Big Data Computing

Paolo Nesi, Gianni Pantaleo, Imad Zaza

DISIT Lab
Dipartimento di Ingegneria dell'Informazione, DINFO
Università degli Studi di Firenze
Via S. Marta 3, 50139, Firenze, Italy
Tel: +39-055-4796567, fax: +39-055-4796363
http://www.disit.dinfo.unifi.it ***alias*** http://www.disit.org

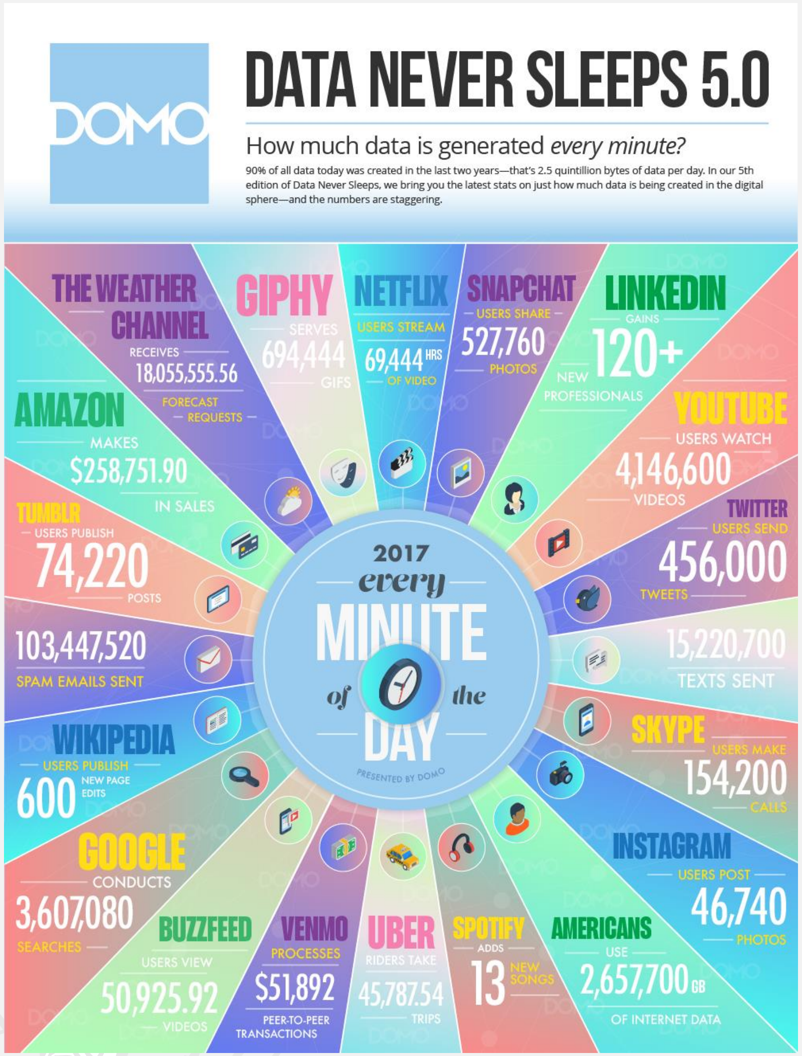
29 Ottobre 2019

Agenda

- Prologue
- Apache Hadoop
- Monitoring
- Apache HBASE
- Apache Phoenix
- Case studio

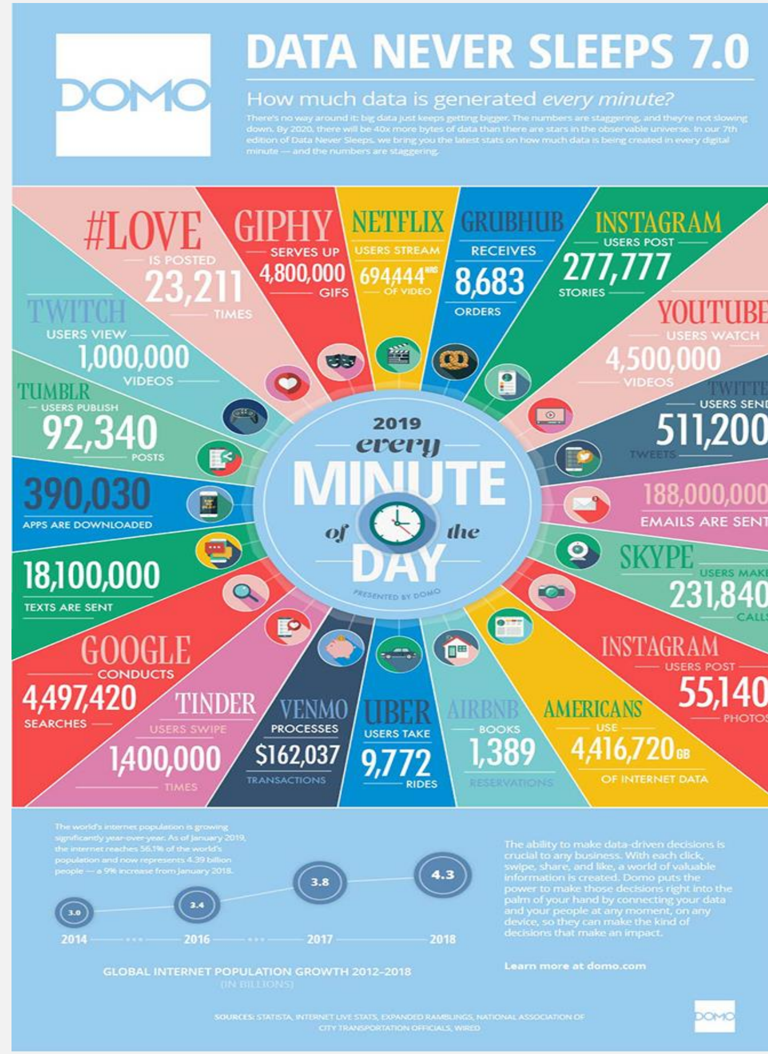
Agenda

- Prologue
- Apache Hadoop
- Monitoring
- Apache HBASE
- Apache Phoenix
- Case studio



Data Never Sleeps 5.0 2017 Report

https://www.domo.com/learn/data-never-sleeps-5



Data Never Sleeps 7.0 2019 Report

https://www.domo.com/learn/data-never-sleeps-7

Big Data Problem: Twitter Data Analytics

Twitter is an example big data source

BI on Twitter & social data is growing in demand

Possibile problems:

- *Count the number of tweets containg occurrence of one or more search string (e.g. «pippo pluto», «pippo OR pluto») per day in a given time interval*
- *NLP (Keywords, Keyphrase extraction and grammatical analysis on natural language text)*
- *Data Analytics…*

Big Data Problem: Twitter Data Analytics

TWITTER ANALYTICS

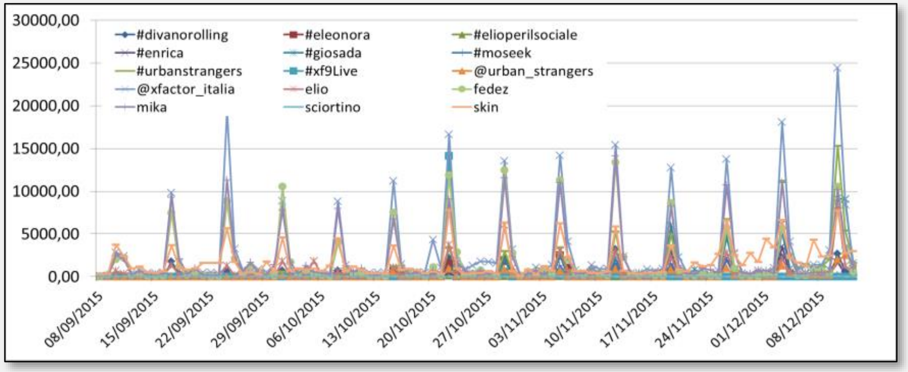
Many different contexts and application areas:

- ***Collect users' information*** *about* ***quality of services***
- ***Event Monitoring*** *- crowd size estimation, voting results, predicting TV audience etc.*
- ***Early Warning*** *- monitoring critical situations for alerts providing (weather alerts, spread of contagious diseases, natural disasters etc.)*

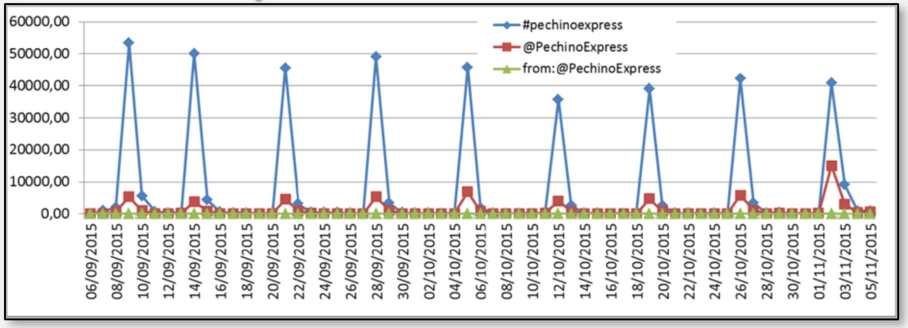
Research @ DISIT Lab

Correlazione tra numero di Tweets e Audience TV

Xfactor 9



Pechino Express 2015



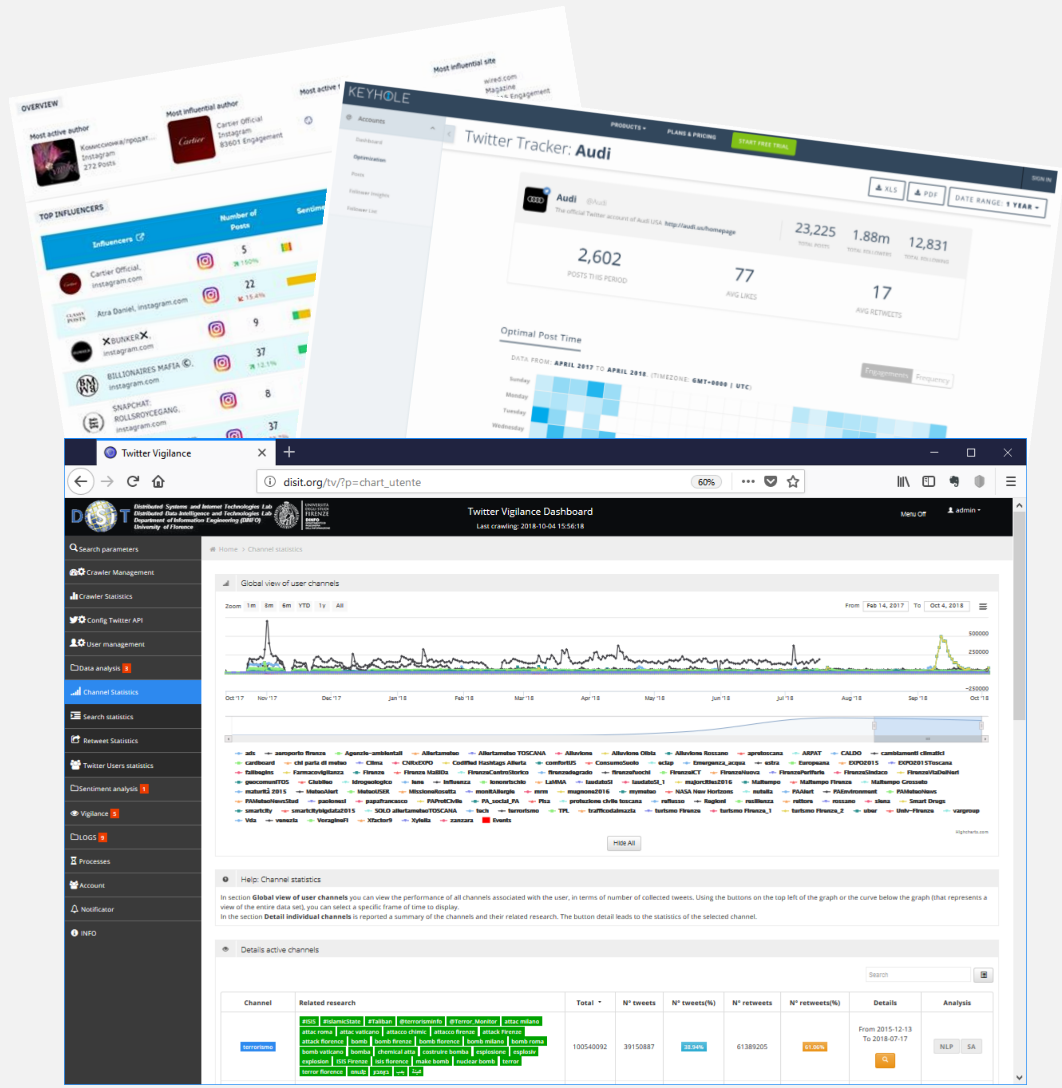
Crisci, A., Grasso, V., Nesi, P., Pantaleo, G., Paoli, I., Zaza, I. (2017), "Predicting TV programme audience by using Twitter based metrics", in Multimedia Tools And Applications, pp. 1-30, ISSN:1380-7501.

Big Data Problem: Twitter numbers

- 511,000 Tweets are sent every minute (2019)
- In 2016, Twitter has 310 million monthly active users
  (almost the same as the U.S. population)
- A total of 1.3 billion accounts have been created
- Of those, 44% made an account and left before ever sending a Tweet

What we need ?

- A system which crawls Twitter for tweets matching our queries
- A system storing collected tweets
- Metric processing procedures and Analytics
- Visual Analytics of processed Big Data (Dashboards, graphs etc...)



Single Host

I. **Develop a data model**
II. Use an RDBMS as data backend
III. Use SQL as query language wrappred in java or php … application

Single Host

I. Develop a data model

II. Use an RDBMS as data backend

III. Use SQL as query language wrappred in java or php … application

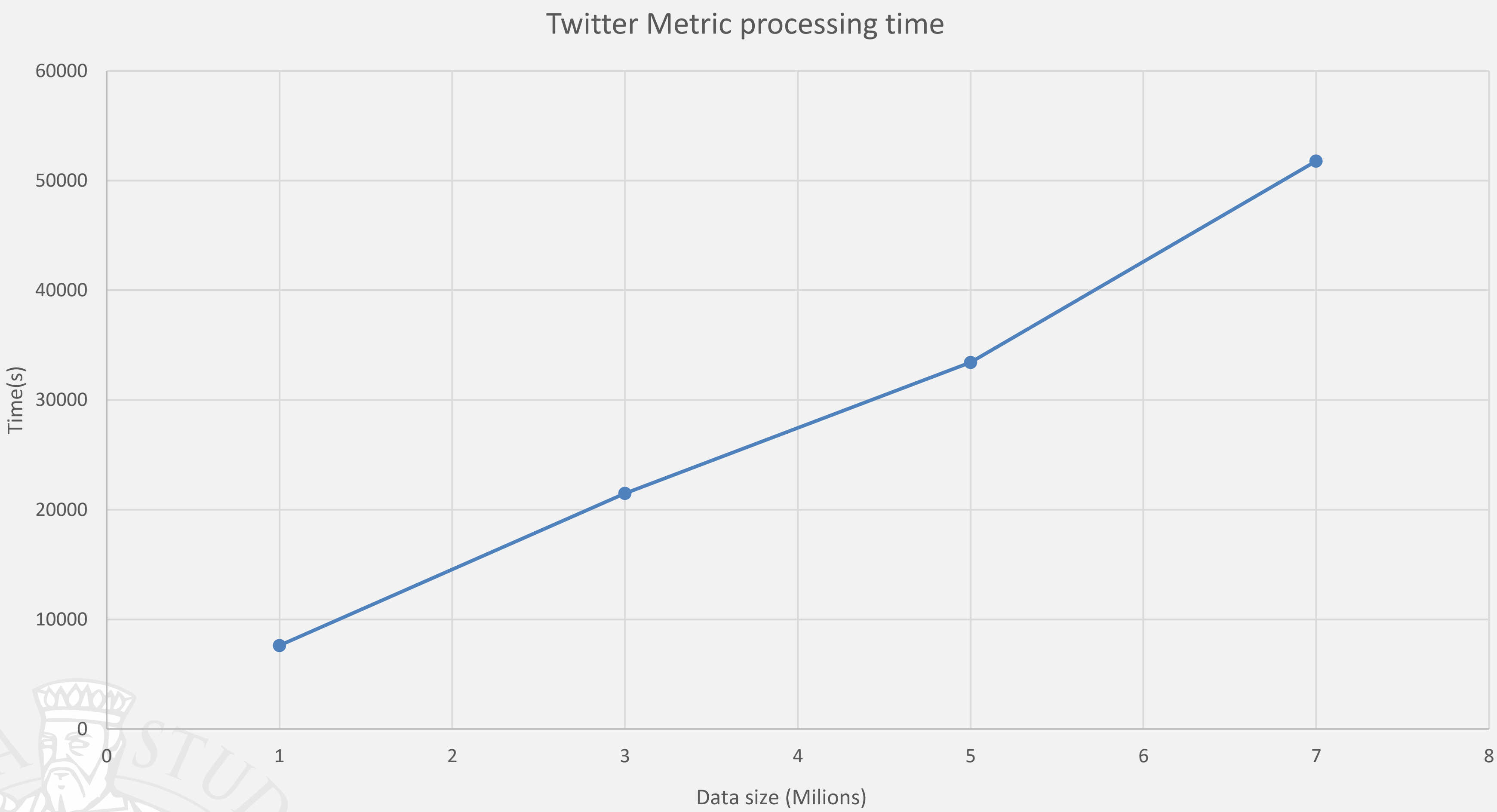
Single Host

I. Develop a data model
II. Use an RDBMS as data backend
III. **Use SQL as query language wrappred in java or php … application**

Problem

- Tweets collected grows fast
  - > Computation time degrade
  - > Reliability and Avalibility depends merely on hardware

Single Host


Twitter Metric processing time
60000
50000
40000
30000
20000
10000
0
Time(s)
0
1
2
3
4
5
6
7
8
Data size (Milions)

Agenda

- Prologue
- Apache Hadoop
- Monitoring
- Apache HBASE
- Apache Phoenix
- Case studio

Cluster

- A **computer cluster** is a group of linked computers, working together closely so that in many respects they form a single computer.

- The components of a cluster are commonly, but not always, connected to each other through fast local area networks.

- Clusters are **usually** deployed to improve performance and/or availability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Cluster (cont.)

Cluster consists of:

- Nodes (master + slaves)
- Network
- OS
- Cluster middleware which permits the computation

Pigsty

- 4 «pigs»
  - 3 pigs
    - 4GB Ram , dual core, 1TB disk
  - 1 pig
    - 8GB ram, dual core, 1TB disk
- 3 Virtual Servers
  - 1 vm
    - 4GB Ram, dual core, 700GB disk
  - 1 vm
    - 2GB Ram, Dual core, 700GB disk
  - 1 vm
    - 1GB Ram, Dual core, 700GB disk

hadoop

HDFS

Storage

MapReduce

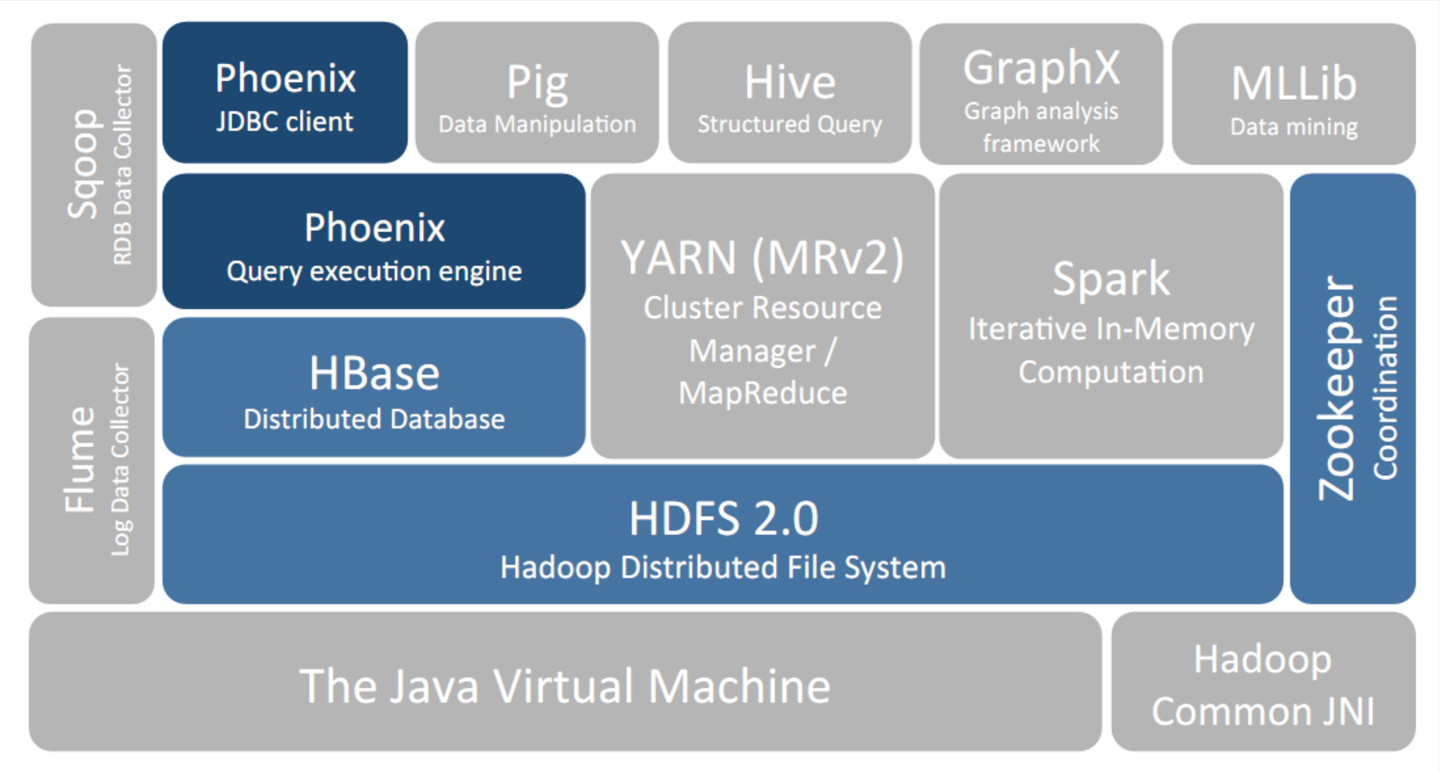
Processing

The Apache™ Hadoop® project develops open-source
Software for reliable, scalable, distributed computing

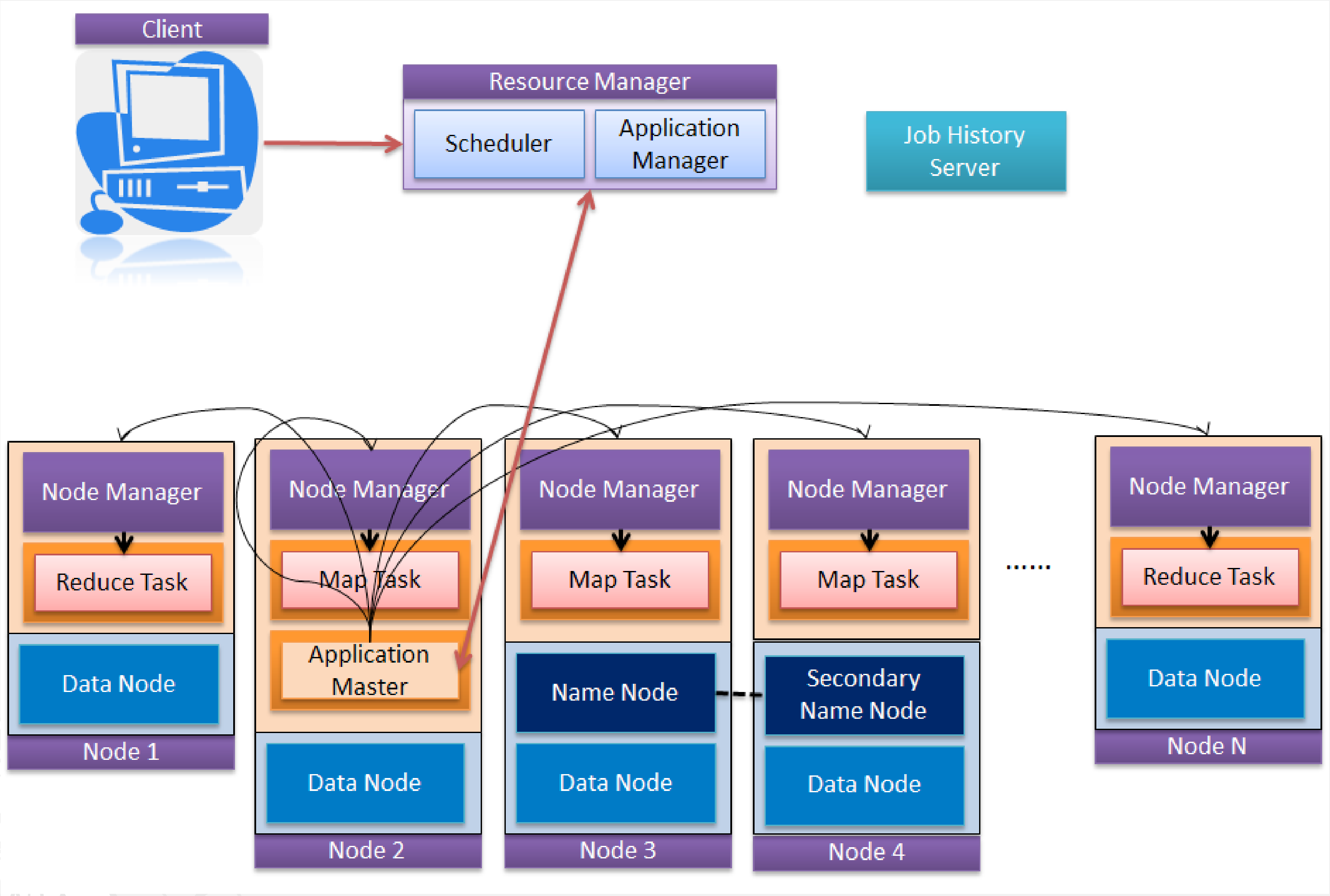
Apache Hadoop

Storing data @hadoop

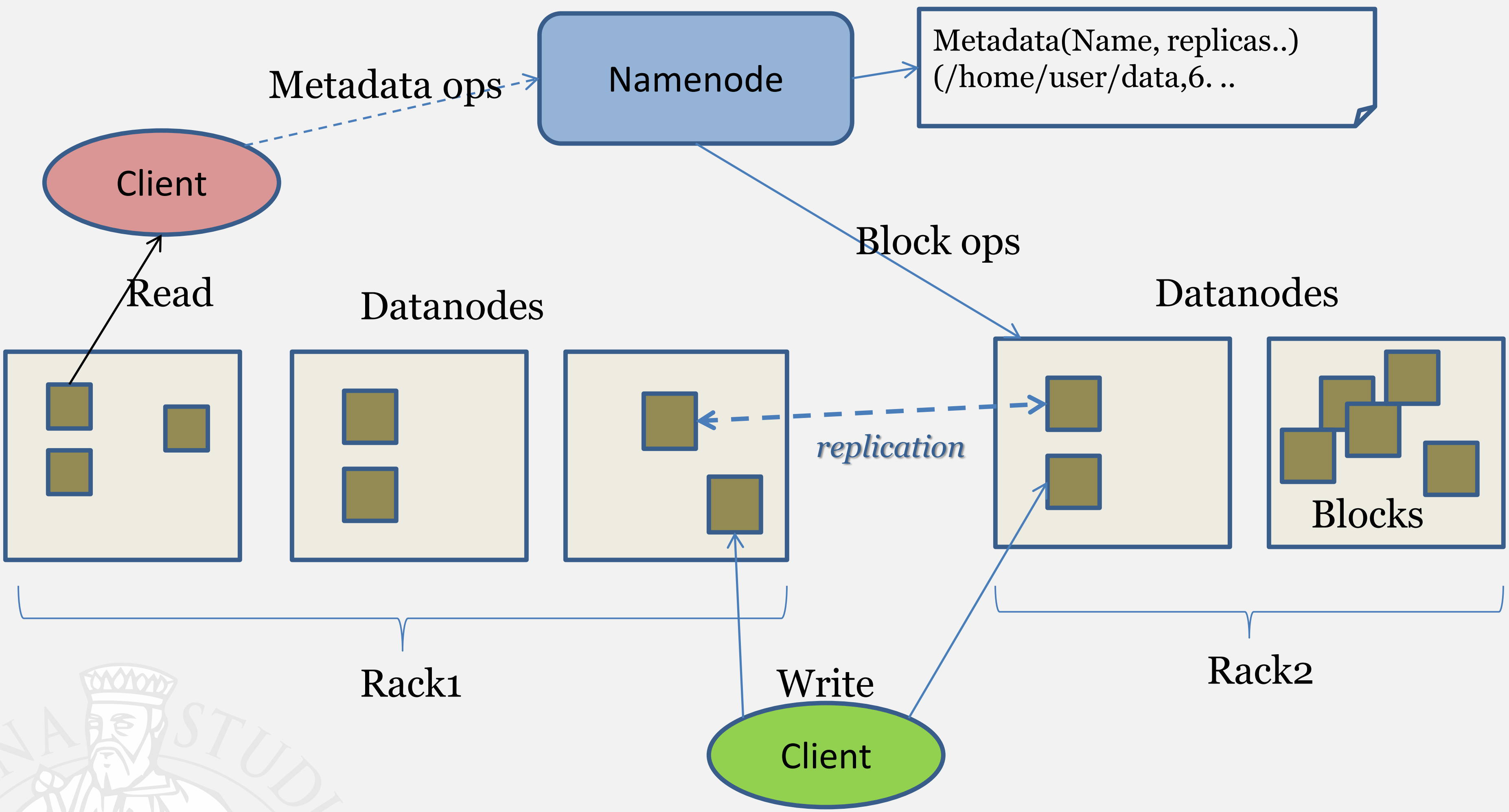
Hadoop Ecosystem



HDFS Architecture



HDFS Architecture



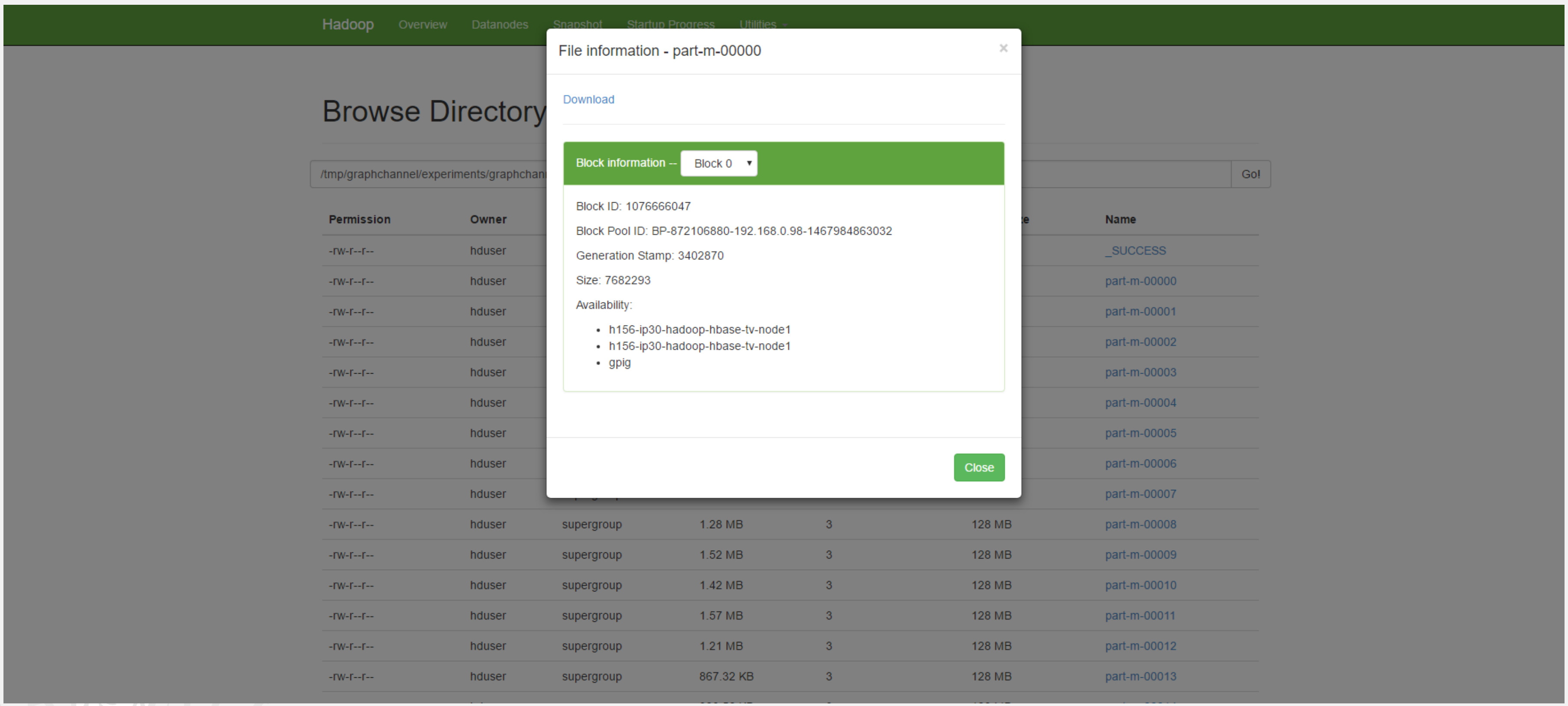
Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of **DataNodes** usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.
- ***New Paradigm: Data Locality → Moving Computation is Cheaper than Moving Data***

File system Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system is recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

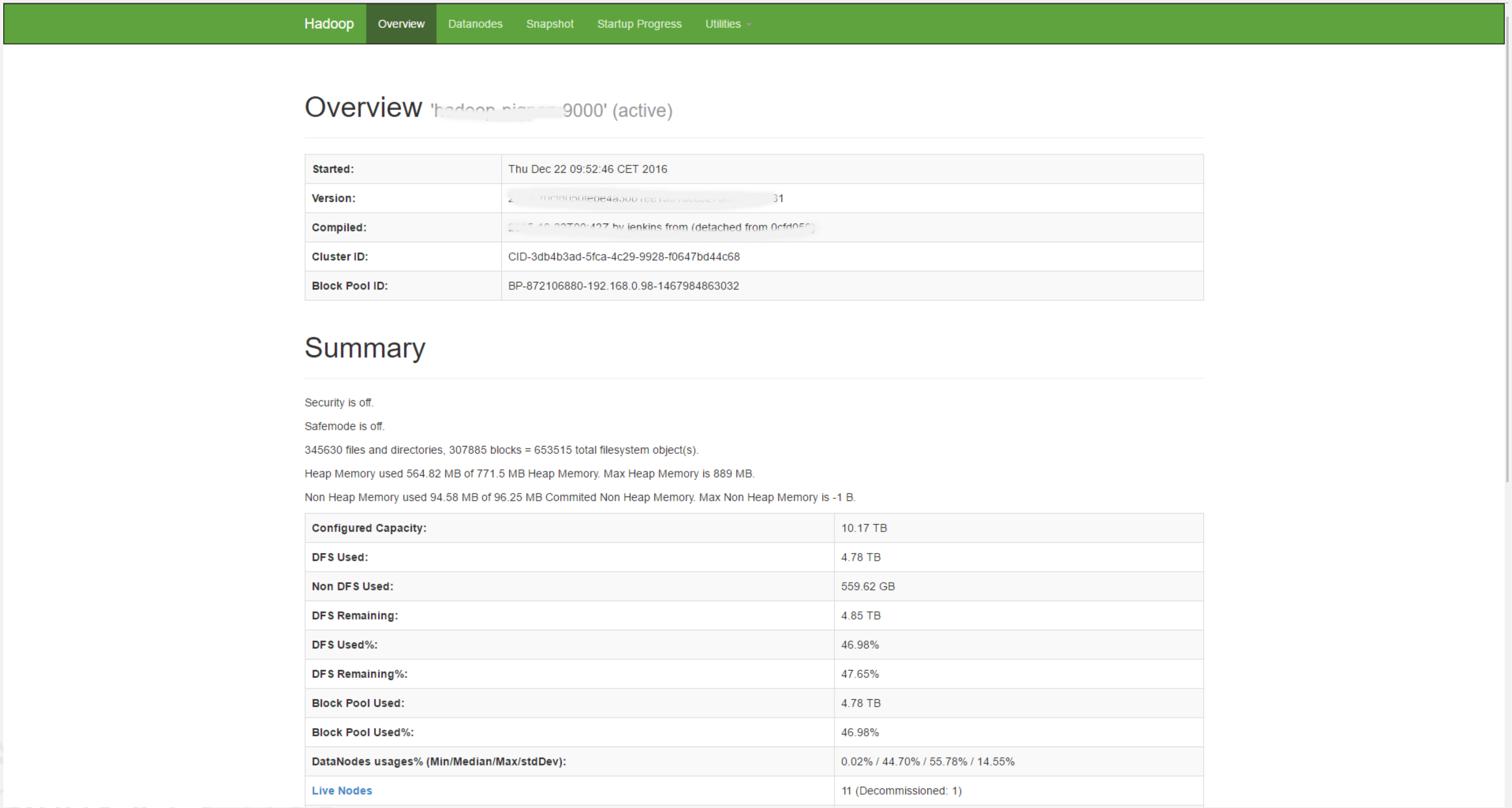
Example


Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Browse Directory
/tmp/graphchannel/experiments/graphchan
Go!
Permission
Owner
Name
_SUCCESS
part-m-00000
part-m-00001
part-m-00002
part-m-00003
part-m-00004
part-m-00005
part-m-00006
part-m-00007
-rw-r--r-- hduser supergroup 1.28 MB 3 128 MB part-m-00008
-rw-r--r-- hduser supergroup 1.52 MB 3 128 MB part-m-00009
-rw-r--r-- hduser supergroup 1.42 MB 3 128 MB part-m-00010
-rw-r--r-- hduser supergroup 1.57 MB 3 128 MB part-m-00011
-rw-r--r-- hduser supergroup 1.21 MB 3 128 MB part-m-00012
-rw-r--r-- hduser supergroup 867.32 KB 3 128 MB part-m-00013
File information - part-m-00000
Download
Block information -- Block 0
Block ID: 1076666047
Block Pool ID: BP-872106880-192.168.0.98-1467984863032
Generation Stamp: 3402870
Size: 7682293
Availability:
h156-ip30-hadoop-hbase-tv-node1
h156-ip30-hadoop-hbase-tv-node1
gpig
Close

Interaction with hdfs

- Web interface
  - Web application bundled with hadoop
  - Hue
- Console Interface
- Java API

Basic Interface


Hadoop | Overview | Datanodes | Snapshot | Startup Progress | Utilities

Overview '...9000' (active)

Started:	Thu Dec 22 09:52:46 CET 2016
Version:	[illegible]
Compiled:	... by jenkins from (detached from ...)
Cluster ID:	CID-3db4b3ad-5fca-4c29-9928-f0647bd44c68
Block Pool ID:	BP-872106880-192.168.0.98-1467984863032

Summary

Security is off.

Safemode is off.

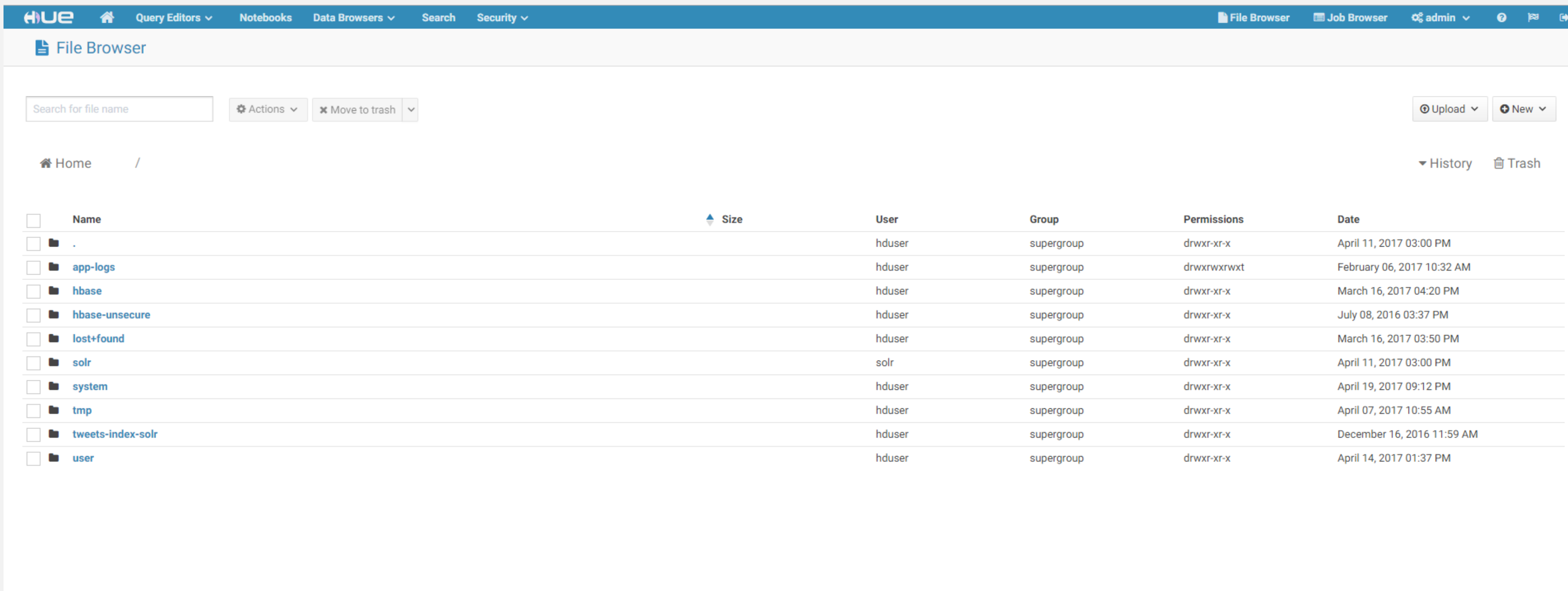
345630 files and directories, 307885 blocks = 653515 total filesystem object(s).

Heap Memory used 564.82 MB of 771.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 94.58 MB of 96.25 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	10.17 TB
DFS Used:	4.78 TB
Non DFS Used:	559.62 GB
DFS Remaining:	4.85 TB
DFS Used%:	46.98%
DFS Remaining%:	47.65%
Block Pool Used:	4.78 TB
Block Pool Used%:	46.98%
DataNodes usages% (Min/Median/Max/stdDev):	0.02% / 44.70% / 55.78% / 14.55%
Live Nodes	11 (Decommissioned: 1)

Advanced Interface



Console Interface: some commands

- Create directory
  hdfs dfs -mkdir <hdfs_path>
- List directory
  hdfs dfs -ls <hdfs_path>

- Delete file
  hdfs dfs –rm <hdfs_path>/file
- Delete directory
  hdfs dfs –rm –r –f <hdfs_path>

- Upload file to hdfs
  hdfs dfs –put file.txt <hdfs_path>
- Download file from hdfs
  hdfs dfs –get <hdfs_path>/file.txt

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- The Hertbeat report contains all information about metadata on each Datanode
- BlockReport contains all the blocks information on a Datanode.

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- 4GB of local RAM is sufficient to support the above data structures that represent the huge number of files and directories.
- When the Namenode starts up it gets the FsImage and Editlog from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

HDFS maintance (Cli)

- Reports basic filesystem information and statistics. Optional flags may be used to filter the list of displayed DataNodes.

  hdfs dfsadmin –report

- Re-read the hosts and exclude files to update the set of Datanodes that are allowed to connect to the Namenode and those that should be decommissioned or recommissioned.

  hdfs dfsadmin –refreshNodes

HDFS maintance (Cli)

- HDFS data might not always be be placed uniformly across the DataNode. One common reason is addition of new DataNodes to an existing cluster. While placing new blocks (data for a file is stored as a series of blocks), NameNode considers various parameters before choosing the DataNodes to receive these blocks. Some of the considerations are:
  - Policy to keep one of the replicas of a block on the same node as the node that is writing the block.
  - Need to spread different replicas of a block across the racks so that cluster can survive loss of whole rack.
  - One of the replicas is usually placed on the same rack as the node writing to the file so that cross-rack network I/O is reduced.
  - Spread HDFS data uniformly across the DataNodes in the cluster.

hdfs balancer -policy blockpool

Robustness

- Primary objective of HDFS is to store data reliably in the presence of failures.
- Three common failures are: Namenode failure, Datanode failure and network failure.

DataNode failure and heartbeat

- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks Datanodes without Hearbeat and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Data Integrity

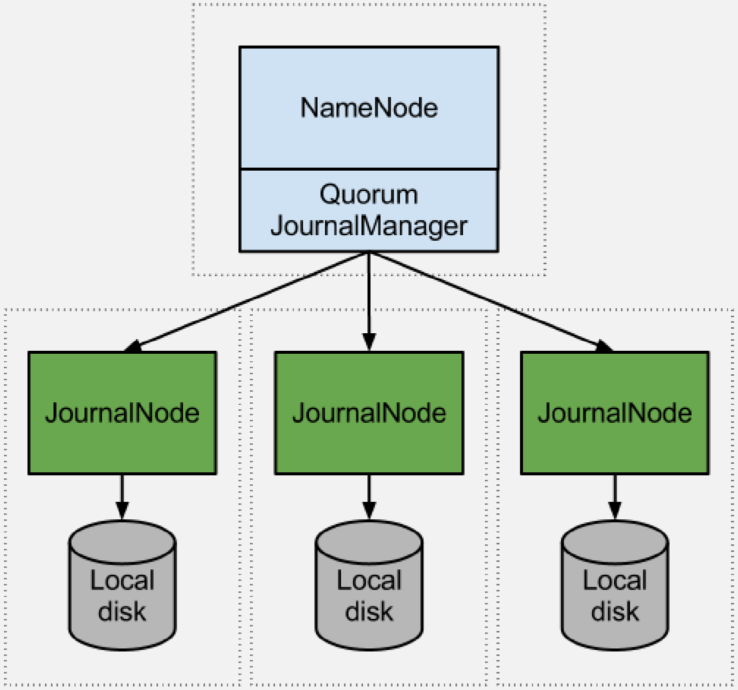
- Consider a situation: a block of data fetched from Datanode arrives corrupted.
- This corruption may occur because of faults in a storage device, network faults, or buggy software.
- A HDFS client creates the checksum of every block of its file and stores it in hidden files in the HDFS namespace.
- When a clients retrieves the contents of file, it verifies that the corresponding checksums match.
- If does not match, the client can retrieve the block from a replica.

Metadata Disk Failure

- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Meta-data is not data-intensive.
- Prior Hadoop 2.x -> The Namenode could be single point failure: automatic failover is NOT supported!
- Hadoop bigger > 2.x has HA (High-Availability) Feature
  - nfs share
  - Journal Node with Zookeeper

Hadoop HA

- Journal nodes are distributed system to store edits.
- Active Namenode as a client writes edits to journal nodes and commit only when its replicated to all the journal nodes in a distributed system.
- Standby NN need to read data from edits to be in sync with Active one.
- ZKFC will make sure that only one Namenode should be active at a time.



Hadoop HA Cont.

- However, when a failover occurs, it is still possible that the previous Active NameNode could serve read requests to clients, which may be out of date until that NameNode shuts down when trying to write to the JournalNodes.

we should configure fencing methods even when using the Quorum Journal Manager.

Fencing Method

- Journal manager uses **epoc numbers**.
- Epoc numbers are integer which always gets increased and have unique value once assigned. Namenode generate epoc number using simple algorithm and uses it while sending RPC requests to the QJM.
  - When you configure Namenode HA, the first Active Namenode will get epoc value 1. In case of failover or restart, epoc number will get increased. The Namenode with higher epoc number is considered as newer than any Namenode with earlier epoc number.
  - Quorum journal manager stores epoc number locally which are called **promised epoc**. Whenever JournalNode receives RPC request along with epoc number from Namenode, it compares the epoch number with promised epoch. If request is coming from newer node which means epoc number is greater than promised epoc then it **records new epoc number as promised epoc**. If the request is coming from Namenode with older epoc number, then QJM simply rejects the request.

Data Blocks

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.

Staging (contd.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion;

Replication Pipelining

- When the client receives response from Namenode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Thus data is pipelined from Datanode to the next.

Application Programming Interface

- HDFS provides Java API for application to use.
- Python access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

An example of read write java program to hdfs

Prerequesite

I. The classpath contains the Hadoop JAR files and its client-side dependencies.

II. The hadoop configuration files on the classpath

III. Log4J on the classpath along with a **log4.properties** resource, or commons-logging preconfigured to use a different logging framework.

First step

Create a FileSystem instance by passing a new Configuration object.

```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
```

Next ?

Given an input/output file name as string, we construct inFile/outFile Path objects. Most of the FileSystem APIs accepts Path objects.

```
Path inFile = new Path(argv[0]);
Path outFile = new Path(argv[1]);
```

Some sanitazing (Validate the input/output paths before reading/writing.)

```
if (!fs.exists(inFile))
      printAndExit("Input file not found");
if (!fs.isFile(inFile))
      printAndExit("Input should be a file");
if (fs.exists(outFile))
      printAndExit("Output already exists");
```

Final step

i. Open inFile for reading.

```
FSDataInputStream in = fs.open(inFile);
```

ii. Open outFile for writing.

```
FSDataOutputStream out = fs.create(outFile);
```

iii. Read from input stream and write to output stream until EOF.

```
while ((bytesRead = in.read(buffer)) > 0) {
out.write(buffer, 0, bytesRead); }
```

Close the streams when done.

```
in.close();
out.close();
```

Compile

- Mkdir <DIR_for_jar>
- Javac –cp $(hadoop classpath) –d <DIR_for_jar> <ClassNameFile.java>
- Jar cvfe <Dest>.jar org/disit/ClassName –C <DIR_for_jar> .

launch

- Hadoop jar <Dest>.jar

Apache Hadoop
Map Reduce

Parallel Computing@hadoop

Issue

- Fundamental of map reduce in practice!

- Working example bundle with hadoop documentation: wordcount

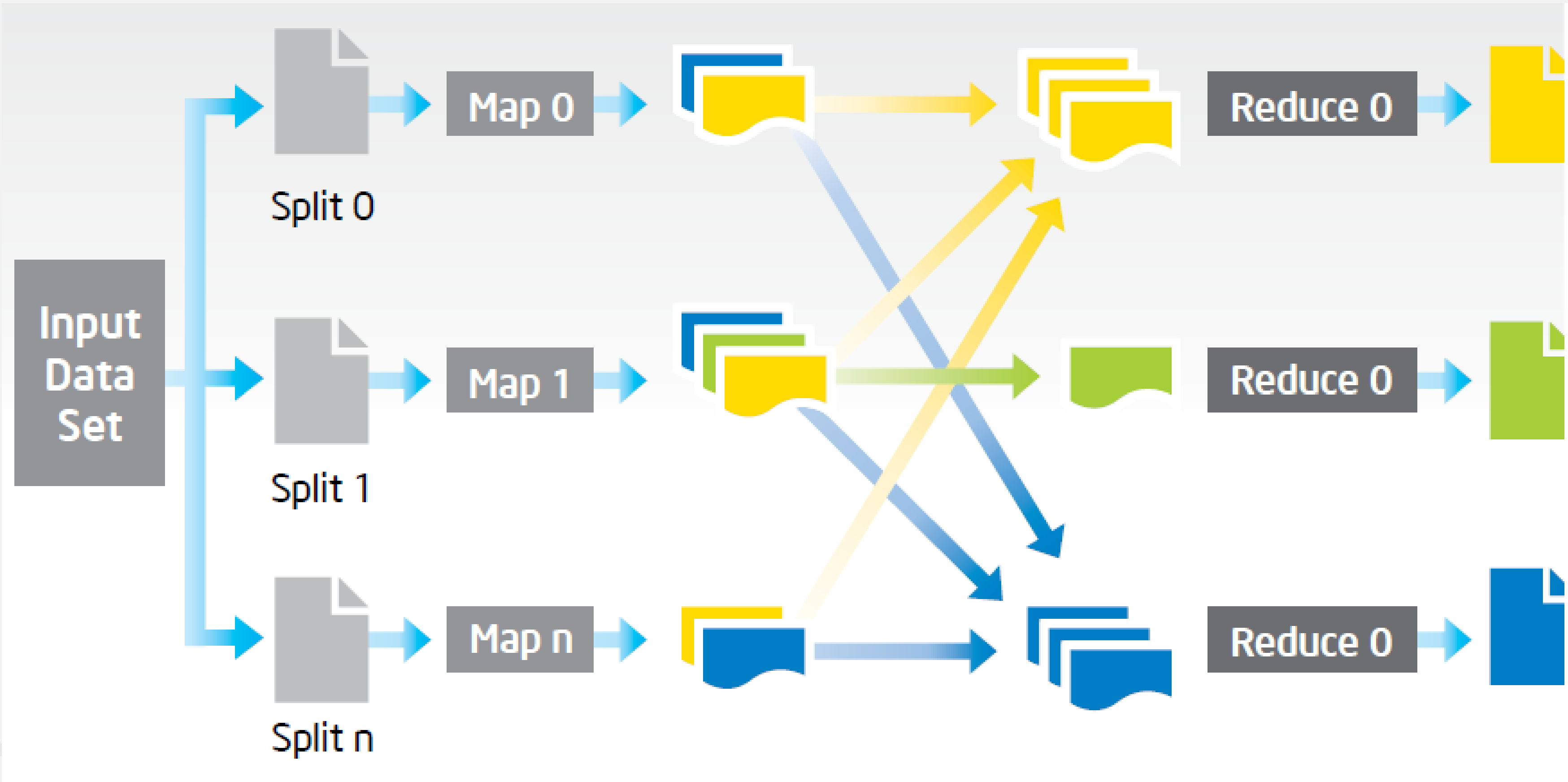
Map Reduce paradigm

- **MapReduce** is the heart of Hadoop. It is this programming paradigm that allows for **massive scalability** across hundreds or thousands of servers in a Hadoop cluster.

- The **Map function** in the master node takes the input, partitions it into smaller sub-problems, and distributes them to operational nodes.
- In the **Reduce function**, the root/master node take the outputs/results of all the sub-problems, combining them to output the answer to the problem it is trying to solve.

- See Yahoo! Hadoop tutorial: https://developer.yahoo.com/hadoop/tutorial/index.html

Review Map/Reduce Flow



Source: https://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf

Map Reduce paradigm

- Mapper
  - map(inputs) → list(key1, value1a)
    list(key1, value1b)
    [. . . .]
- Reducer:
  - reduce (key1, list (value1a, value1b,…)) → list(key2, value2)
- When the mapping phase has completed, the intermediate (key, value) pairs must be exchanged (***Shuffle & Sort*** phase) among machines to send all values with the same key to a single reducer.

Real Word Count Example

- Simple word counting program
- Document1.txt

```
Queste sono le slide di Pippo XYZ
```

- Document2.txt

```
Le slide del corso di Big Data Architecture
```

- Expected Output:

```
Queste          1
sono            1
le              1
slide           2
di              2
Pippo           1
XYZ             1
Le              1
del             1
corso           1
Big             1
Data            1
Architecture    1
```

Real Word Count Example

- *mapper (filename, file-contents):*
- ***for each*** *word* ***in*** *file-contents:*
- ***emit*** *(word, 1)*
-
- *reducer (word, values):*
- *sum = 0*
- ***for each*** *value* ***in*** *values:*
- *sum = sum + value*
- ***emit*** *(word, sum)*

Real Word Count Example

- WordCount.java → simple word counting Java program to be executed trhough MapReduce in a Hadoop Cluster.

- SampleDoc.txt → Input text file (programma del corso di Sistemi Distribuiti):

Programma del corso
dettagli e slide possono essere ottenuti da social network, smart city.
Overview parte 0, ver:0.6: una vista generale al corso
Introduzione (Parte 1, ver:2.0): (versione 2.4) Cosa sono i sistemi distribuiti, Tecnologie dei sistemi distribuiti, Internet e sua Evoluzione, Intranet, Penetrazione di internet, Crescita, Sistemi Mobili, Condivisione delle risorse, Web Server and Web Services, Caratteristiche: Eterogenei, aperti, sicuri, trasparenti, architetture, n-tier.
XML (parte 1b): fondamenti di XML, uso avanzato dell'XML
PHP e Drupal: Parte 1cI, Parte 1cII, architetture web server, programmazione in PHP, costrutti dell linguaggio, operatori, get/post, esempi; Parte II: Content Management Systems, CMS, moduli, call back, ruoli, etc. WEB services e REST remote invocation via Web Services and REST architectures, strumenti per i WEB services, verifica, SOAP.

[…]

NLP in Hadoop – A Real Case

```java
package org.disit;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper  extends Mapper<Object, Text,
Text, IntWritable>{
      //  [. . .]
  }

  public static class IntSumReducer extends Reducer<Text, IntWritable,
   Text, IntWritable> {
      //  [. . .]
  }
```

```java
public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);

}
```

NLP in Hadoop – A Real Case

Map Class

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

      StringTokenizer itr = new StringTokenizer(value.toString());     // Tokenizzazione del file di testo

      while (itr.hasMoreTokens()) {

          word.set(itr.nextToken());
          context.write(word, one);

      }

    }

  }
```

NLP in Hadoop – A Real Case

Reduce Class

```
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

      int sum = 0;

      for (IntWritable val : values) {

         sum += val.get();             /* Somma i valori unitari del conteggio di ogni singola parola
                                       iterando su tutte le parole            */
      }

      result.set(sum);
      context.write(key, result);

    }

 }
```

Real Word Count Example

- Make executable jar:

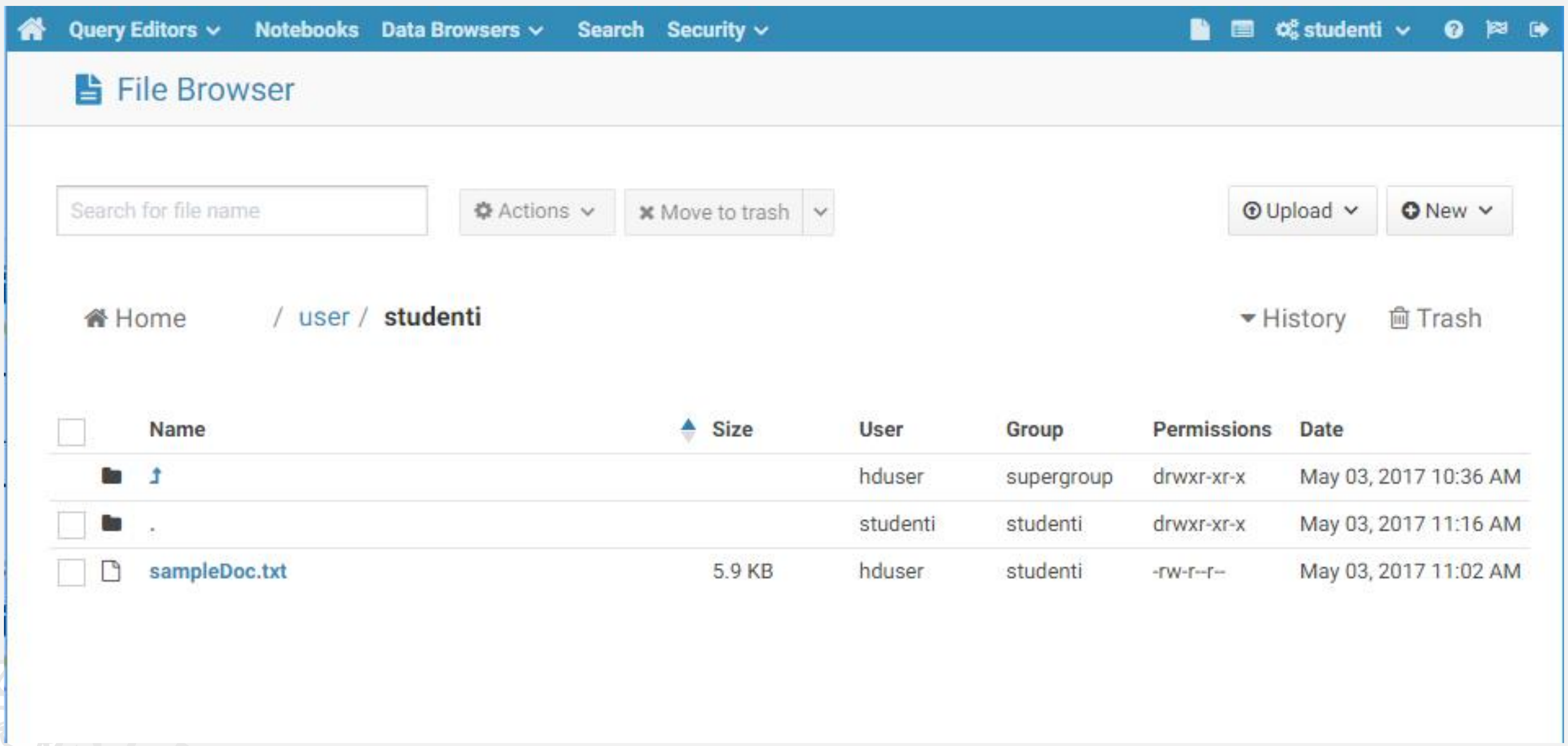
```
$ javac -cp $(hadoop classpath) -d ./jar WordCount.java
$ jar cvfe wordCount.jar org/disit.WordCount -C ./jar .
```

- Copy input text file `sampleDoc.txt` in HDFS: `hadoop fs -copyFromLocal` *`<file_to_be_copied> <HDFS_Folder_Path>`*

```
$ hadoop fs -copyFromLocal sampleDoc.txt /users/studenti/
```

Real Word Count Example

- Browsing HDFS Filesystem → http://<dedicatedHueHostIP>:8000



Real Word Count Example

- Execute the Word Count program (wc.jar) in HDFS:

```
hadoop jar <jarFile.jar> <input_File_HDFS_Path> <output_HDFS_Folder>

$ hadoop jar wordCount.jar /user/studenti/sampleDoc.txt /user/studenti/output
```

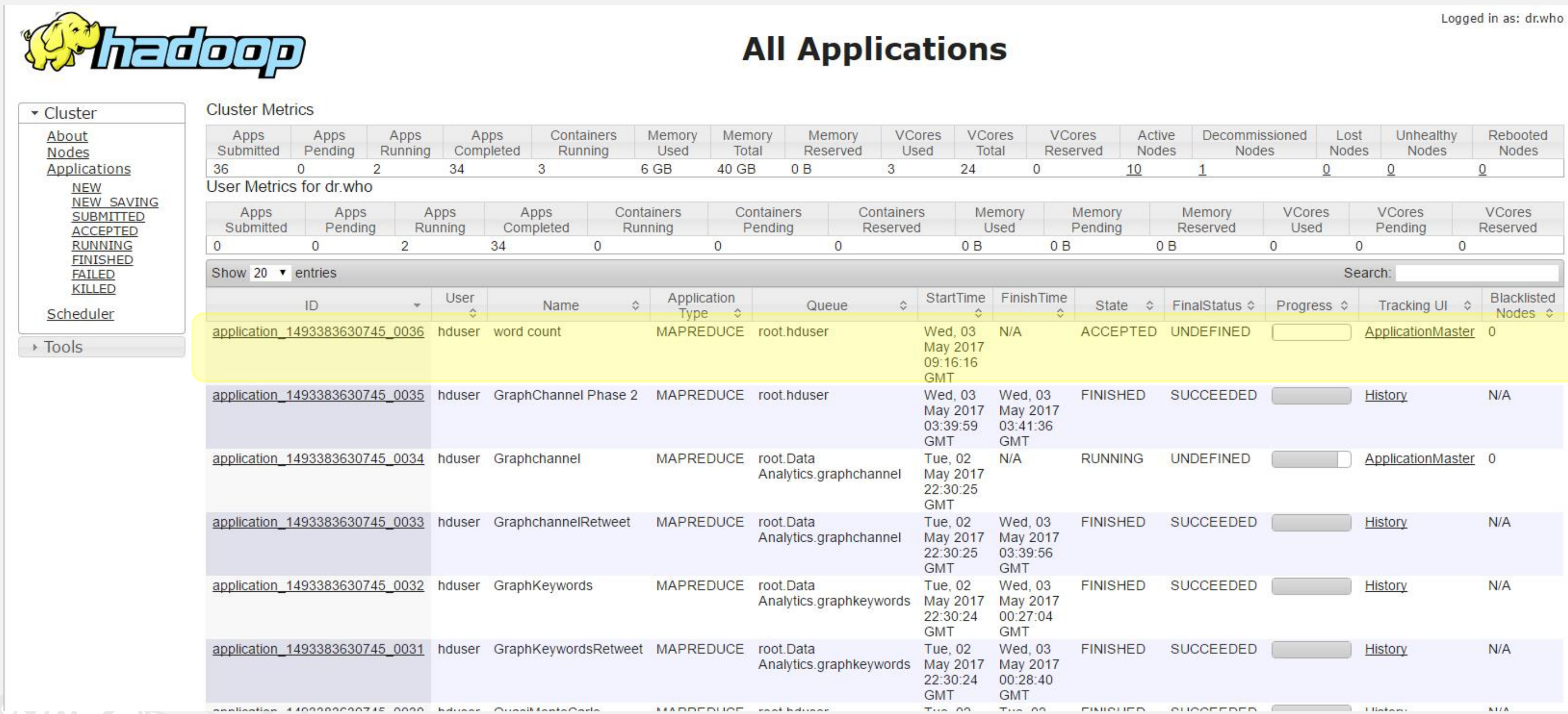
```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-49-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

412 packages can be updated.
285 updates are security updates.

New release '16.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

You have new mail.
Last login: Wed May  3 12:50:42 2017 from 192.168.0.242
hduser@hadoop-pigpen:~$ cd /
hduser@hadoop-pigpen:/$ cd srv/hadoop/share/hadoop/mapreduce/
hduser@hadoop-pigpen:/srv/hadoop/share/hadoop/mapreduce$ hadoop jar wordCount.jar /user/studenti/sampleDoc.txt /user/studenti/output
```

Real Word Count Example

```
hduser@hadoop-pigpen:/srv/hadoop/share/hadoop/mapreduce$ hadoop jar wordCount.jar /user/studenti/sampleDoc.txt /user/studenti/output
17/05/03 13:36:49 INFO client.RMProxy: Connecting to ResourceManager at /192.168.0.98:8050
17/05/03 13:36:49 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and
execute your application with ToolRunner to remedy this.
17/05/03 13:36:50 INFO input.FileInputFormat: Total input paths to process : 1
17/05/03 13:36:51 INFO mapreduce.JobSubmitter: number of splits:1
17/05/03 13:36:51 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1493383630745_0039
17/05/03 13:36:52 INFO impl.YarnClientImpl: Submitted application application_1493383630745_0039
17/05/03 13:36:52 INFO mapreduce.Job: The url to track the job: http://hadoop-pigpen:8080/proxy/application_1493383630745_0039/
17/05/03 13:36:52 INFO mapreduce.Job: Running job: job_1493383630745_0039
17/05/03 13:37:01 INFO mapreduce.Job: Job job_1493383630745_0039 running in uber mode : false
17/05/03 13:37:01 INFO mapreduce.Job:  map 0% reduce 0%
17/05/03 13:37:47 INFO mapreduce.Job:  map 100% reduce 0%
17/05/03 13:37:55 INFO mapreduce.Job:  map 100% reduce 33%
17/05/03 13:37:56 INFO mapreduce.Job:  map 100% reduce 100%
17/05/03 13:37:58 INFO mapreduce.Job: Job job_1493383630745_0039 completed successfully
17/05/03 13:37:58 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=5605
                FILE: Number of bytes written=442041
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6164
                HDFS: Number of bytes written=5319
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=6
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=3
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=42617
                Total time spent by all reduces in occupied slots (ms)=34252
                Total time spent by all map tasks (ms)=42617
                Total time spent by all reduce tasks (ms)=17126
                Total vcore-seconds taken by all map tasks=42617
                Total vcore-seconds taken by all reduce tasks=17126
                Total megabyte-seconds taken by all map tasks=87279616
                Total megabyte-seconds taken by all reduce tasks=70148096
        Map-Reduce Framework
                Map input records=15
                Map output records=806
                Map output bytes=9272
                Map output materialized bytes=5593
                Input split bytes=117
                Combine input records=806
                Combine output records=513
                Reduce input groups=513
                Reduce shuffle bytes=5593
                Reduce input records=513
                Reduce output records=513
                Spilled Records=1026
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=1894
```

Real Word Count Example

- Monitoring Running Apps and Resoruces → http://<masterNodeHostIP>:8080



Logged in as: dr.who

All Applications

- Cluster
  - About
  - Nodes
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
36	0	2	34	3	6 GB	40 GB	0 B	3	24	0	10	1	0	0	0

User Metrics for dr.who

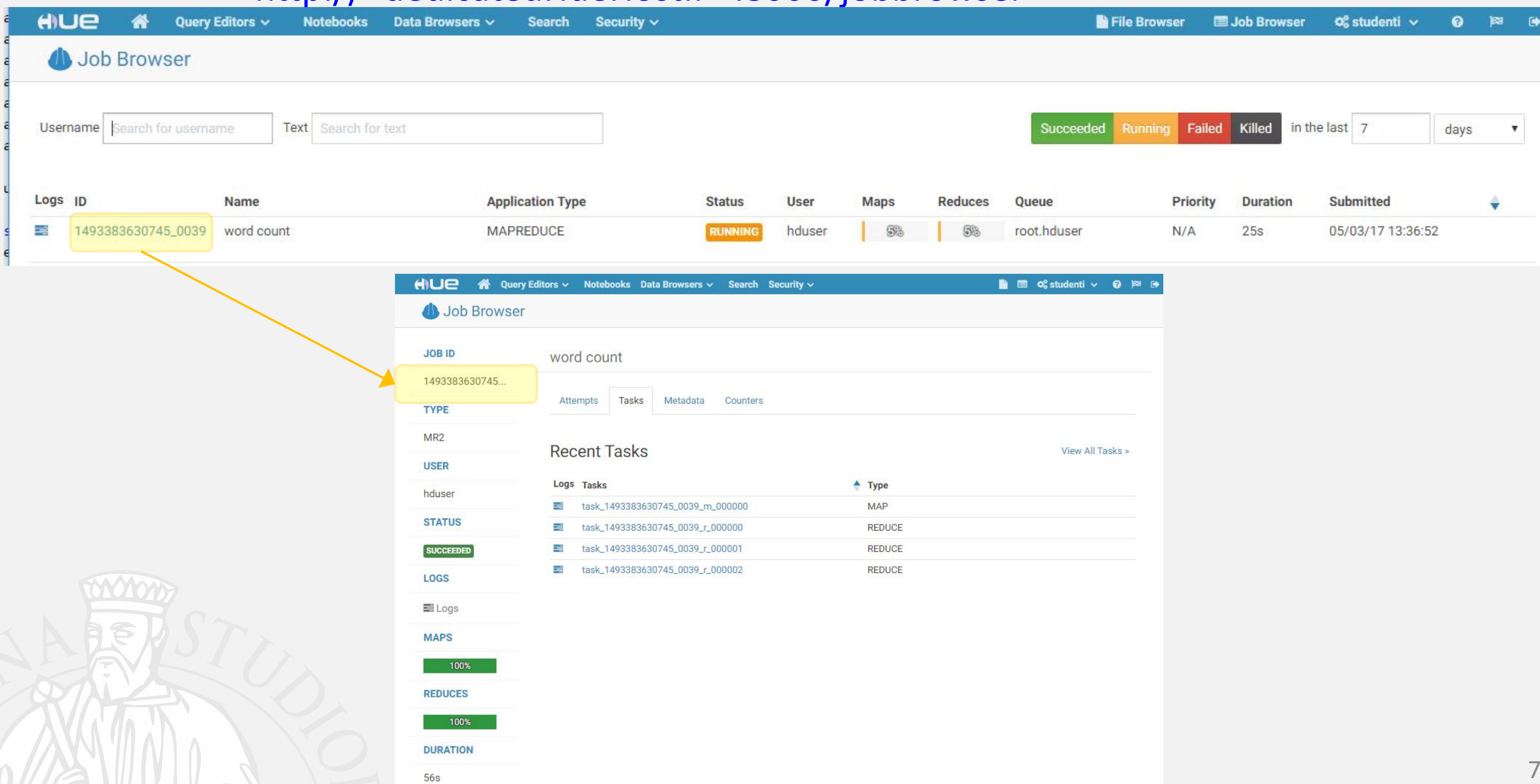
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	2	34	0	0	0	0 B	0 B	0 B	0	0	0

Show 20 entries   Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1493383630745_0036	hduser	word count	MAPREDUCE	root.hduser	Wed, 03 May 2017 09:16:16 GMT	N/A	ACCEPTED	UNDEFINED		ApplicationMaster	0
application_1493383630745_0035	hduser	GraphChannel Phase 2	MAPREDUCE	root.hduser	Wed, 03 May 2017 03:39:59 GMT	Wed, 03 May 2017 03:41:36 GMT	FINISHED	SUCCEEDED		History	N/A
application_1493383630745_0034	hduser	Graphchannel	MAPREDUCE	root.Data Analytics.graphchannel	Tue, 02 May 2017 22:30:25 GMT	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1493383630745_0033	hduser	GraphchannelRetweet	MAPREDUCE	root.Data Analytics.graphchannel	Tue, 02 May 2017 22:30:25 GMT	Wed, 03 May 2017 03:39:56 GMT	FINISHED	SUCCEEDED		History	N/A
application_1493383630745_0032	hduser	GraphKeywords	MAPREDUCE	root.Data Analytics.graphkeywords	Tue, 02 May 2017 22:30:24 GMT	Wed, 03 May 2017 00:27:04 GMT	FINISHED	SUCCEEDED		History	N/A
application_1493383630745_0031	hduser	GraphKeywordsRetweet	MAPREDUCE	root.Data Analytics.graphkeywords	Tue, 02 May 2017 22:30:24 GMT	Wed, 03 May 2017 00:28:40 GMT	FINISHED	SUCCEEDED		History	N/A
application_1493383630745_0030	hduser	QuasiMonteCarlo	MAPREDUCE	root.hduser	Tue, 02	Tue, 02	FINISHED	SUCCEEDED		History	N/A

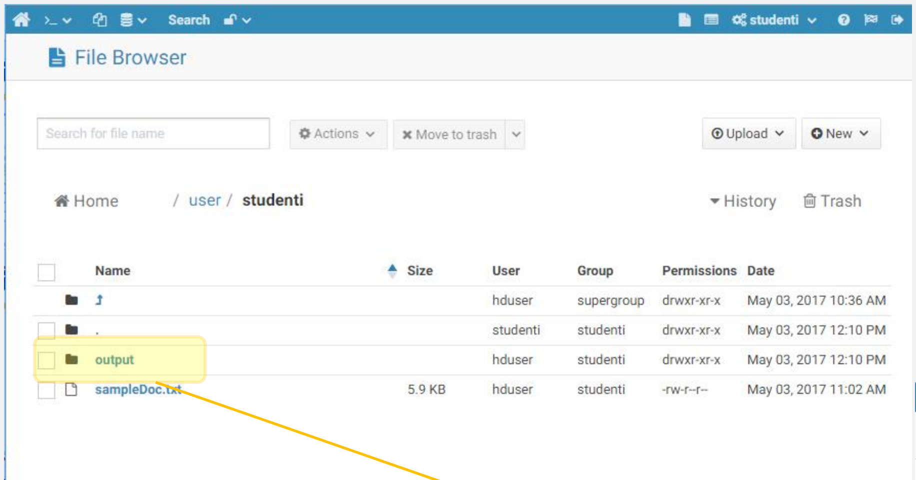
Real Word Count Example

- Monitoring Running Apps and Resoruces → http://<dedicatedHueHostIP>:8000/jobbrowser



Real Word Count Example

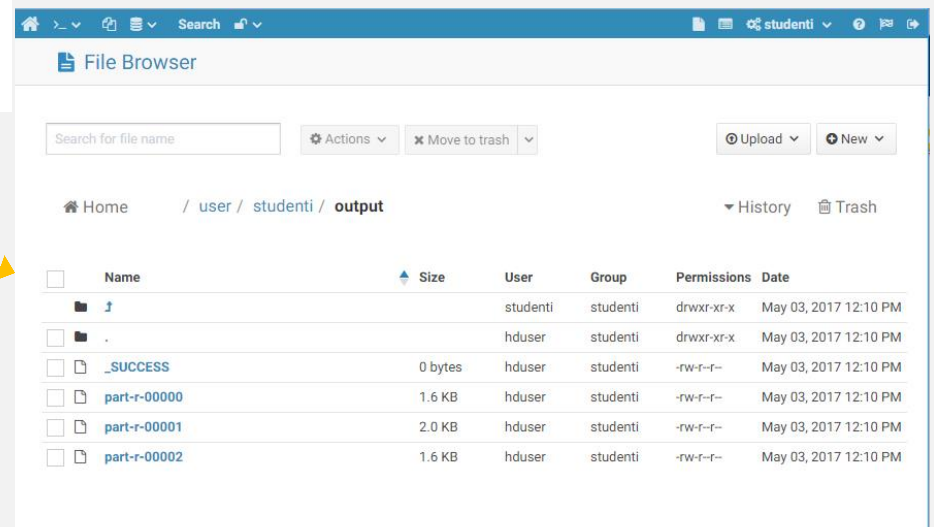
- Browsing the Output in HDFS:



File Browser

Home / user / studenti

Name	Size	User	Group	Permissions	Date
↥		hduser	supergroup	drwxr-xr-x	May 03, 2017 10:36 AM
.		studenti	studenti	drwxr-xr-x	May 03, 2017 12:10 PM
output		hduser	studenti	drwxr-xr-x	May 03, 2017 12:10 PM
sampleDoc.txt	5.9 KB	hduser	studenti	-rw-r--r--	May 03, 2017 11:02 AM

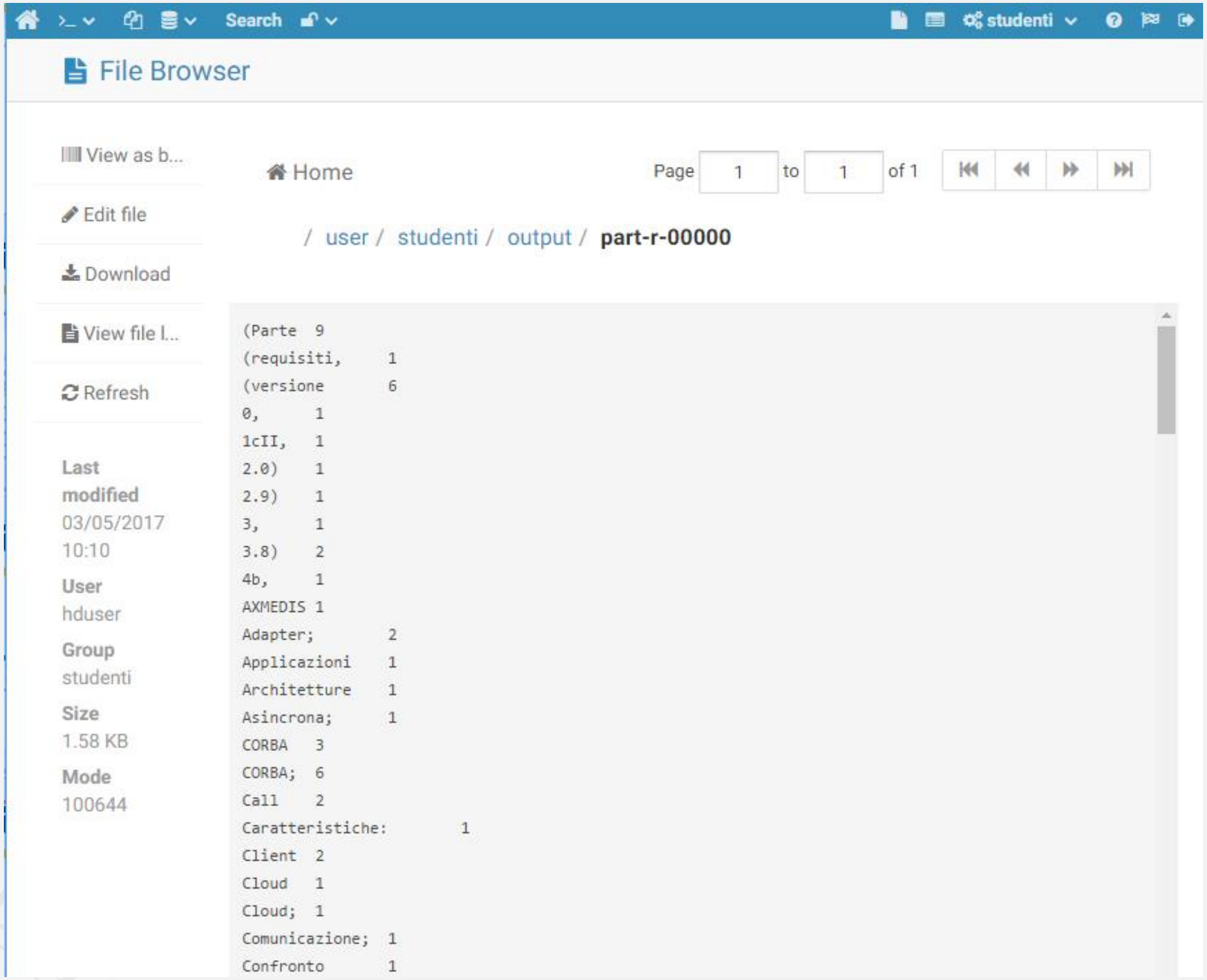


File Browser

Home / user / studenti / output

Name	Size	User	Group	Permissions	Date
↥		studenti	studenti	drwxr-xr-x	May 03, 2017 12:10 PM
.		hduser	studenti	drwxr-xr-x	May 03, 2017 12:10 PM
_SUCCESS	0 bytes	hduser	studenti	-rw-r--r--	May 03, 2017 12:10 PM
part-r-00000	1.6 KB	hduser	studenti	-rw-r--r--	May 03, 2017 12:10 PM
part-r-00001	2.0 KB	hduser	studenti	-rw-r--r--	May 03, 2017 12:10 PM
part-r-00002	1.6 KB	hduser	studenti	-rw-r--r--	May 03, 2017 12:10 PM

Real Word Count Example



Advanced NLP in Hadoop Fashion

Main Class Declaration

```
package principale;
import gate.Corpus;
import gate.creole.ExecutionException;
import gate.creole.ResourceinstantiationException;
import gate.util.GateException;
import java.io.IOException;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobPriority;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.TextinputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class KeywordExtraction
{
      //        [...]
}
```

Advanced NLP in Hadoop Fashion

Main Implementation

```
public static void main(String[] args) throws Exception
{
   Configuration conf = new Configuration();
   conf.set("mapred.Job.priority", JobPriority.VERY_HIGH.toString());
   Path localGateTreeTaggerApp = new Path("/home/hduser/GATETreeTagger.zip");
   Path hdfsGateTreeTaggerApp = new Path("/tmp/GATE-app.zip");
   Path inputFile = new Path("/home/hduser/input6.txt");
   Path hdfsInputFile =new Path("tmp/inputFile.txt");
   FileSystem fs = FileSystem.get(conf);
   fs.copyFromLocalFile(localGateTreeTaggerApp, hdfsGateTreeTaggerApp);
   DistributedCache.addCacheArchive(hdfsGateTreeTaggerApp.toUri(), conf);
   fs.copyFromLocalFile(inputFile, hdfsinputFile);
   DistributedCache.addCacheArchive(hdfsinputFile.toUri(), conf);
   Job job = new Job(conf);
   job.setJarByClass(KeywordExtraction.class);
   job.setJobName("Keyword Extraction"):
   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);
   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(Text.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(NullWritable.class);
   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);
   TextinputFormat.addInputPath(job, new Path("/tmp/inputFile.txt"));
   TexcOutpucFormat.setOutputPath(job, new Path("mnt/bigdsk/new_data "+ args[0]));
   boolean success - job.waitForCompletion(true);
   if (success) {
      FileSystem.get(job.getConfiguration ()).deleteOnExit(hdfsGateTreeTaggerApp);
   }
   System.exit(success? 0 :-1);
}
```

general architecture GATE for text engineering

Advanced NLP in Hadoop Fashion

Map Class Implementation

```
public static class Map
    extends Mapper<LongWritable, Text, Text, Text>
{
    public void map(LongWritable key, Text value, Context context)
       throws IOException, InterruptedException
    {
       try
       {
          String line = value.toString();
          String domainString = "";
          String parsedText = "";
          if (line.contains(" TEXT:: "))
          {
              domainString = getHost(line.split(" TEXT:: ")[0].split("URL:: ")[1]);
              parsedText = "";
              if (!line.endsWith(" TEXT:: ")) {
                  parsedText = line.split(" TEXT:: ")[1];
              }
          else {
              domainString line;
              parsedText = "";
          }
          context.write(new Text(domainString), new Text(parsedText));
    }
    catch (IOException e)
    {
          e.printStackTrace();
    }
    catch (InterruptedException e)
    {
          e.printStackTrace();
    }
```

URL:: *http://www.domain.com* ***TEXT::*** *this is a text…*
URL:: *http://dom.org* ***TEXT::*** *this is another text from…*

```
public String getHost(String url)
{
   if ((url == null) || (url.length() == 0)) {
       return "";
   }
   int doubleslash = url.indexOf("//");
   if (doubleslash == -1) {
       doubleslash = 0;
   } else {
       doubleslash += 2;
   }
   int end = url.indexOf('/', doubleslash);
   end = end >= 0 ? end : url.length();
   int port= url.indexOf(':', doubleslash);
   end = (port > 0) && (port < end) ? port : end;
   return url.substring(doubleslash, end);
}
```

Advanced NLP in Hadoop Fashion

Reduce Class Implementation

```
public static class Reduce
    extends Reducer<Text, Text, Text, NulllWritable>
{
    private static GATEApplication gate;

    protected void setup()
        throws IOException, InterruptedException
    {
        if (gate == null)
        {
            Configuration c = context.getConfiguration();
            Path[] localCache = DistributedCache.getLocalCacacheArchives(c);
            try
            {
                gate= new GATEApplication(localCache[0].toString());
            }
            catch (GateException e)
            {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Advanced NLP in Hadoop Fashion

Reduce Class Implementation

```
public void reduce(Text key, Iterable<Text> values, Context context)
   throws IOException, Ineerrupted.Exception
{
   for (Text v : values)
   String parsedText = v.toString();
   try
   {
       String POSKeywords = gate.POSKeywordsAnnotation(parsedText);
       SimpleDateFormat sdf = new SimpleDateFormat();
       sdf.applyPattern("yyyy-MM-dd");
       String dataStr = sdf.format(new Date());
       String[] lines = POSKeywords.split(System.getProperty ("line.separator"));
       for (int i = 0; i < lines.length; i++) {
           if (lines[i] .cont.ai.ns("KPH"))
           {
              String[] keyphrase = lines(i].split(" KPH");
              String st = key + ", " + keyphrase [0] + " (KPH), " + dataStr;
              Text t = new Text();
              t.set(st);
              context.write(t, NullWritable.get());
           }
           else if ((lines[i].contains("NOM")) || (lines[i].contains("ADJ") || (lines[i].contains("VER")))
           {
              String[] keyword = lines[i] .split(" ");
              String st = key + ", " + keyword[0] + " (" + keyword[1] + ") " + dataStr;
              Text t = new Text();
              t.set(st);
              context.write(t, NullWritable.get());
           }
       }
   }
   catch (ResourceInstantiationException localResourceInstantiationException) {}
   catch (ExecutionException e)
   {
       System.out.println(key + "execution exception" + e + "\n");
       gate.corpus.clear();
   }
   catch (MalformedURLException localMalformedURLException) {}
   catch (IOEexception e)
   {
       System.out .println (key + "IO exception" + e + "\n");
       gate.corpus.clear();
   }
}
```