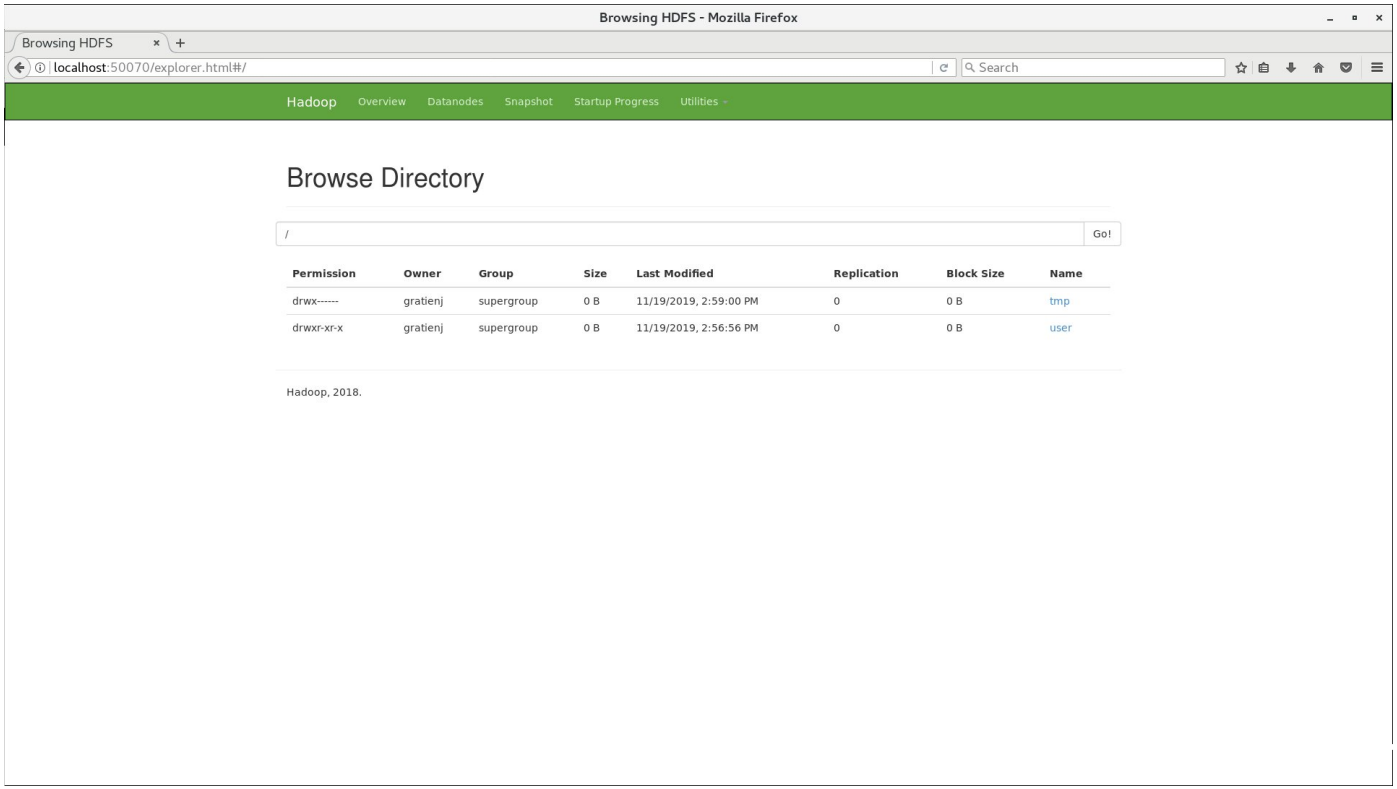
Shutting down YARN

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

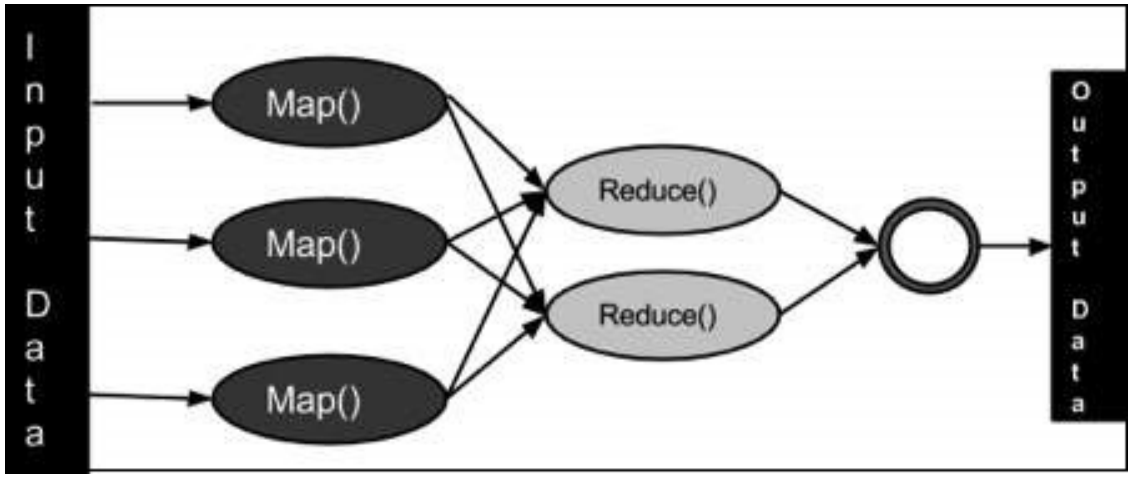
Web tools on : `http://<hostname>:<port>`
<hostname>:<port> (default localhost:50070) are defined in hdfs-site.xml
Overview web page:



File browser web page:



Outline



MapReduce Framework

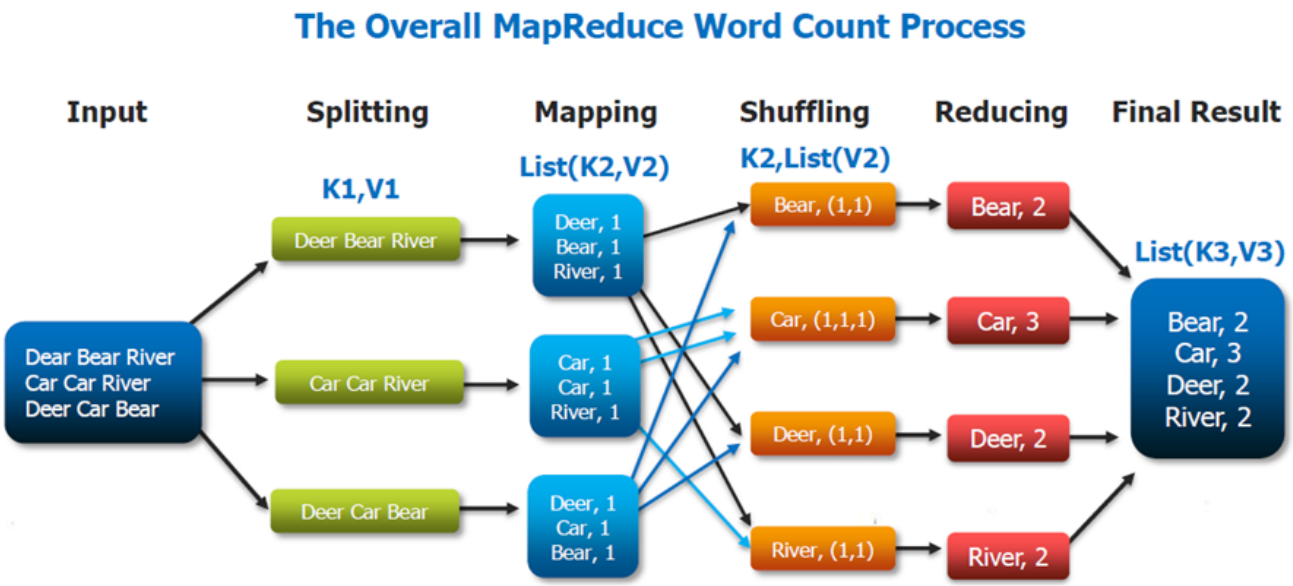
MapReduce Algorithm :

- 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.

Input and Output :

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

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


The Overall MapReduce Word Count Process
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1,V1
List(K2,V2)
K2,List(V2)
List(K3,V3)
Dear Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

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 }
```

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 }
```

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 }
```

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 }
```

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
```

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
```

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))
```

Reducer python script Part 1

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

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))
```

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
```

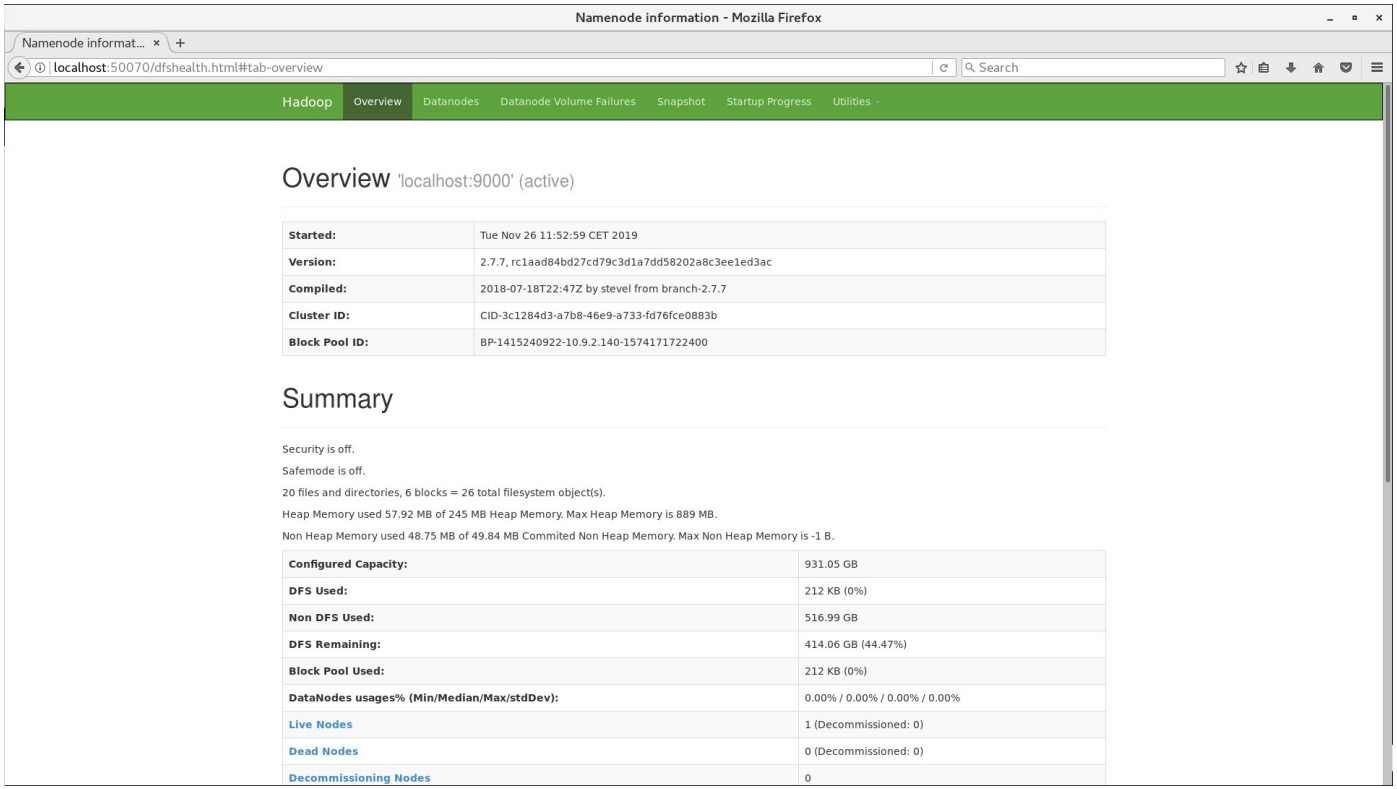
Python example with Hadoop Streaming : Part 2

Check results

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

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

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



Outline

Cloud Storage Solution AWS Bucket S3 API

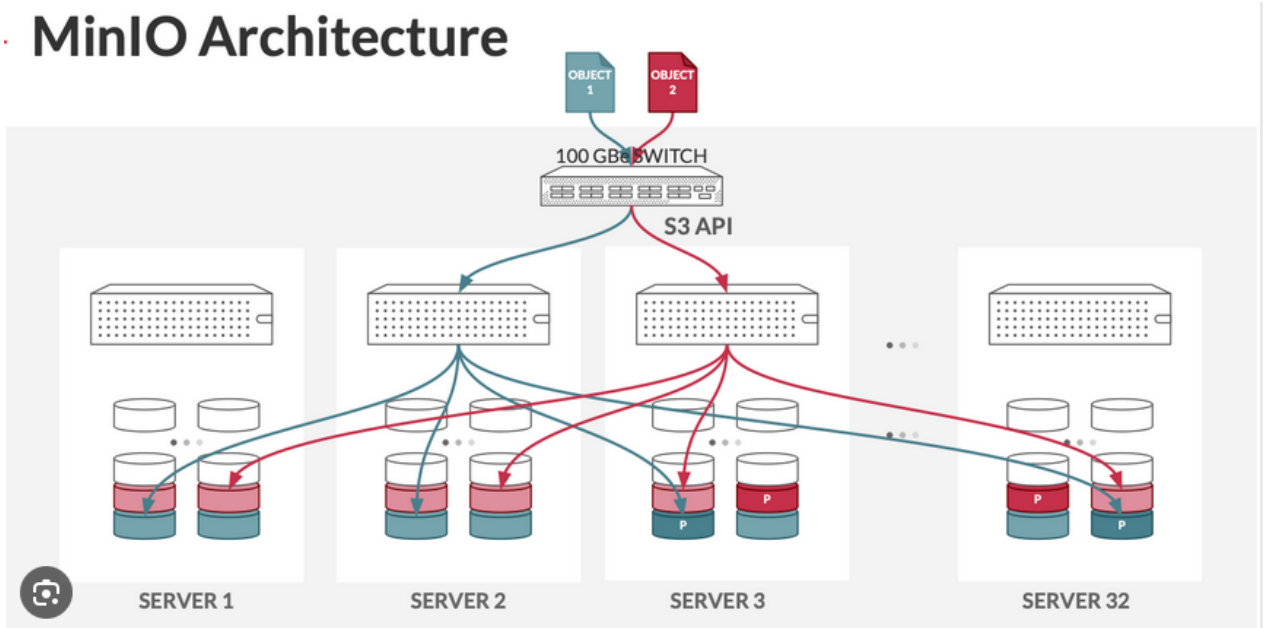
Introduction Bucket S3 storage

- An Amazon S3 bucket : units called objects instead of files.
  - a public cloud storage resource available in Amazon Web Services (AWS)
- Simple Storage Service (S3) platform :
  - provides object-based storage, where data is stored inside S3 buckets in distinct developped by AWS
- S3 API

Outline

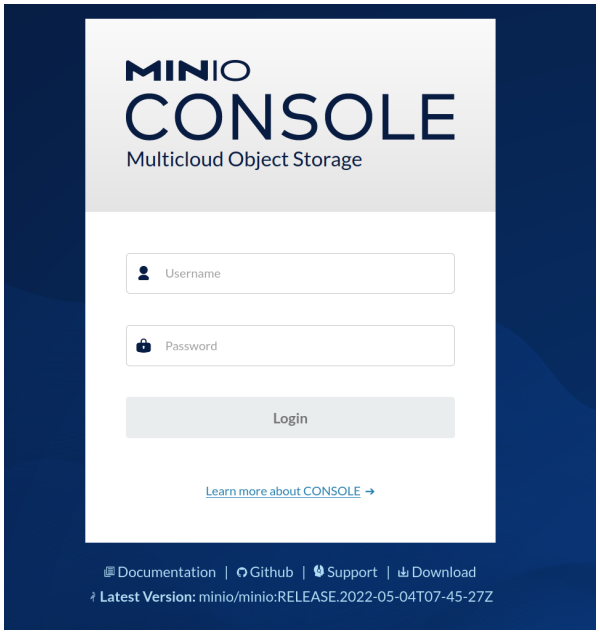
MinIO storage

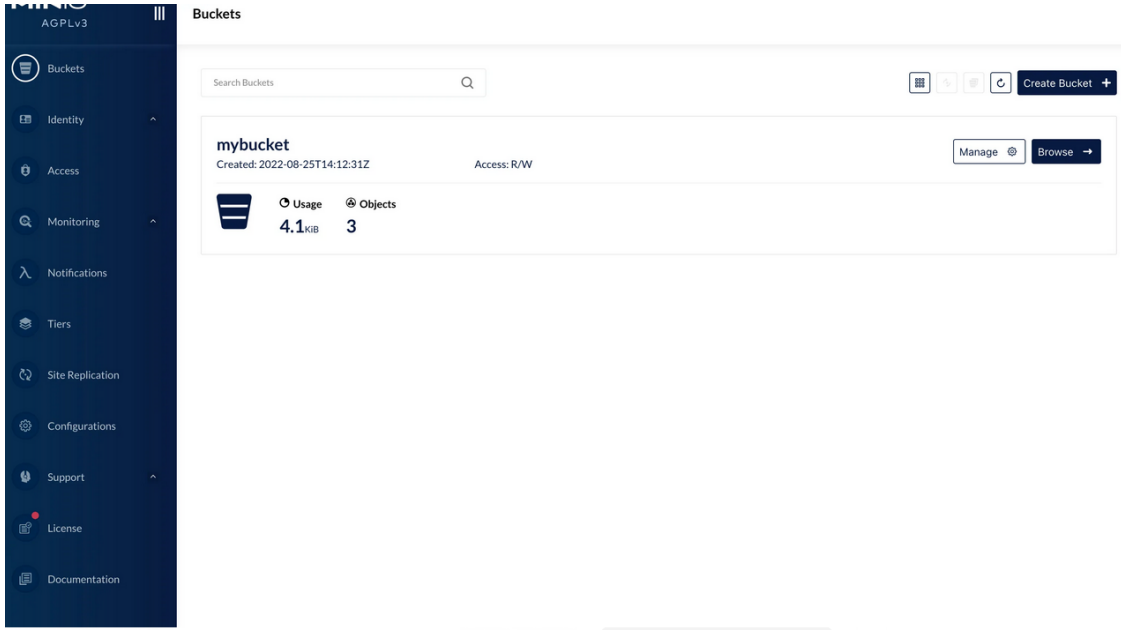
MinIO is an object storage solution that provides an Amazon Web Services S3-compatible API and supports all core S3 features.
MinIO is built to deploy anywhere - public or private cloud, baremetal infrastructure, orchestrated environments, and edge infrastructure.



MinIO storage

MinIO Client : Web Interface





- MinIO CLI

Commande name	Description
alias	set, remove and list aliases in configuration file
ls	list buckets and objects
mb	make a bucket
rb	remove a bucket
cp	copy objects
mirror	synchronize object(s) to a remote site
cat	display object contents
head	display first 'n' lines of an object
pipe	stream STDIN to an object
share	generate URL for temporary access to an object
find	search for objects
sql	run sql queries on objects
stat	show object metadata

MinIO storage

MinIO Client : Command line

- MinIO CLI

Commande name	Description
mv	move objects
tree	list buckets and objects in a tree format
du	summarize disk usage recursively
retention	set retention for object(s)
legalhold	set legal hold for object(s)
diff	list differences in object name, size, and date between tw
rm	remove objects
encrypt	manage bucket encryption config
event	manage object notifications
watch	listen for object notification events
undo	undo PUT/DELETE operations
anonymous	manage anonymous access to buckets and objects
tag	manage tags for bucket(s) and object(s)
ilm	manage bucket lifecycle
version	manage bucket versioning
replicate	configure server side bucket replication
admin	manage MinIO servers

- MinIO CLI

Commande name	Description
ilm	manage bucket lifecycle
version	manage bucket versioning
replicate	configure server side bucket replication
admin	manage MinIO servers
update	update mc to latest release
ping	perform liveness check

Python example

```
1  from minio import Minio
2  # Initialize MinIO client
3  client = Minio('localhost:9000',
4                 access_key='minio_access_key',
5                 secret_key='minio_secret_key',
6                 secure=False)
7  # Make a bucket
8  client.make_bucket('mybucket')
9  # Upload an object
10 client.fput_object('mybucket', 'myfile.txt', 'myfile.txt')
11 # List objects
12 objects = client.list_objects('mybucket', recursive=True)
13 for obj in objects:
14     print(obj.object_name)
```

Step 1 : launch Minio server in local mode

```
1  $ docker run -p 9000:9000 -p 9001:9001 \
2    quay.io/minio/minio server /data --console-
         address ":9001"
3
4    Formatting 1st pool, 1 set(s), 1 drives per set.
5  WARNING: Host local has more than 0 drives of set.
       A host failure will result in data becoming
       unavailable.
6  WARNING: Detected default credentials 'minioadmin:
       minioadmin', we recommend that you change these
        values with 'MINIO_ROOT_USER' and '
       MINIO_ROOT_PASSWORD' environment variables
7  MinIO Object Storage Server
8  Copyright: 2015-2023 MinIO, Inc.
9  License: GNU AGPLv3 <https://www.gnu.org/licenses/
       agpl-3.0.html>
10 Version: RELEASE.2023-11-11T08-14-41Z (go1.21.4
```

Step 2 : connect server with MC CLI client

```
$ mc alias set docker_minio http://127.0.0.1:9000
    minioadmin minioadmin

Added `docker_minio` successfully.
```

Step 3 : get info

```
$ mc admin info docker_minio
   127.0.0.1:9000
   Uptime: 21 minutes
   Version: 2023-11-11T08:14:41Z
   Network: 1/1 OK
   Drives: 1/1 OK
   Pool: 1

Pools:
   1st, Erasure sets: 1, Drives per erasure set: 1

0 B Used, 1 Bucket, 0 Objects
1 drive online, 0 drives offline
```

MinIO client :

- CLI console
- Web based console : `http://127.0.0.1:9001`
- Programing S3 Client (Java, Python, ...)

- add in hadoop-env.sh : export $HADOOP_OPTIONAL_TOOLS = hadoop - awsModifycore - site.xml$

core-site.xml

```
1  <property>
2    <name>fs.s3a.access.key</name>
3    <description>AWS access key ID used by S3A file system. Omit for IAM role-based or provider-based
         authentication.</description>
4    <value>theroot</value>
5  </property>
6
7  <property>
8    <name>fs.s3a.secret.key</name>
9    <description>AWS secret key used by S3A file system. Omit for IAM role-based or provider-based
         authentication.</description>
10   <value>theroot123</value>
11 </property>
12
13 <property>
14   <name>fs.s3a.endpoint</name>
15   <description>
16     AWS Simple Token Service Endpoint. If unset, uses the default endpoint.
17     Only used if AssumedRoleCredentialProvider is the AWS credential provider.
18   </description>
19   <value>http://127.0.0.1:9000</value>
20 </property>
```

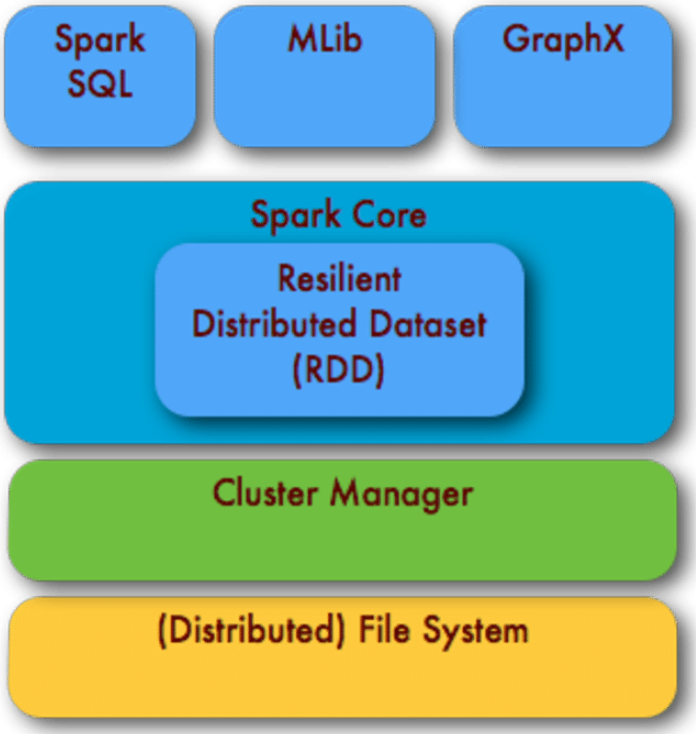
Outline

- 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

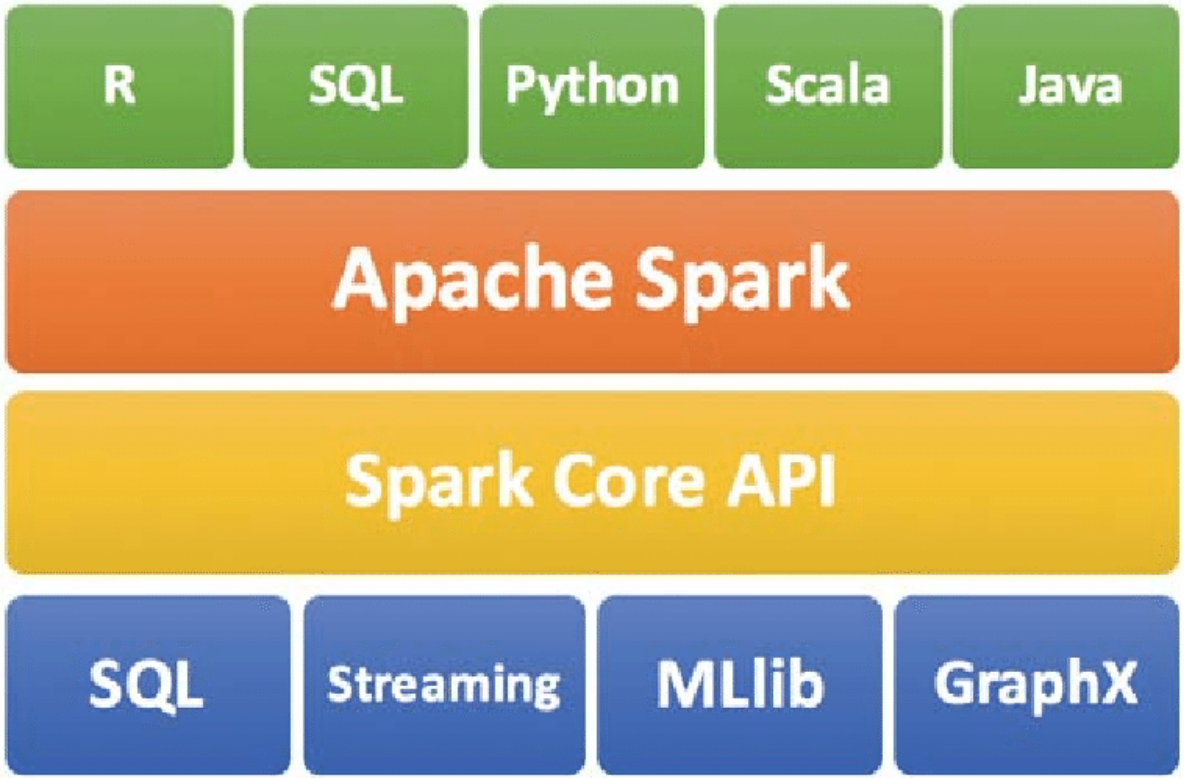
Spark Framework

Apache Spark Architecture



Spark Framework

Apache Spark Ecosystem



Outline

Spark Cluster Configurations :

- Local mode
- Cluster mode
- Client mode

Spark parallel concepts:

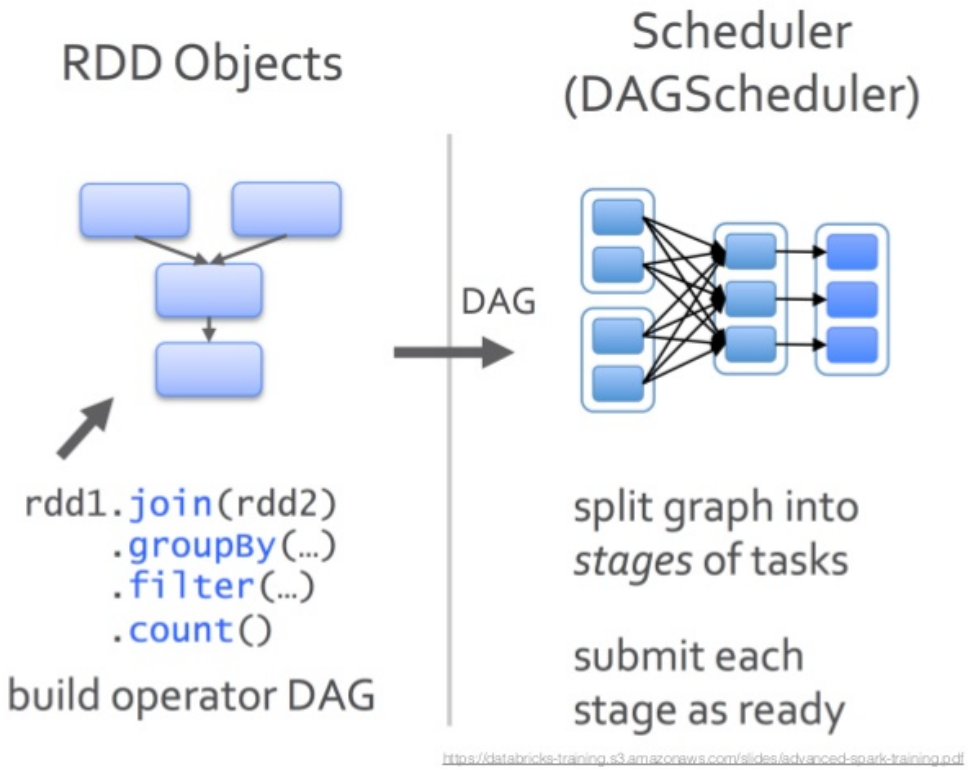
- multiple executors (private JVM)
- multiple cores per executor

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)
```

Create a SparkContext

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


RDD Objects
Scheduler
(DAGScheduler)
DAG
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
.count()
build operator DAG
split graph into
stages of tasks
submit each
stage as ready
https://databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf

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

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 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
```

Transformation :

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

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
groupBy()	create new set of (key,value)
groupByKey()	
reduceByKey()	
sample()	selection of lines
union()	fusion of two RDDs
join()	union without duplicate keys

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

Unified Data Abstraction

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

DATABRICKS

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)
```

Print DataFrame Schema

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

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)
```

Select columns

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

Select columns

```
+---+------+
|age|fnlwgt|
+---+------+
| 39| 77516|
| 50| 83311|
| 38|215646|
| 53|234721|
| 28|338409|
+---+------+
only showing top 5 rows
```

Select columns

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

Select columns

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

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

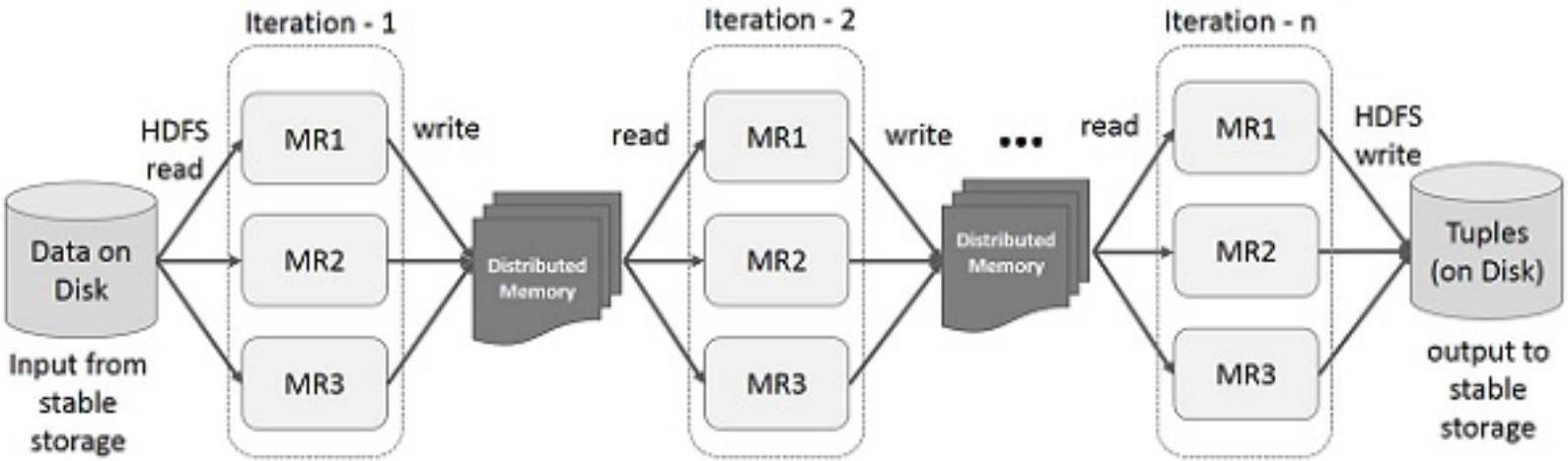
- count,
- mean,min,max
- standarddeviation

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 +-------+------------------+
```


Iteration - 1
Iteration - 2
Iteration - n
HDFS read
MR1
MR2
MR3
write
Data on Disk
Input from stable storage
Distributed Memory
read
write
read
HDFS write
Tuples (on Disk)
output to stable storage

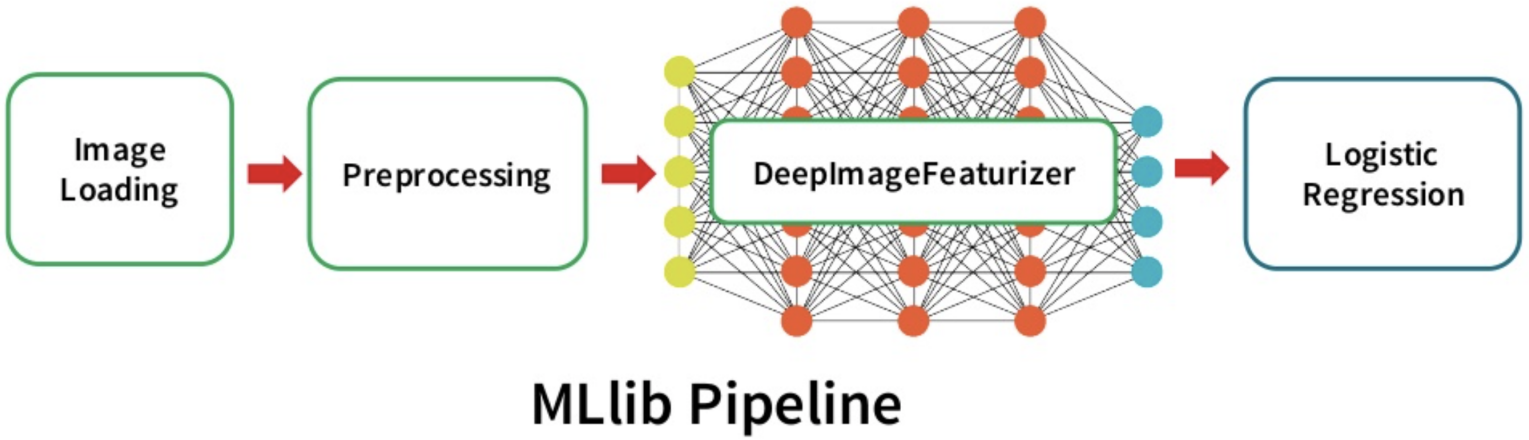


Image Loading
Preprocessing
DeepImageFeaturizer
Logistic Regression
MLlib Pipeline

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 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

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)
```

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)
```

Pipeline

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

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)
```

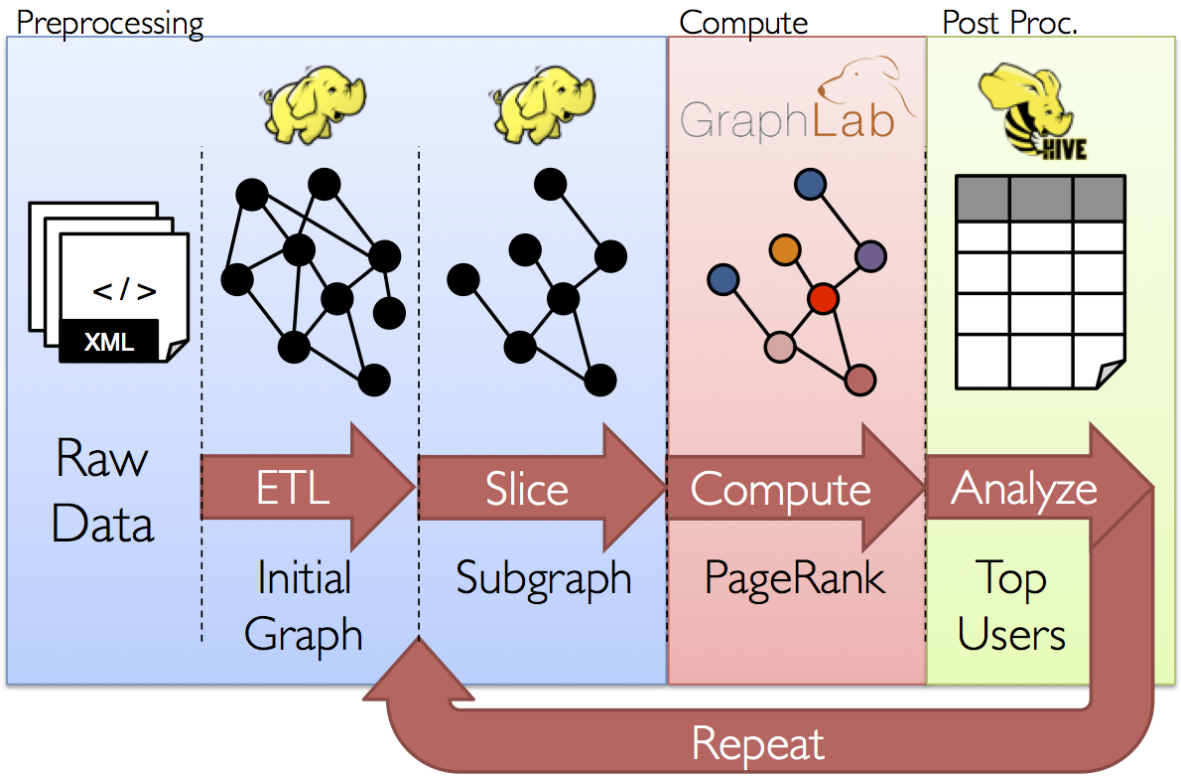
Train model

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

Make prediction

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


Kafka
Flume
HDFS/S3
Kinesis
Twitter
Spark
Streaming
HDFS
Databases
Dashboards


Preprocessing
Compute
Post Proc.
GraphLab
HIVE
</>
XML
Raw Data
ETL
Initial Graph
Slice
Subgraph
Compute
PageRank
Analyze
Top Users
Repeat

Outline

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 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 Cluster Mode

```
PYSPARK_DRIVER_PYTHON=/python_root/bin/python \
PYSPARK_PYTHON=./environment/bin/python \
spark-submit \
--conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=./environment/bin/python \
--master yarn \
--deploy-mode client \
--principal gratienj@IFP.FR \
--keytab /tmp/krb5cc_1109 \
--archives environment.tar.gz#environment \
script.py
```

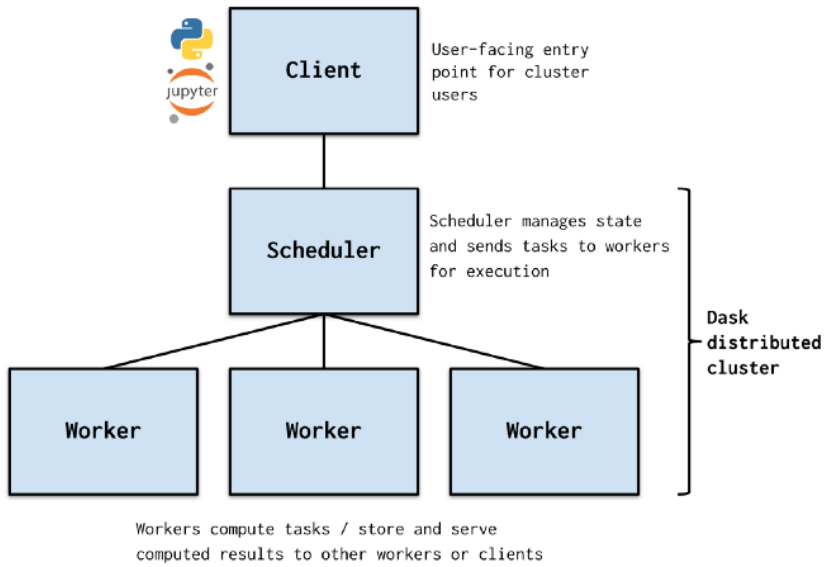
Outline

- 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

Dask architecture:

- Dask Cluster
- Dask Scheduler
- Dask collections

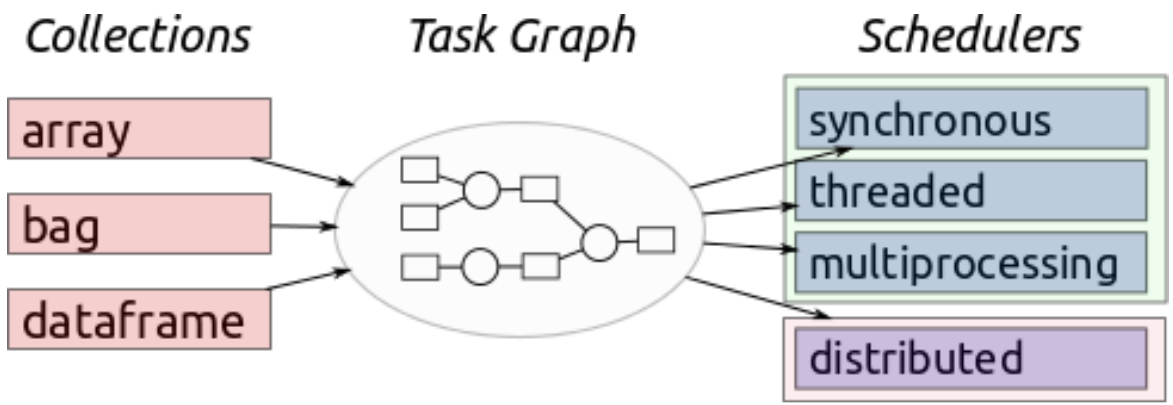


Various Dask cluster types:

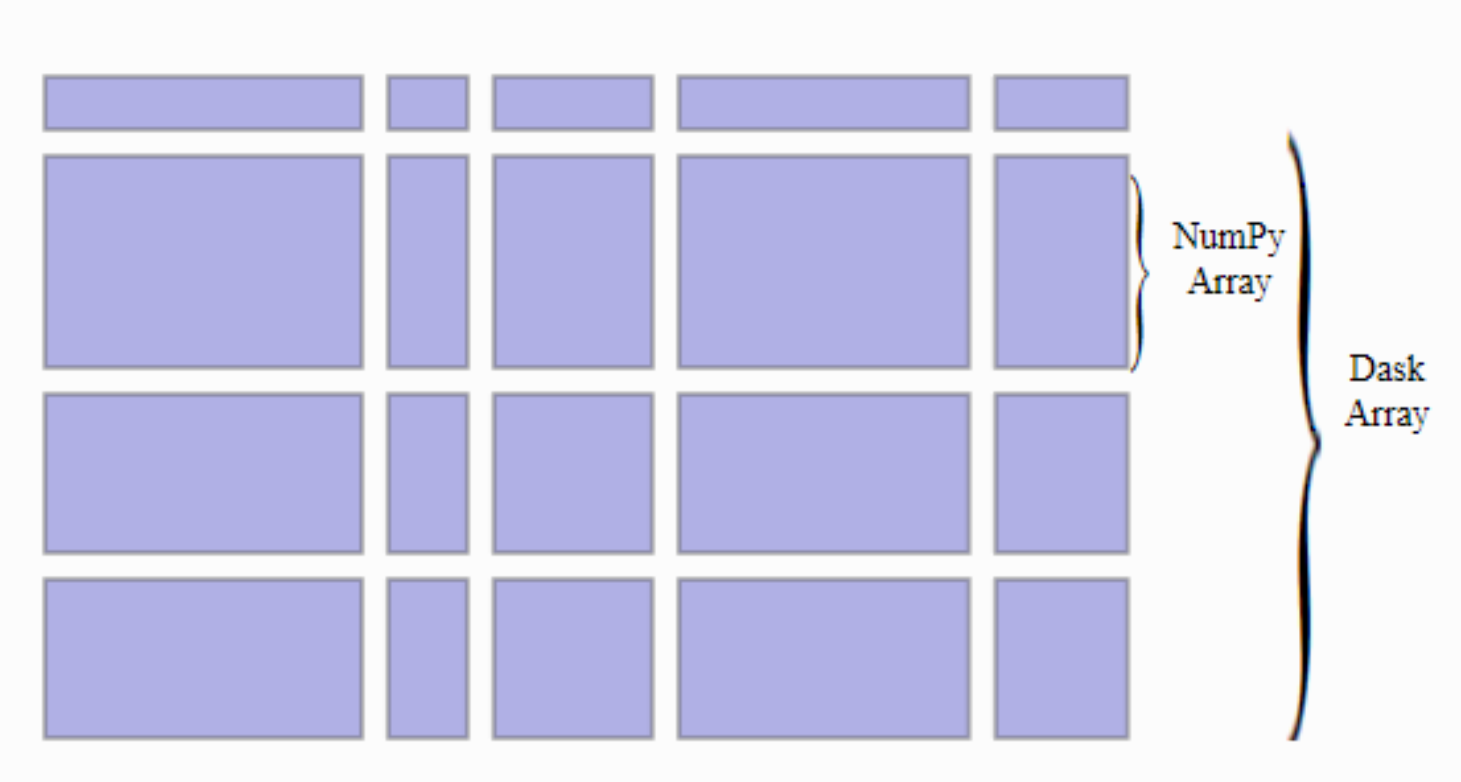
- Hadoop/Spark clusters running YARN
- HPC clusters running job managers like SLURM, SGE, PBS, LSF, or others common in academic and scientific labs
- Kubernetes clusters

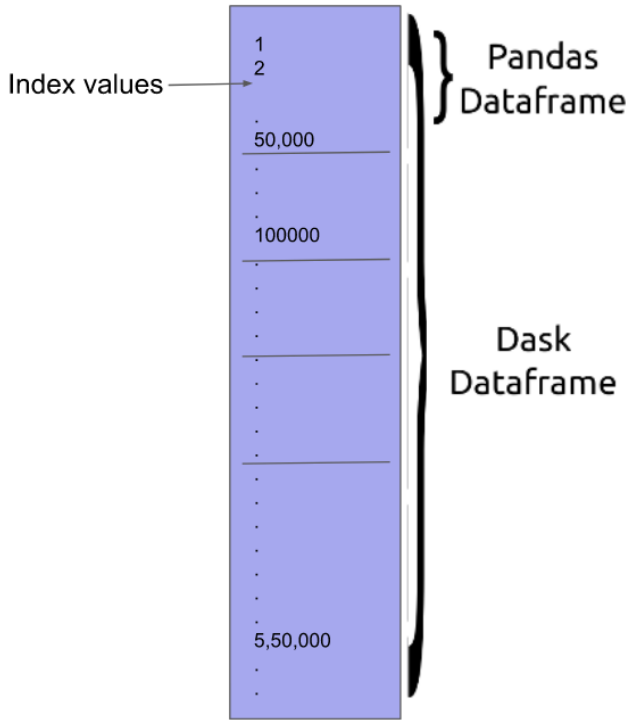
Dask Scheduler:

- 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 Array





Dask DataFrame

Outline

Installation

```
conda install dask
```

OR

Installation

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

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 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

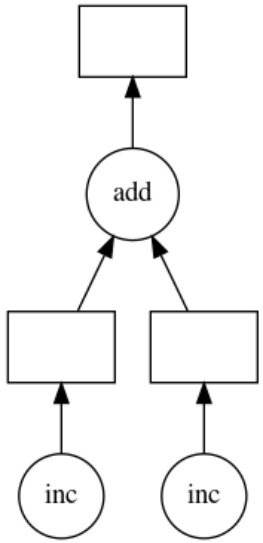
Dask Distributed

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 #visualize DAG
5 total.visualize()
6 # execute all tasks
7 total.compute()
```



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 evaluation

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

Dask progess

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

Dask get results

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

Dask persist

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

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

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 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 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 Ecosystem

Dask Array

Dask bag : Numpy lazy API

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

Dask bag : evaluation

```
print(result.compute())
```

Dask Ecosystem

Dask DataFrame

Dask DatFrame

```
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 bag : evaluation

```
1 print(df.DepDelay.max().compute())
```

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  # Scale out to ten such workers
9  cluster.scale(10)
10
11 # Connect to the cluster
12 client = Client(cluster)
```

Dask Scheduler and Workers SetUp

```
$ dask-scheduler
Scheduler at:   tcp://192.0.0.100:8786

$ dask-worker tcp://192.0.0.100:8786
Start worker at:  tcp://192.0.0.1:12345
Registered to:    tcp://192.0.0.100:8786

$ dask-worker tcp://192.0.0.100:8786
Start worker at:  tcp://192.0.0.2:40483
Registered to:    tcp://192.0.0.100:8786
```

Dask Client SetUp

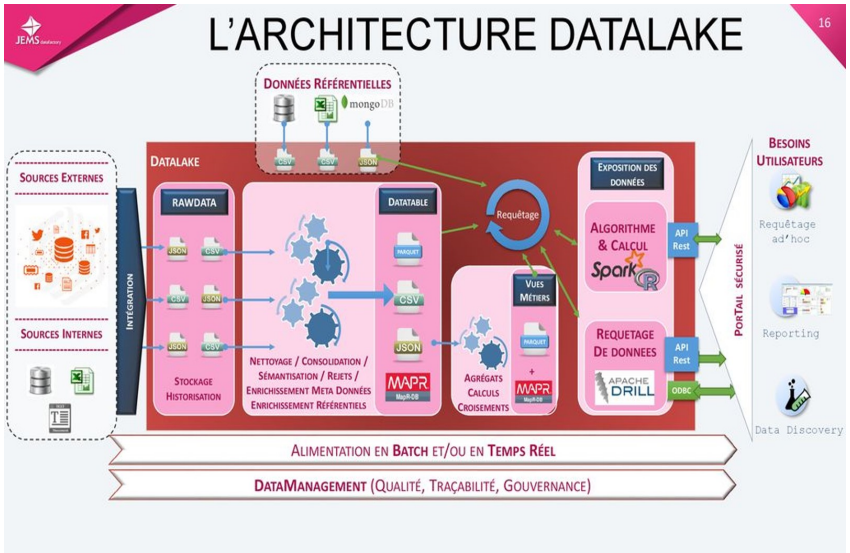
```
from distributed import Client
client = Client('192.0.0.100:8786')
```

Outline

Hardware architecture to handle large scale data analysis issues

Issues:

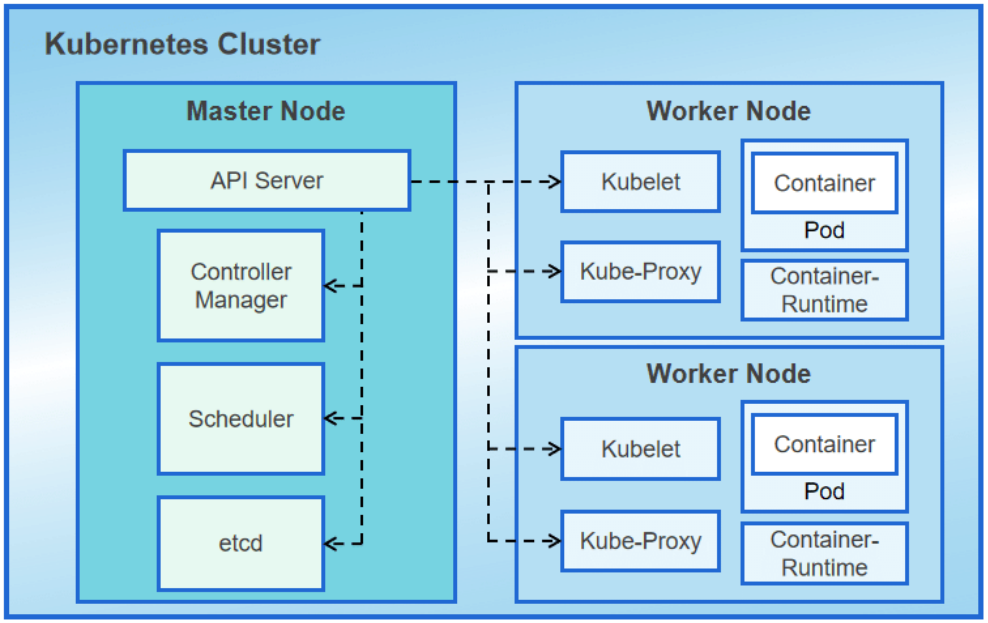
- Data collection
- Data Storage
- Data processing
- Post processing



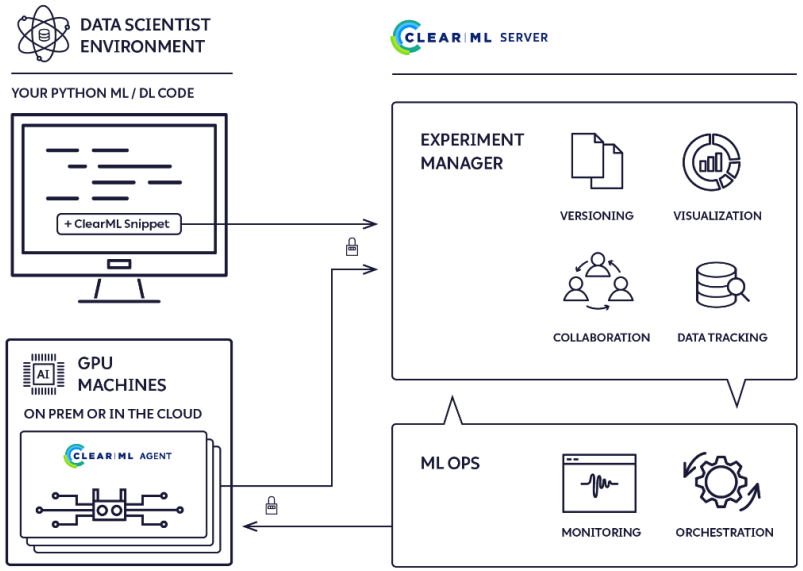
Software environment to handle computing issues

Issues:

- Variety of algorithms
- Variery of software environment
- Software requirements
- Hardware requirements



MLOp : set of tools to design and execute large scale data analysis pipelines



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

Hadoop prerequires : 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
```

Hadoop prerequires : 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
```

Hadoop prerequires : Check ssh service and Java installation

Check java

```
> java -version
```

Installation Java Jdk

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

Check java version

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

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
```

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
```

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
```

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>
```

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>
```

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>
```

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>
```

Lauch all services

```
> $HADOOP_HOME/sbin/start-dfs.sh
> $HADOOP_HOME/sbin/start-yarn.sh
```

Check lauched services

```
> jps

26867 DataNode
28228 Jps
27285 ResourceManager
26695 NameNode
27082 SecondaryNameNode
27420 NodeManager
```

- 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

- Check Docker installation

Docker : Set up

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

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

Docker : Safely close the hadoop cluster

```
> docker-compose down
```

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

Outline

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
```

Java project:

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

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

- apply Java WordCount application
- check results

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

Remind : Docker command Quick Sheet :
`BigDataHadoopSparkDaskCourse/TPs/doc/Docker.md`

Docker : exec running conatairner named namenode

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

Docker : copy file in a container

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

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

Spark Installation: spark-3.0.1-bin-hadoop-3.2.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
```

Installation

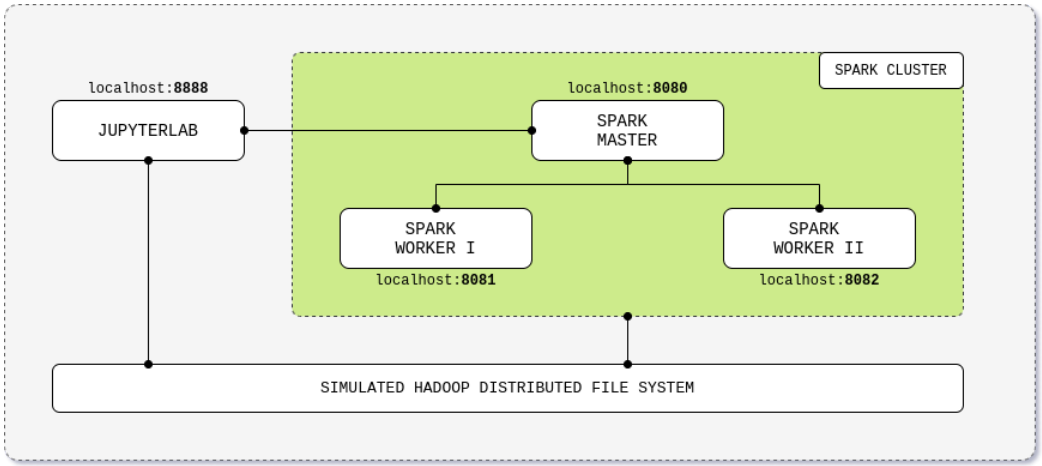
```
> cd /home/hduser
> export SPARK_HOME=/home/hduser/local/spark
> export PATH=$SPARK_HOME/bin:$PATH
> export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
> export PYSPARK_PYTHON="path_to_python"
> pip install pyspark
> pip install findspark
> sbin/start-master.sh
> sbin/start-slave.sh spark://localhost:7077
```

test Spark shell

```
> spark-shell

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://localhost:4040
Spark context available as 'sc' (master = local[*], app id = local-1578948405576).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.0-preview2
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_92)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

test Spark shell

```
> pyspark


Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
/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.
  DeprecationWarning)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0-preview2
      /_/

Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
>>>
```



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
```

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

Realize Test0, Test 1 and Test2 with the JupiterLab Nodebook

Deploy Spark Cluster

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

Connect to JupiterLab NoteBook at `http://localhost:8888`

Outline

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

Spark ML :

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

Outline

project :

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

Outline

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