MapReduce

Fabio Cumbo

Corso: Introduzione ai Big Data – A.A. 2016/2017

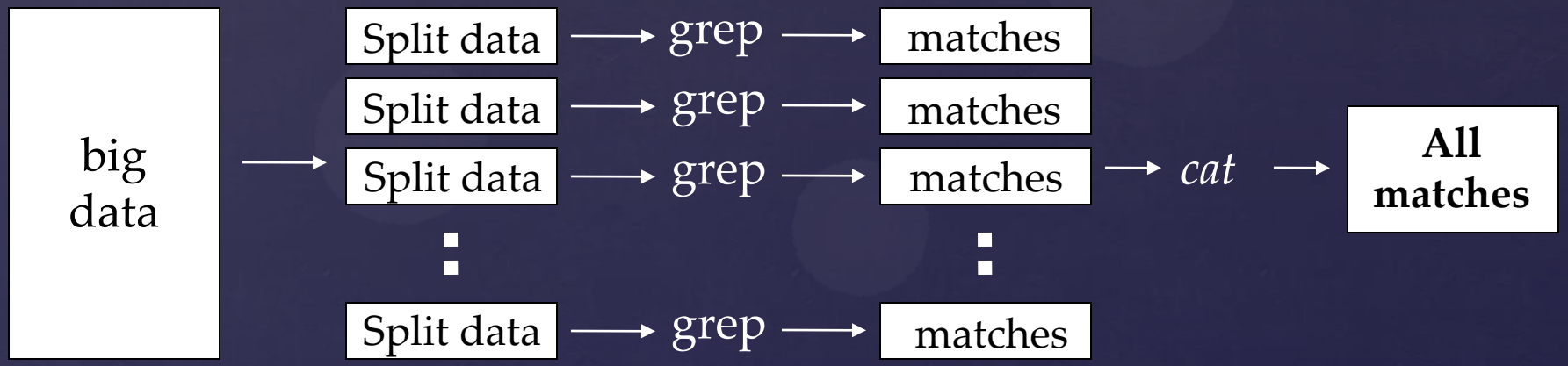
Indice

- Introduzione a MapReduce
- Modello di sviluppo
- Fault Tolerance
- Hadoop
- RHadoop
- Esercizio

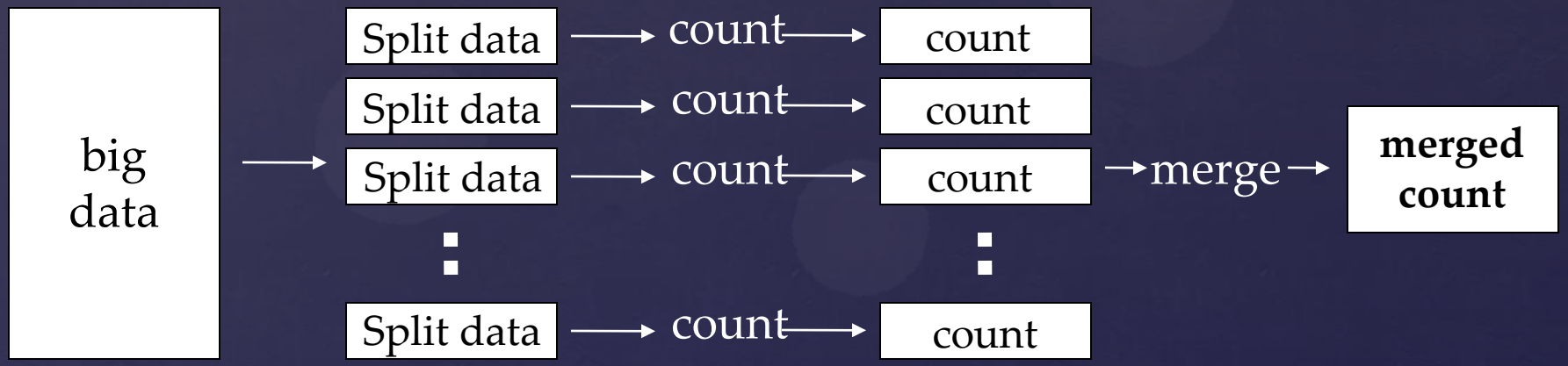
Introduzione a MapReduce

- Introdotto da Google e presentato al *6th Symposium on Operating Systems Design & Implementation – San Francisco, CA – 2004*
- Un semplice modello di programmazione
- Modello funzionale
- Creato per il processamento di grandi moli di dati
  - Sfrutta le risorse di grandi insiemi di calcolatori
  - Esegue processi in maniera distribuita
- Motivazioni
  - Cresce la necessità di processare grandi moli di dati
  - Scalabilità

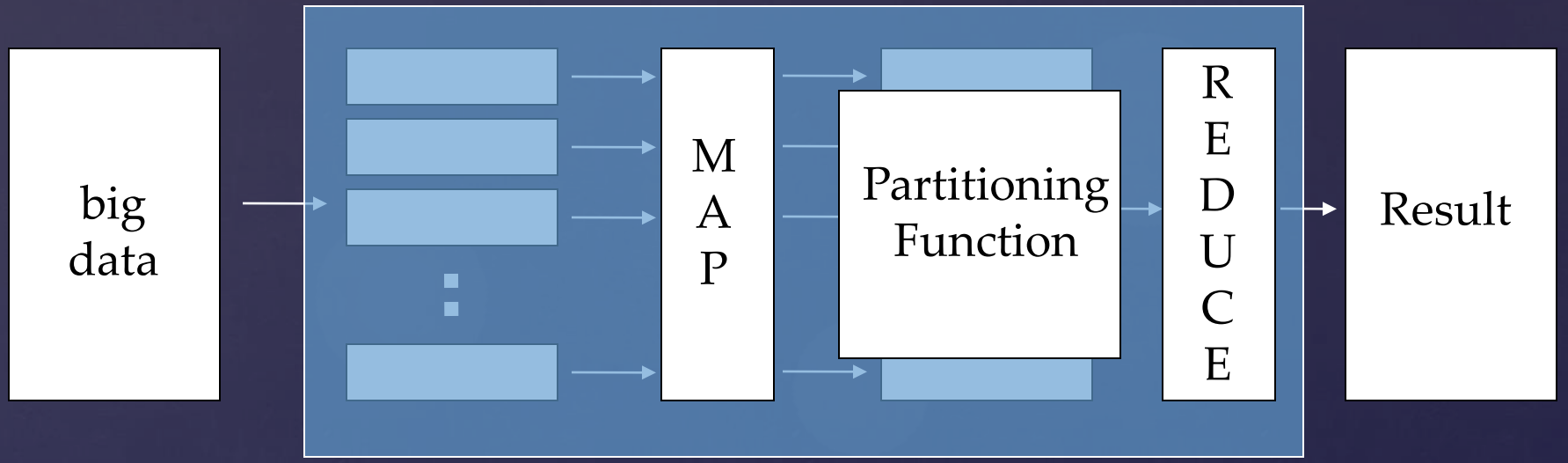
Grep distribuito



Word Count distribuito

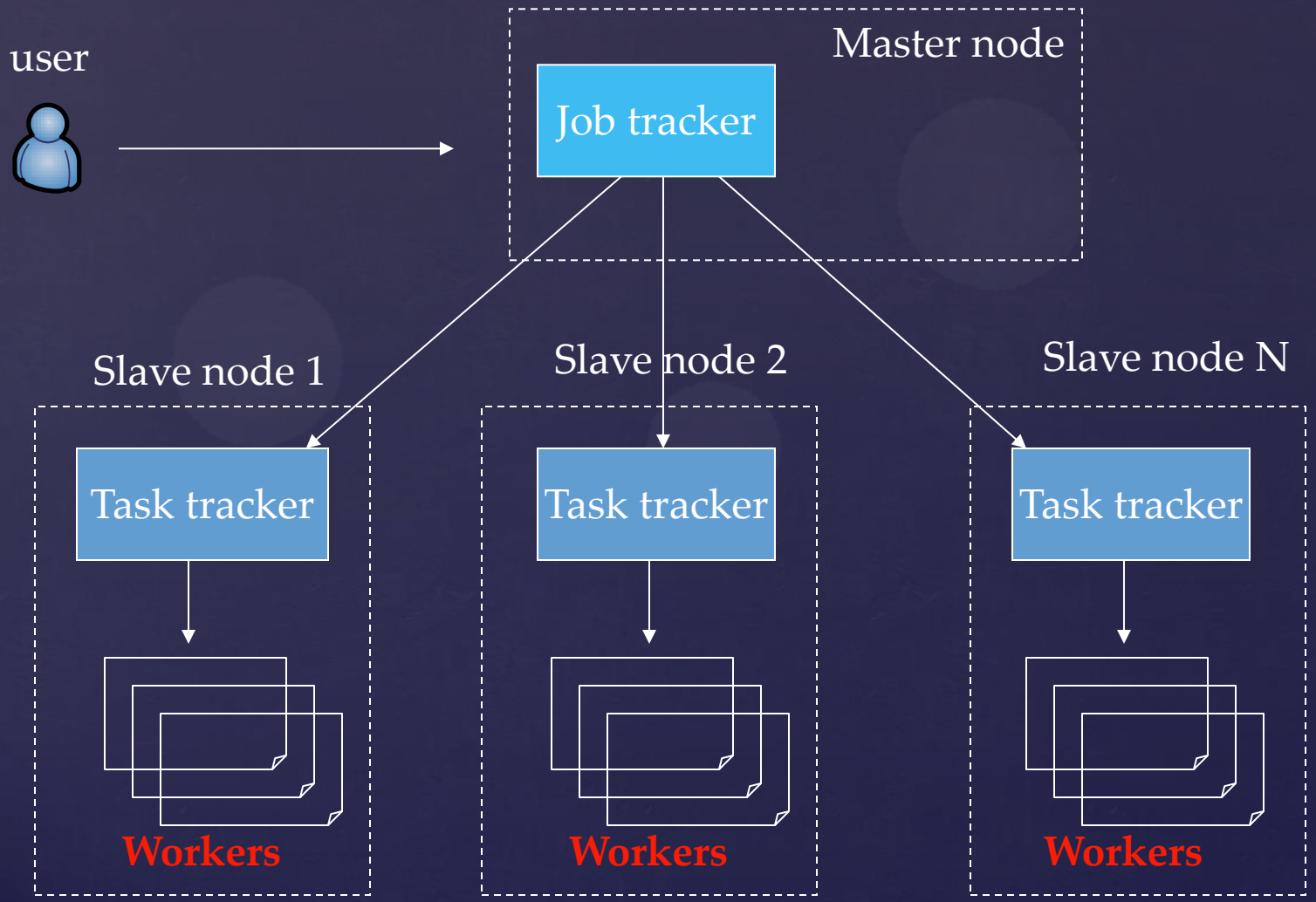


Map + Reduce



- Map:
  - Accetta in input una coppia chiave/valore
  - Restituisce in output un risultato intermedio come coppia chiave/valore
- Reduce :
  - Accetta in input il risultato intermedio come coppia chiave/valore
  - Restituisce in output una coppia chiave/valore

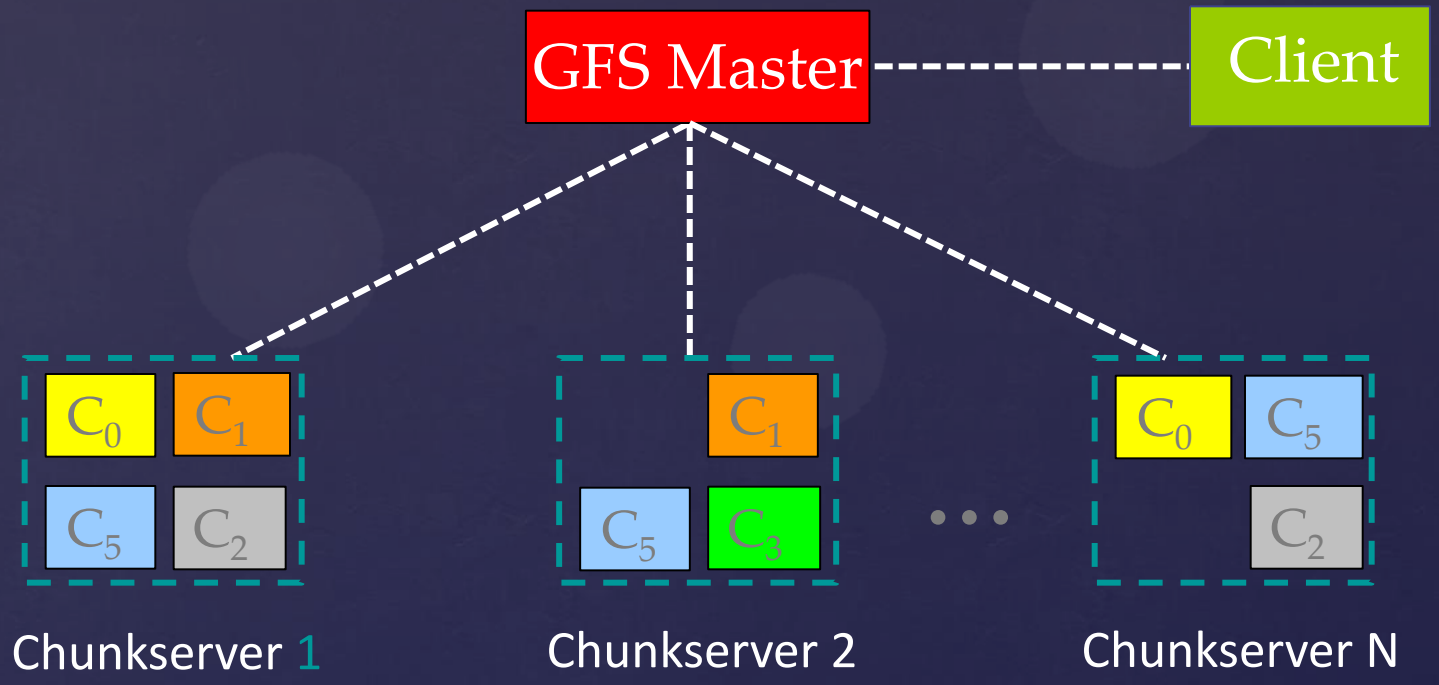
Modello di sviluppo



GFS: Global File System

- Obiettivo
  - *Ottenere una vista globale*
  - *Rendere facilmente accessibili grandi moli di dati*
- Master Node (meta server)
  - Nodo centralizzato, indicizza tutti i *chunks* sui data server
- Chunk Server (data server)
  - I file sono divisi in porzioni (chunk) contigue, tipicamente tra i 16 e i 64 MB.
  - Ogni chunk viene replicato (x2 o x3)
    - Mantenere le repliche in diverse locazioni

GFS: Architettura



Funzioni

- Map
  - Processare una coppia *chiave/valore* per generare un risultato intermedio come coppie *chieve/valore*
- Reduce
  - Unisce tutti i valori intermedi con la stessa chiave
- Partition
  - Partiziona il risultato intermedio in output dalla funzione Map secondo determinate condizioni definite a priori dall'utente

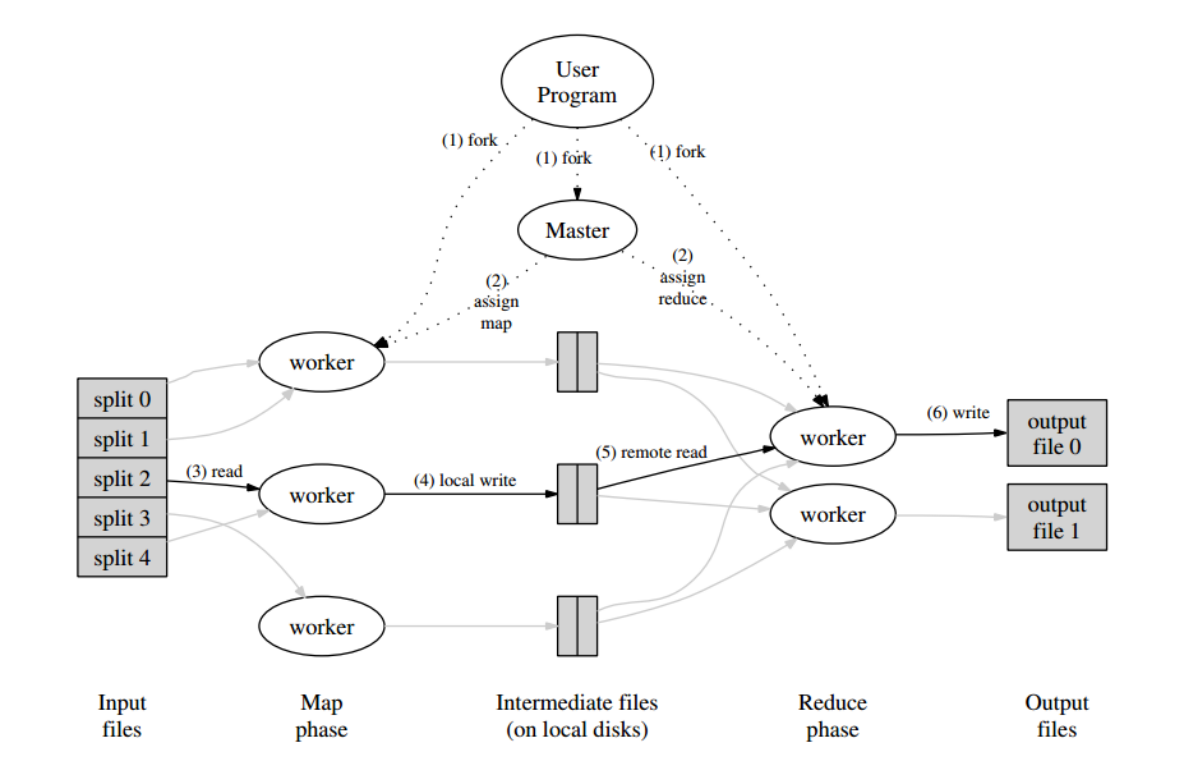
Esempio: Word Count

Conteggio delle parole in un grande insieme di documenti

```
map(string key, string value)
    //key: nome del documento
    //value: contenuto del documento
    for each word w in value
        EmitIntermediate(w, "1");

reduce(string key, iterator values)
    //key: parola
    //values: lista di conteggi
    int results = 0;
    for each v in values
        result += ParseInt(v);
    Emit(AsString(result));
```

Esempio: Word Count



Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

Fault Tolerance

- Comportamento reattivo
  - Worker Failure
    - *Il master verifica periodicamente che i worker siano attivi (ping)*
      - *Nessuna risposta = worker failure*
    - *Al verificarsi di un fallimento, il task in esecuzione sul worker viene riassegnato ad un altro worker*
  - Master Failure
    - *Il master crea periodicamente alcuni checkpoint*
    - *Un altro master può essere fatto partire dall'ultimo stato di checkpoint*
    - *Se il master fallisce, l'intero job viene interrotto*

Fault Tolerance

- Comportamento proattivo (esecuzione ridondante)
  - Problema degli “stragglers” (worker lenti)
    - *Ci sono altri job che consumano risorse sulla macchina*
    - *Se i dischi contengono errori causano un rallentamento nel trasferimento di dati*
  - Quando un processo è quasi terminato, rischedula i task in esecuzione
  - Ogni volta che l’esecuzione primaria o quella di backup termina, viene segnata come completata

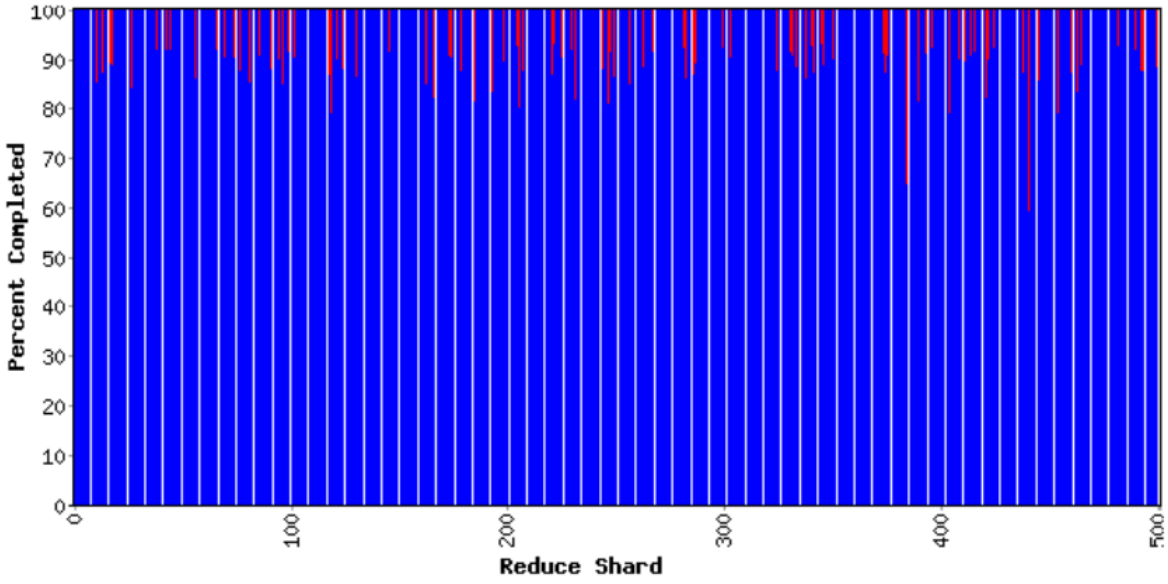
Status Monitor

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



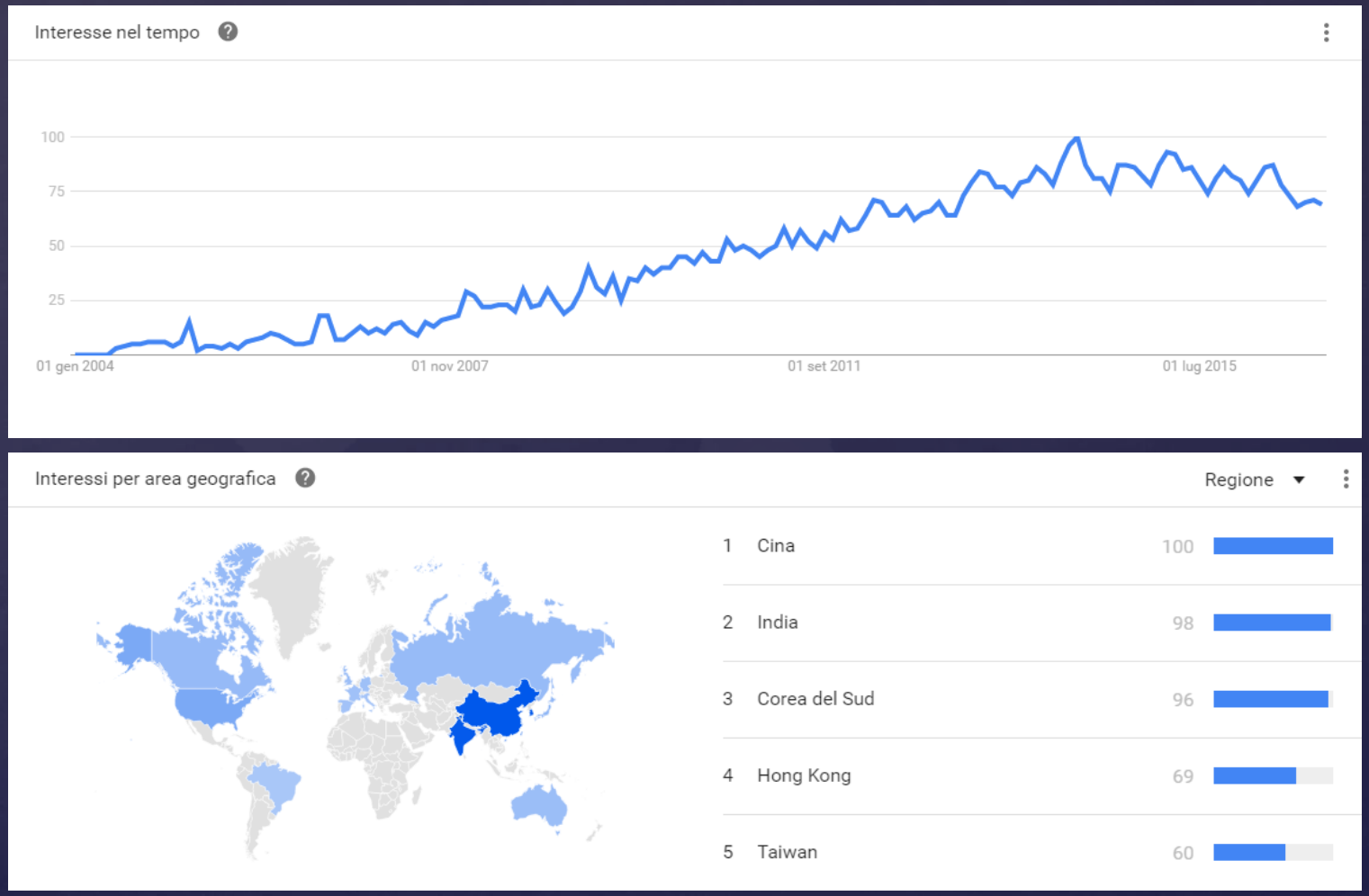
Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

Punti da enfatizzare

- Nessuna operazione di *reduce* può iniziare finchè le operazioni di *map* non vengono completate
- Se un nodo "*map worker*" fallisce prima che le operazioni di *reduce* si concludano, l'intero task viene riavviato
- La libreria MapReduce si occupa di fare gran parte del nostro lavoro!

Applicabilità del modello



Google Trends – 11 ottobre 2016

Hadoop

http://hadoop.apache.org/

- Open source
- Implementazione Java di MapReduce
- Utilizza HDFS come file system sottostante

Google	Yahoo
• MapReduce	• Hadoop
• GFS	• HDFS

Hadoop: installazione

LINUX

Pre-configurazione del sistema

- Oracle Java JDK (versione 7)

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-jdk7-installer
```

- Aggiunta al sistema un utente dedicato ad Hadoop

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

- Configurazione SSH

```
su - hduser
ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
<enter>
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Hadoop: installazione

LINUX

Pre-configurazione del sistema

```
ssh localhost

ssh: connect to host localhost port 22: Connection refused
```

```
sudo apt-get install openssh-server

ssh localhost

The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44:f9:36:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
```

Hadoop: installazione LINUX

http://hadoop.apache.org/

```
cd /usr/local
sudo tar xzf hadoop-2.4.0.tar.gz
sudo mv hadoop-2.4.0 hadoop
sudo chown -R hduser:hadoop hadoop
```

Aggiungere le seguenti righe al file `$HOME/.bashrc` per l'utente `hduser`

```
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export PATH=$PATH:$HADOOP_HOME/bin
```

Sostituire la seguente riga nel file `/usr/local/hadoop/etc/hadoop/hadoop-env.sh`:

```
export JAVA_HOME=${JAVA_HOME}
```

con

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
```

Hadoop: installazione

LINUX

Configurazione

```
sudo mkdir -p /app/hadoop/tmp
sudo chown hduser:hadoop /app/hadoop/tmp
sudo chmod 750 /app/hadoop/tmp
```

Modificare il file
`/usr/local/hadoop/etc/hadoop/core-site.xml`

```
<property>
<name>hadoop.tmp.dir </name>
<value>/app/hadoop/tmp </value>
<description>Base per altre directory temporanee</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description></description>
</property>
```

Hadoop: installazione

LINUX

Configurazione

Modificare il file
`/usr/local/hadoop/etc/hadoop/mapred-site.xml`

```
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
<description>
Host e porta su cui viene lanciato il MapReduce job tracker.
Se "locale", allora i jobs sono eseguiti in un unico processo
come un singolo map e reduce task.
</description>
</property>
```

Hadoop: installazione

LINUX

Configurazione

Modificare il file
`/usr/local/hadoop/etc/hadoop/hdfs-site.xml`

```
<property>
<name>dfs.replication</name>
<value>1</value>
<description>
Default block replication.
Il numero di repliche può essere specificato quando viene
creato il file. Il default viene usato se la replica non è
specificata al momento della creazione.
</description>
</property>
```

Formattare il file system
`/usr/local/hadoop/bin/hadoop namenode -format`

Hadoop: installazione

LINUX

Esecuzione

Avviare Hadoop:

```
/usr/local/hadoop/sbin/start-all.sh
```

Interfaccia WEB: Hadoop Administration

```
http://localhost:50070
```

Terminare Hadoop:

```
/usr/local/hadoop/sbin/stop-all.sh
```

RHadoop: installazione

Installare le seguenti dipendenze:

```
install.packages(c("rJava", "Rcpp",
      "RJSONIO", "bitops", "digest",
      "functional", "stringr", "plyr",
      "reshape2", "dplyr", "R.methodsS3",
      "caTools", "Hmisc"))
```

Scaricare i pacchetti `rhdfs, rhabse, rmr2, plyrmr` dal seguente link:

`https://github.com/RevolutionAnalytics/RHadoop/wiki`

Installare i pacchetti `rhdfs, rhbase, rmr2, plyrmr`:

```
install.packages("<path>/rhdfs_1.0.8.tar.gz", repos=NULL, type="source")
install.packages("<path>/rmr2_2.2.2.tar.gz", repos=NULL, type="source")
install.packages("<path>/plyrmr_0.2.0.tar.gz", repos=NULL, type="source")
install.packages("<path>/rhbase_1.2.0.tar.gz", repos=NULL, type="source")
```

Esercizio: Word Count

Problema: contare le parole in un documento

Scaricare il dataset di esempio dal seguente link:
https://gist.githubusercontent.com/StevenClontz/4445774/raw/1722a289b665d940495645a5eaaad4da8e3ad4c7/mobydick.txt

Spostare il dataset su HDFS

```
bin/hadoop fs -copyFromLocal mobydick.txt wordcount/data/
```

Esercizio: Word Count

```
library(rmr2)

## map function
map <- function(k,lines) {
      words.list <- strsplit(lines, '\\s')
      words <- unlist(words.list)
      keyval(words, 1)
}

## reduce function
reduce <- function(word, counts) {
      keyval(word, sum(counts))
}

wordcount <- function (input, output=NULL) {
      mapreduce(input=input, output=output,
      input.format="text", map=map, reduce=reduce)
}
```

Esercizio: Word Count

```
## svuota la cartella di output
system("bin/hadoop fs -rmr wordcount/out")

## sottomette il job
hdfs.root <- 'wordcount'
hdfs.data <- file.path(hdfs.root, 'data')
hdfs.out <- file.path(hdfs.root, 'out')
out <- wordcount(hdfs.data, hdfs.out)

## recupera i dati dal file system distribuito (HDFS)
results <- from.dfs(out)

## stampa le prime 30 parole più frequenti
results.df <- as.data.frame(results, stringsAsFactors=F)
colnames(results.df) <- c('word', 'count')
head(results.df[order(results.df$count, decreasing=T), ], 30)
```

Referenze

Articoli:

- https://research.google.com/archive/**mapreduce**-osdi04.pdf
- http://pages.cs.wisc.edu/~akella/CS838/F15/838-CloudPapers/hdfs.pdf

Tutorial:

- http://wiki.apache.org/hadoop/Hadoop2OnWindows
- https://wiki.apache.org/hadoop/
  Running_Hadoop_On_OS_X_10.5_64-bit_(Single-Node_Cluster)

Esercizi:

- https://github.com/michiard/CLOUDS-LAB/blob/master/labs/
  mapreduce-lab/README.md