Map-Reduce e Spark

Map Reduce - Genesi

- nasce nel 2003-2004
  - [“MapReduce: simplified Data Processing on Large Clusters”, Dean, Ghemawat. Google Inc.]
  - Closed-source, scritto in c++
  - nasce dall’esigenza di eseguire problemi “semplici” su big data (>1TB) - vedremo degli esempi…
- 2006: Apache & Yahoo! → rilasciano Hadoop e HDFS
  - Open source, scritto in Java
- 2008: diventa un progetto indipendente di Apache
- Oggi: Usato come piattaforma general purpose di storage e analisi di big data

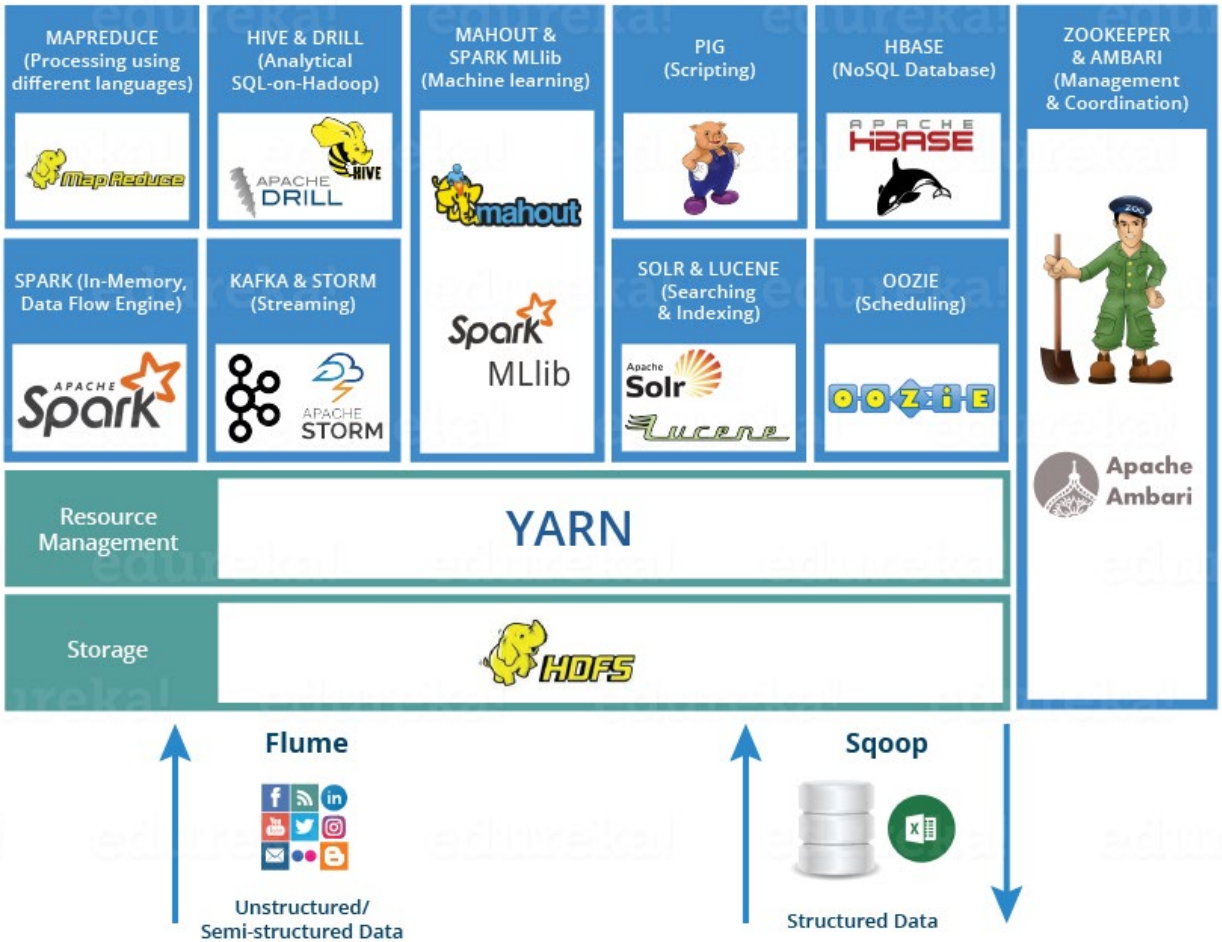
Nella pratica

Permettono a programmatori senza alcuna esperienza di sistemi distribuiti di utilizzare facilmente le risorse di un data center per elaborare grandi moli di dati.

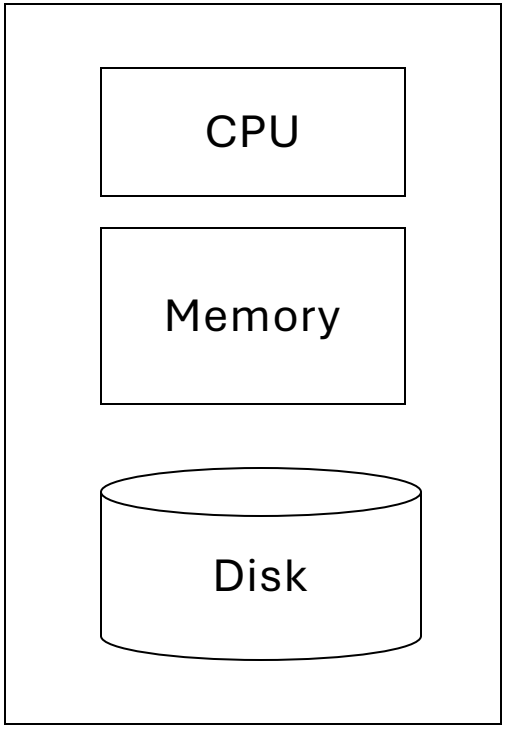
Apache Hadoop

- open source
- struttura e interfaccia relativamente semplici
- fatto di due sotto-componenti:
  - MapReduce Runtime (coordinatore)
    - Paradigma di programmazione per realizzare la programmazione distribuita su diversi server
  - Hadoop Distributed File System (HDFS)
    - File system distribuito
    - Server possono fallire ma non causano l'abort della computazione
    - Dati replicati con ridondanza nel cluster

Hadoop Ecosystem



Single Node Architecture



Machine Learning, Statistica

Data Mining "Classico"

Ma…

- I dati (es. web) possono essere molto grandi
  - Decine di centinaia di TB
- Non si può fare il mining su un server singolo (perché?)

- Esempio
  - 20+ miliardi di pagine web x 20KB = 400+ TB
  - 1 computer legge 30-35 MB/sec dal disco
    - ~4 mesi per leggere il web
  - ~1,000 hd per immagazzinare il web
  - Ma ci vuole molto più tempo per fare qualcosa di utile con questi dati!!!

- Large scale computing per il data mining:
  - Sfide:
    - Come distribuire il calcolo?
    - La programmazione parallela/distribuita è complicata.

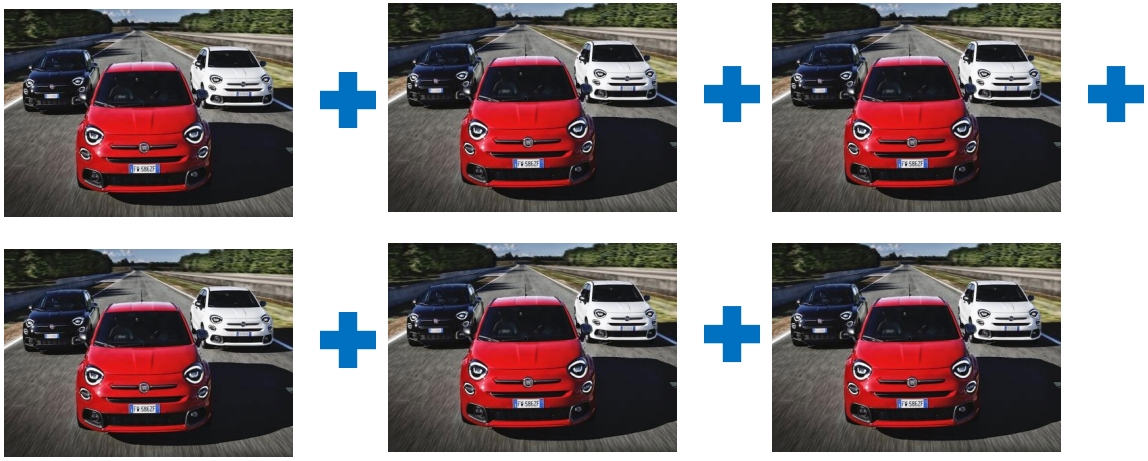
Motivazioni

- Negli anni sono emerse nuove architetture per questi problemi:
  - Cluster of commodity Linux node
  - Commodity network (ethernet)

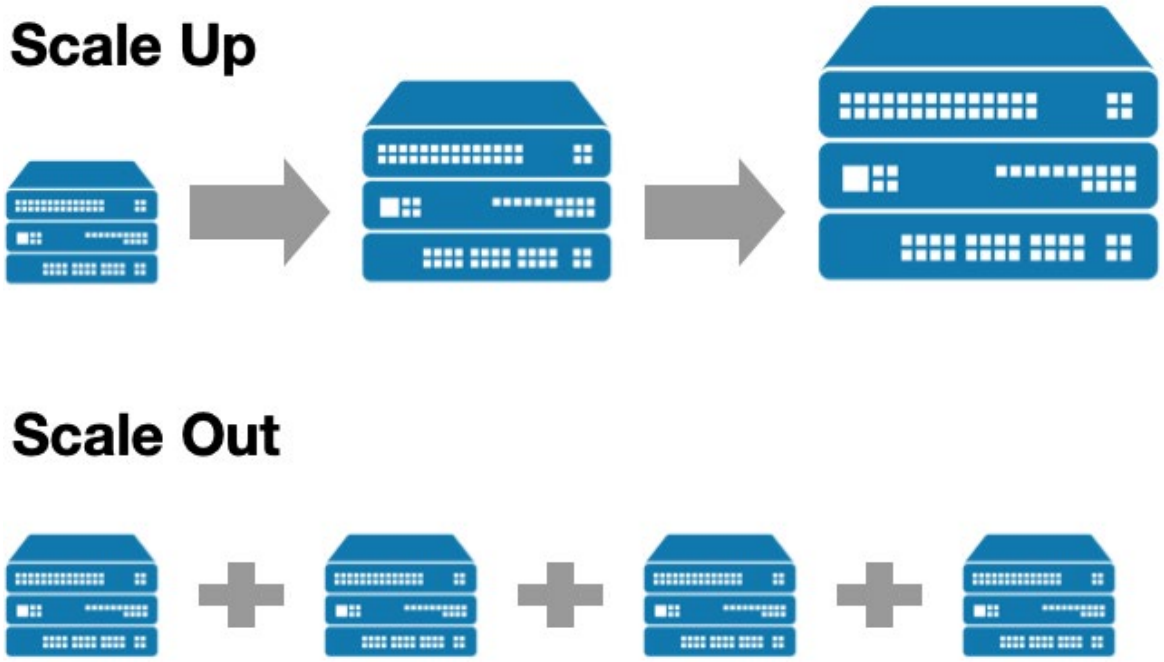
Commodity clusters

- Come organizzare il calcolo su questa architettura?
- Nascondere problemi legati al fallimento dell'hardware

Scale up vs Scale out



Scale up vs Scale out



Idea e Soluzioni

- Problema:
  - Copiare i dati sulla rete richiede tempo

- Idea:
  - Portare il calcolo vicino ai dati
  - Immagazzinare i file più volte per la lettura

Riassumendo..

1. Voglio processare grandi moli di dati
2. Voglio usare molte (centinaia/ migliaia) di CPU
3. Voglio che tutto questo sia SEMPLICE

Idea e Soluzioni

- Map-reduce risolve questi problemi
  - Modo elegante per lavorare con i big data
- Infrastruttura di Storage – File system
  - Google: GFS. Hadoop: HDFS
- Modello di programmazione
  - Map-Reduce

Storage Infrastructure

- Problema:
  - Se un nodo fallisce come mantengo i dati?
- Soluzione:
  - File system distribuito:
    - Global file namespace
    - Google GFS; Hadoop HDFS;
- Tipico pattern di utilizzo
  - File molto grandi (100s of GB to TB)
  - Raramente aggiornati
  - Letture e inserimenti in append

File System distribuito

- File divisi in chunk
- Tipicamente ogni chunk è 16-64MB
- Ogni chunk replicato (usualmente 2x o 3x)
- Repliche su rack diversi
- Dimensione dei chunk e grado di replica sono stabiliti dall'utente
- Un file speciale (nodo master) memorizza, per ogni file, la posizione dei chunk
- Il nodo master è replicato
- Una directory per il file system conosce dove trovare il master node

File System distribuito

- Master node
  - Name Node in Hadoop HDFS (senza name node l'HDFS non puo' essere usato)
  - Immagazzina i metadati relativi alle locazioni dove sono memorizzati i file
  - Può essere replicato
  - Conosce tutti i datanode che contengono i blocchi di un file
- Data Node
  - Immagazzinano e recuperano i blocchi quando richiesto.
  - I blocchi sono memorizzati in diversi datanode.

Quali sono le sfide

- Come assegnare le unità di lavoro ai worker?
- Si possono avere piu' unità di lavoro che worker?
- I worker possono condividere risultati parziali?
- Come aggreghiamo i risultati?
- Come facciamo a sapere che tutti i worker hanno finito?
- Se un worker fallisce?

Esempio di tipico large-data problem

MAP

- Iterare su un grande numero di record in parallelo
- Ogni iterazione estrae qualcosa di interessante
- Fare uno shuffling e sort dei risultati intermedi dalle iterazioni concorrenti
- Aggregare i risultati intermedi

Reduce

- Generare l'output finale

Trova le sue radici nella programmazione funzionale

- MAP: prende una funzione f e la applica ad ogni elemento di una lista
- FOLD: iterativamente applica una funzione g per aggregare i risultati

MapReduce

- MAP: prende in input un oggetto con una chiave ed un valore (k,v) e resituisce un elenco di coppie chiave valore (k_1,v_1), (k_2,v_2),..., (k_n,v_n)
- Il framework colleziona tutte le coppie con la stessa chiave k e associa a k tutti i valori $(k,[v_1,v_2,....v_l])$
- REDUCE: prende in input una chiave e una lista di valori $(k,[v_1,v_2,....v_l])$ e li combina in una qualche maniera

Programming Model: MapReduce

Warm-up task:

- Grande documento di testo
- Contare il numero di volte in cui ogni parola distinta appare nel documento

- Applicazione d'esempio:
  - Analizzare i log del web server per trovare gli URL popolari

Task: Word Count

Caso 1:

- File troppo grande per andare in memoria, ma le coppie <word, count> entrano in memoria

Caso 2:

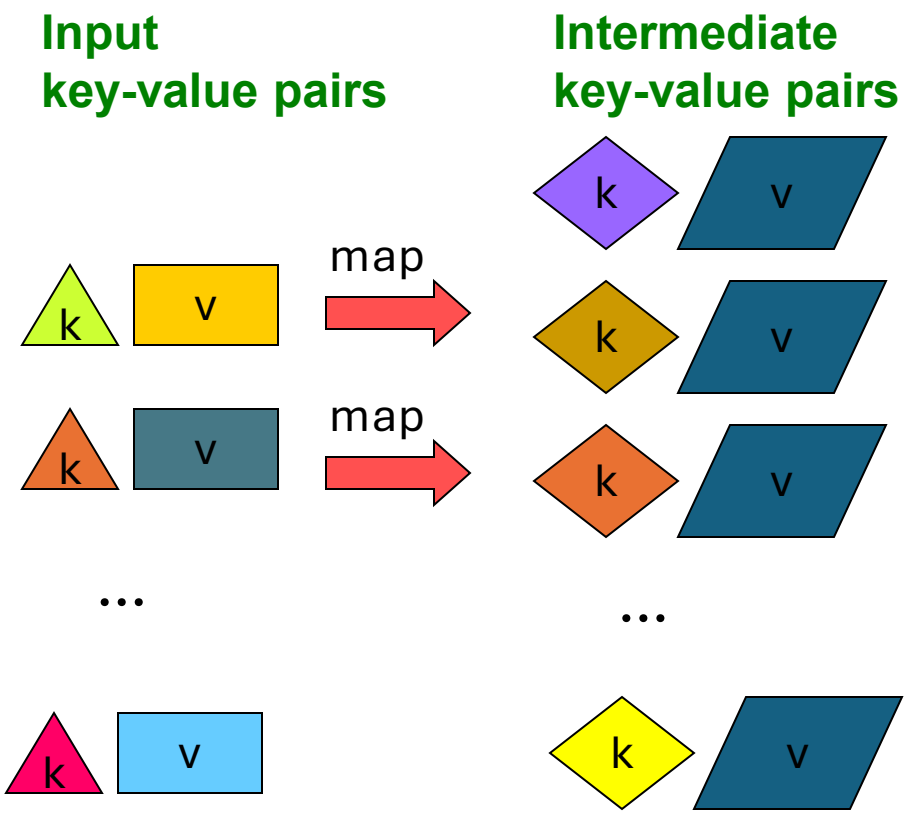
- Calcolare le occorrenze di parole:
  - **`words(docs/*) | sort | uniq -c`**
    - dove **`words`** prende in input il file e da in output le parole presenti nel file, una per riga.
- Caso 2 cattura l'essenza di MapReduce
  - Altamente parallelizzabile

MapReduce: Overview

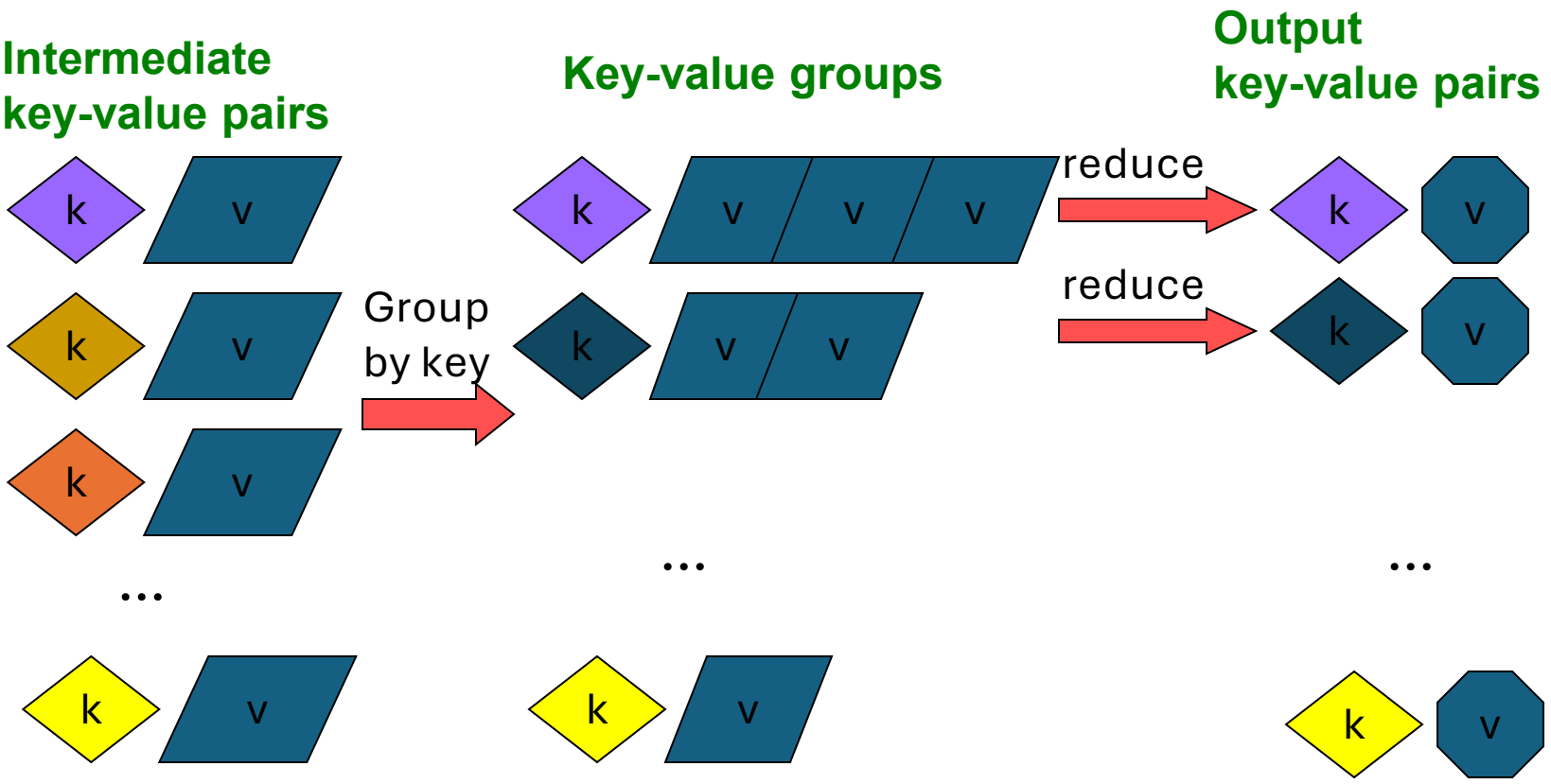
- Legge sequenzialmente molti dati
- Map:
  - Estrarre qualcosa di utile per il calcolo
- Group by key: Sort e Shuffle
- Reduce:
  - Aggrega, somma, filtra e trasforma
- Scrivi I risultati

Pipeline sempre la stessa, Map e Reduce cambiano da problema a problema

MapReduce: The Map Step



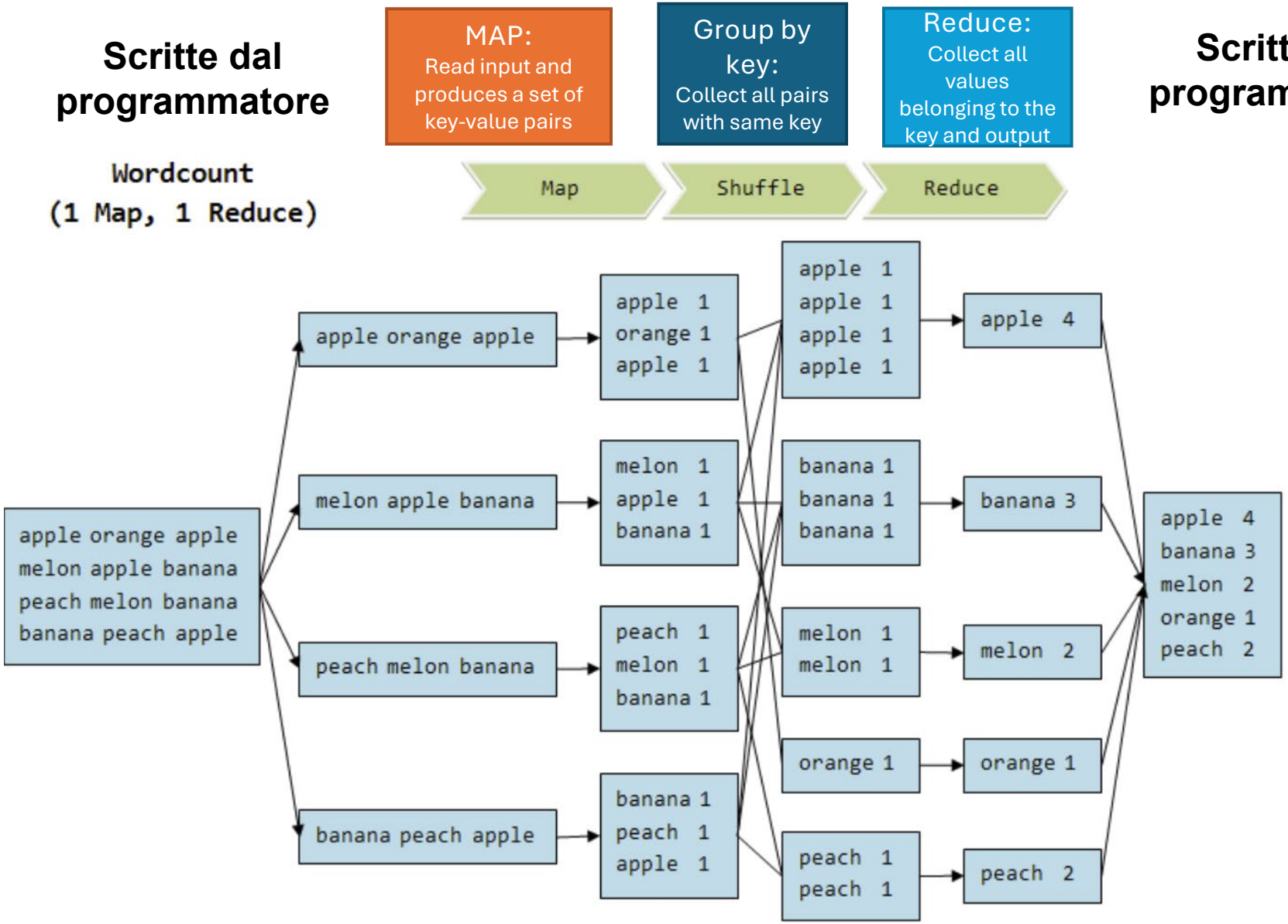
MapReduce: The Reduce Step



Più in dettaglio

- Input: set di coppie key-value (chiave-valore)
- Il programmatore scrive due metodi:
  - **Map(k, v)** → list(<k', v'>)
    - Prende coppie key-value e da in output un insieme di coppie key-value
      - Es. key il nome del file, value una singola linea del file
    - Una call di Map per ogni coppia (k,v)
  - **Reduce(k', list(<v'>))** → list(<k', v''>)
    - Tutti I valori v' con la tessa chiave k' sono ridotti assieme e processati nell'ordine di v'
    - Una call della funzione Reduce per ogni chiave univocal k'

MapReduce: Word Counting



Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
      emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce: Ambiente

L'ambiente Map-Reduce si occupa di:

- Partizionare i dati di input
- Scheduling l'esecuzione del programma sulle machine
- Effettuare lo step group by key
- Gestire i fallimenti delle machine
- Gestire la comunicazione tra le machine

Esempio

- Problema: contare quante parole di una certa lunghezza esistono in una collezione di documenti
- Input: repository di documenti

- Map:

- Reduce

Soluzione

```
map(key, value):
// key: document name; value: text of the document
 for each word w in value:
     emit(length(w), 1)
```

```
reduce(key, values):
// key: a word; value: an iterator over counts
       result = 0
       for each count w in values:
             result += 1
       emit(key, result)
```

Data Flow

- Input e output finale memorizzati nel distributed file system (FS):
  - Scheduler prova a schedulare i map task “vicini” allo storage fisico dei dati di input
- Risultati intermedi memorizzati su local FS dei worker Map e Reduce
- Solitamente l’output di un task è l’input di un altro

Coordinamento: Master

- Il Master node effettua il coordinamento:
  - Task status: (idle, in-progress, completati)
  - Idle task schedulati appena il worker diventa disponibile
  - Quando un map task completa, spedisce al master la posizione e la size dei suoi R file intermedi, uno per ogni reduce
  - Il Master fa il push di queste info ai reducer
- Il Master pinga i worker periodicamente per identificare fallimenti

Quanti Job Map e Reduce?

- M map task, R reduce task
- linea guida euristica:
  - M molto più grande del numero di nodi del cluster
  - Comunemente: un chunk DFS per map
  - Migliora il load-balancing dinamico e aumenta la velocità del recovery
- Usualmente R più piccolo di M e perché l'output è suddiviso su R file

Raffinamenti

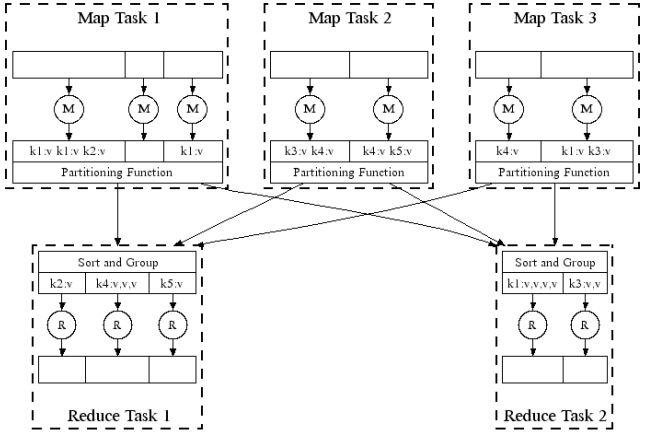
- Problema
  - Worker lenti aumentano in modo significativo il tempo di completamento del job:
    - Altri job nella macchina
    - Dischi cattivi
    - Ecc.
- Soluzione
  - Vicino alla fine della fase, genera copie di backup delle attività
    - Chi finisce per primo «vince»
- Effetto
  - Riduce notevolmente il tempo di completamento del job

Raffinamento: Combiners

- Quando la funzione Reduce è associativa e commutativa possiamo far fare al map parte del task del reducer.

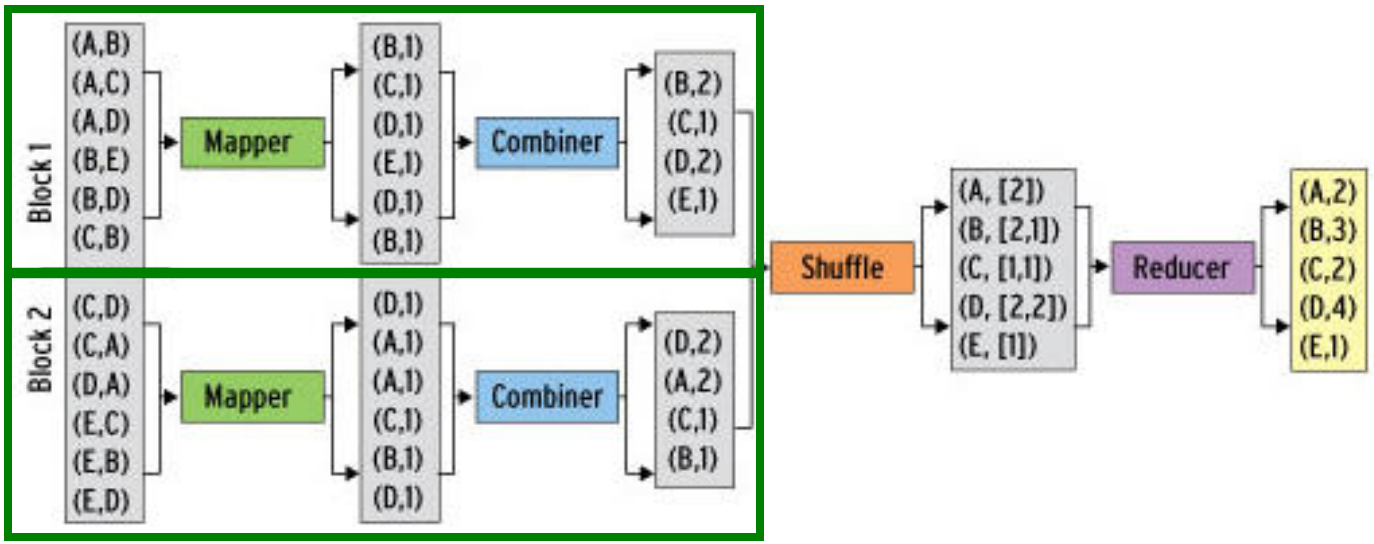
- Spesso un task Map produrrà diverse coppie del tipo (k,v_1), (k,v_2), ... per la stessa chiave k

- Ridurre network time pre-aggregando I valori del mapper:
  - combine(k, list(v_1)) → v_2
  - Combiner usulamente lo stesso della funzione reduce



Refinement: Combiners

- Torniamo all'esempio del word count:



Raffinamenti: Partition Function

- Controllare le come chiavi vengono partizionate?
  - Gli input per mappare le attività sono creati da split contigue del file di input
  - Ridurre le esigenze per garantire che i record con la stessa chiave intermedia finiscano nello stesso worker
- Il Sistema usa una funzione di partizione di default:
  - **hash(key) mod *R***
- A volte è utile fare l'override di tale hash function:
  - Es. **hash(hostname(URL)) mod *R*** garantisce che URL da un host finiscono nello stesso output file

MapReduce: il quadro completo

- Il programmatore specifica due funzioni

  map(k1,v1) → [(k2,v2)]

  reduce(k2,[v2]) → [(k3,v3)]

  Tutti i valori con la stessa chiave sono ridotti assieme

- Il programmatore può anche specificare:

  combine(k2,[v2]) → [(k3,v3)]

  - Mini reducer che vengono eseguite dopo la fase di map
  - Usate per ottimizzare e ridurre il traffico nella rete

  partition(k2,numero di partizioni) → partizioni per k2

  - Divide lo spazio delle chiavi per l'esecuzione parallela delle operazioni di reduce

- Al resto ci pensa il framework!!

Problemi di base per Map-Reduce

Esercizio 1: Host Size

- Supponiamo di avere un grande corpus web
  - Guardiamo i metadati
  - Le linee hanno il seguente formato (URL, dimensioni, data, ...)
- Per ogni host, trova il numero totale
- di byte
  - cioè la somma delle dimensioni della pagina per tutti URL da quell'host

Esercizio 1: Soluzione

```
map(key, value):
// key: URL; value:   {size,date,..}
    emit(hostname(URL), size)

reduce(key, values):
// key: a hostname; values: an iterator over
sizes
    result = 0
    for each size s in values:
        result += s
    emit(key,result)
```

Esercizio 2: Graph reversal

- Dato un grafo diretto descritto con lista di adiacenza:
  - src1: dest11, dest12, …
  - src2: dest21, dest22, …
- Costruisci il grafo in cui tutti i link sono invertiti

```
map(key value):
key: filename; value:     list of adjacency
of each node
    for each r in value
        for each v in adj(r)
            emit(v,r)
```

Reduce:

- Identity function
  - <target, list(source)>

Esercizio 3: Inverted Index

- Supponiamo di avere un grande corpus web, ogni documento identificato da un ID
  - Per ogni parola che appare nel corpus, restituire l'elenco di docID dove la parola si trova

  Es. parola: doc_id1, doc_id2 ..

Esercizio 4: Distributed grep

- Voglio filtrare le linee di un insieme di documenti in cui appare una parola X.

Esercizio 5: Distributed sort

- Dato un elenco di record di un documento spezzato in chunk, ordinarli secondo un certo criterio.

Problemi idonei per Map-Reduce

Example: Language Model

- Statistical machine translation:
  - Necessità di contare il numero di volte in cui ogni sequenza di 5 parole si presenta in un corpus di documenti

- Molto semplice MapReduce:
  - Map:
    - extrai (5-word sequence, count) dal documento
  - Reduce:
    - Combina i count

Example: Join By Map-Reduce

- Calcolare la natural join R(A,B) ⋈ S(B,C)
- R ed S immagazzinate in un file
- Tuples sono coppie (a,b) or (b,c)

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

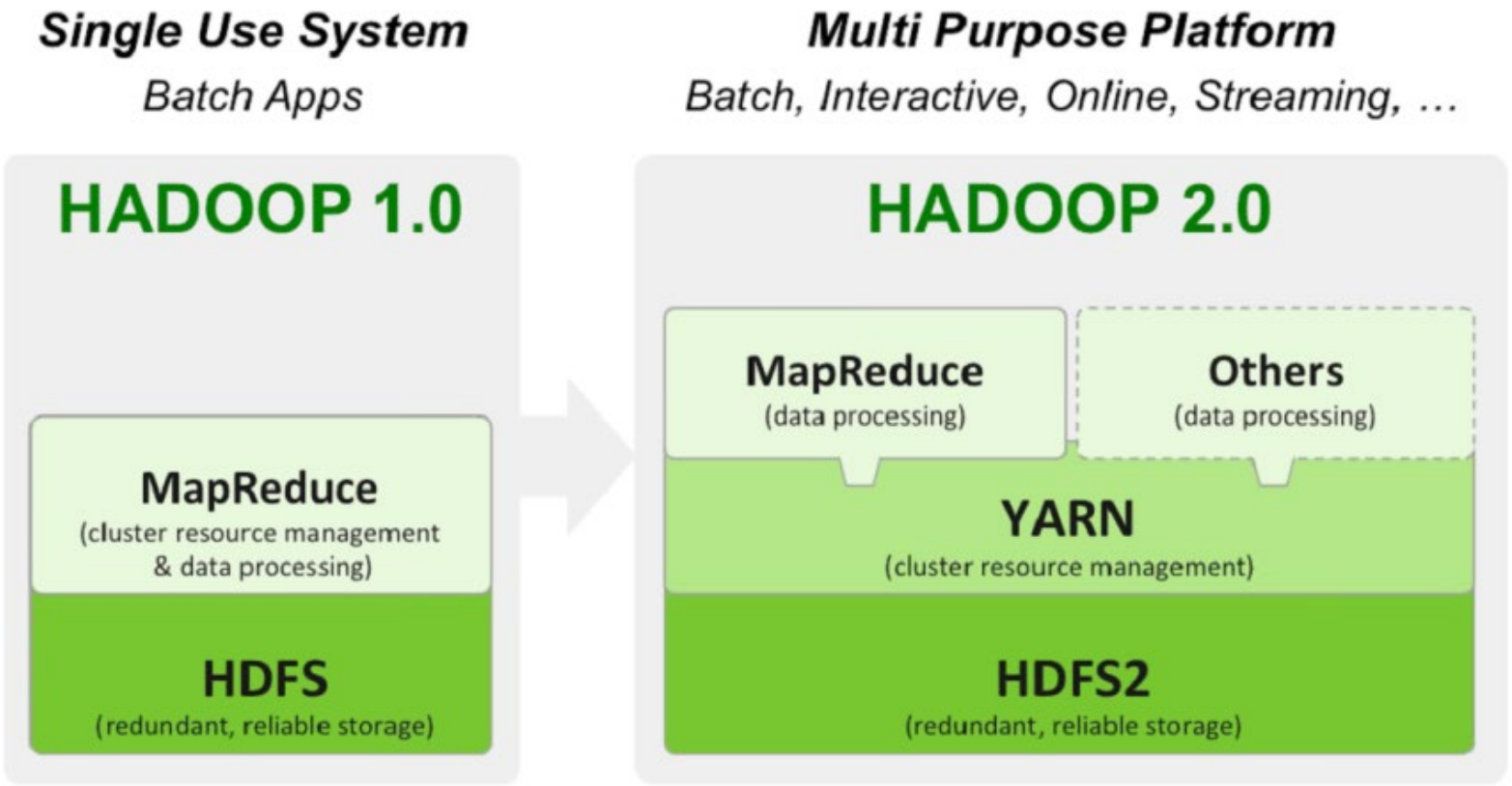
Map-Reduce Join

- A Map process :
  - Ogni tupla R(a,b) emette la coppia (b,(a,R))
  - Ogni tuple S(b,c) emette la coppia (b,(c,S))

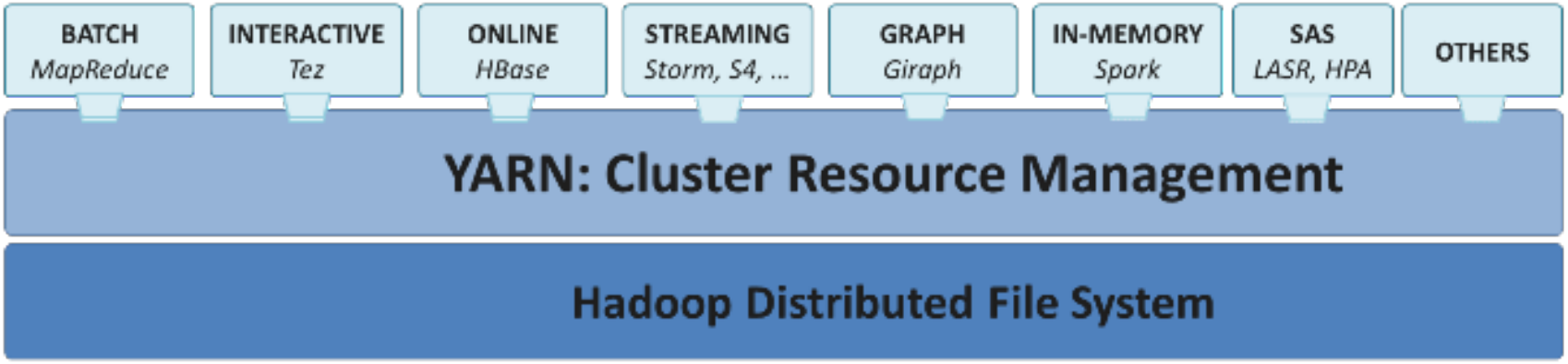
- Ogni Reduce fa il match di tutte le pairs (b,(a,R)) con (b,(c,S)) e restituisce in output (a,b,c).

Hadoop!

Hadoop 1.0 vs Hadoop 2.0



YARN: Yet Another Resource Negotiator



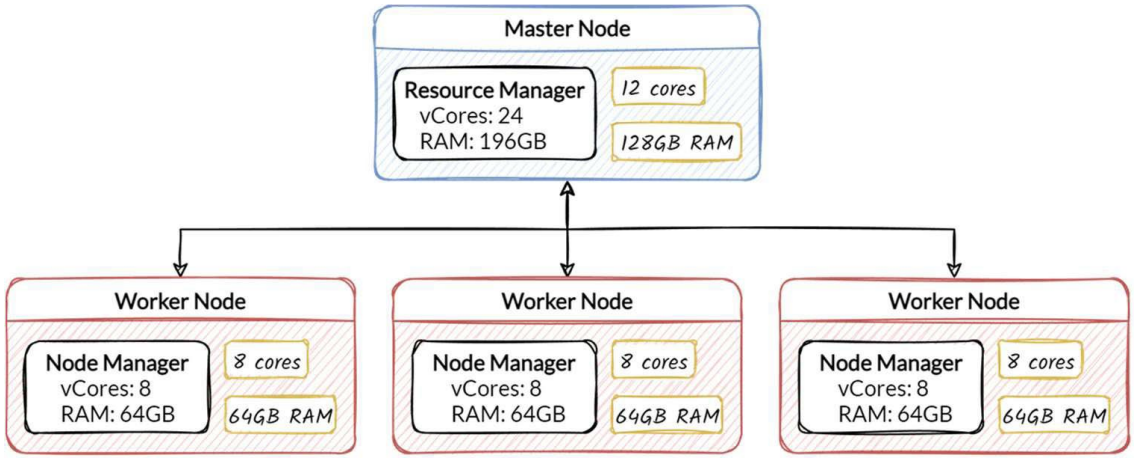
Architettura

YARN e' composto da quattro pezzi:

- Resource Manager
- Node Manager
- Application Master
- Container

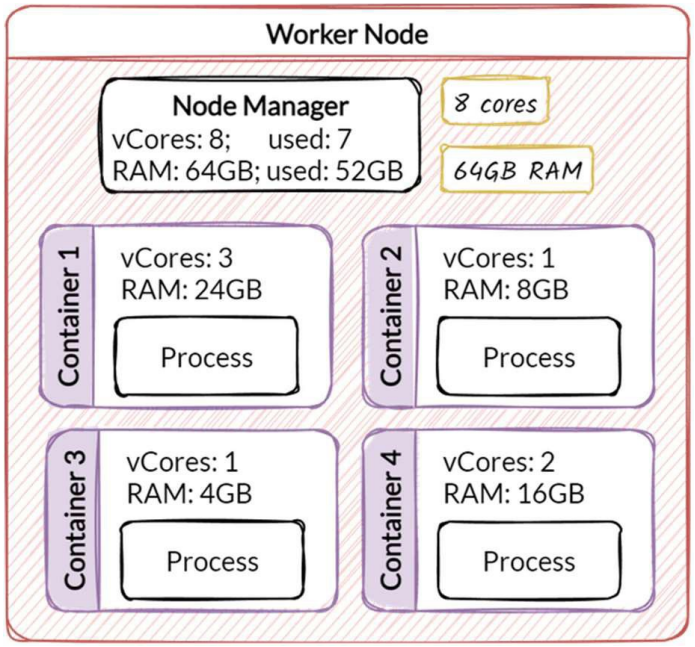
Hadoop: Concetti di base

- **Gestore delle risorse**
  - Controlla le risorse disponibili nel cluster (per tutte le applicazioni).
- **Gestore nodi**
  - Avvia e tiene traccia dei processi assegnati ai worker (un processo per nodo)

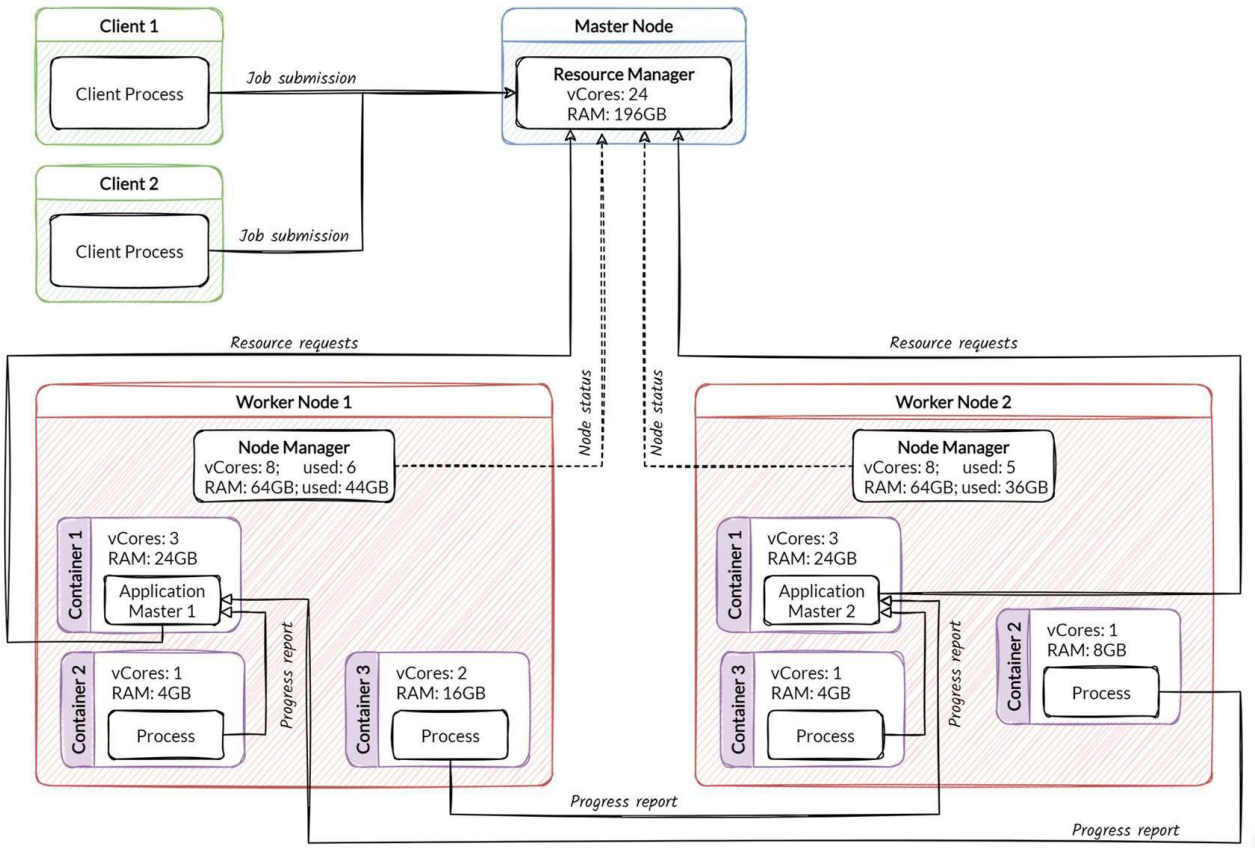


- **Container:** è un sottoinsieme di risorse del cluster (concetto chiave!)

- **Application master:** gestisce una particolare applicazione e viene eseguita in un contenitore. Responsabile della tolleranza ai guasti



Hadoop: Concetti di base– Resource Negotiator Process

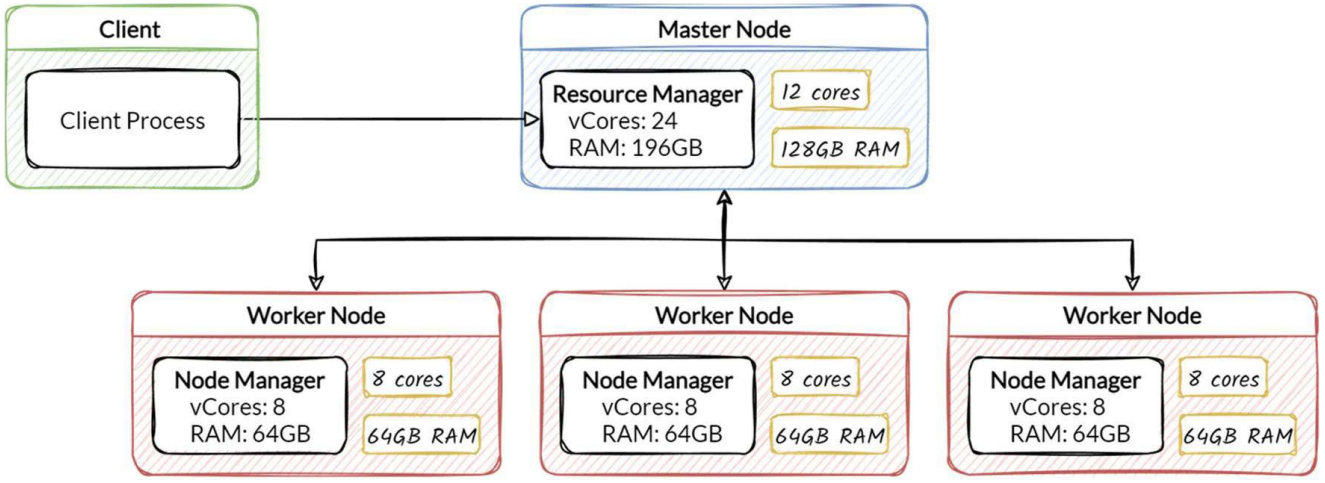


Hadoop: Concetti di base– Configurazioni

- Hadoop può essere eseguito con 3 diverse configurazioni:
  1. **Local / Standalone**. Viene eseguito in un unico JVM (Java Virtual Machine). *Molto utile per il debug!*
  2. **Pseudo-distributed** (Cluster simulator)
  3. **Distributed** (Cluster)

Hadoop V2 – Esecuzione di un processo MapReduce

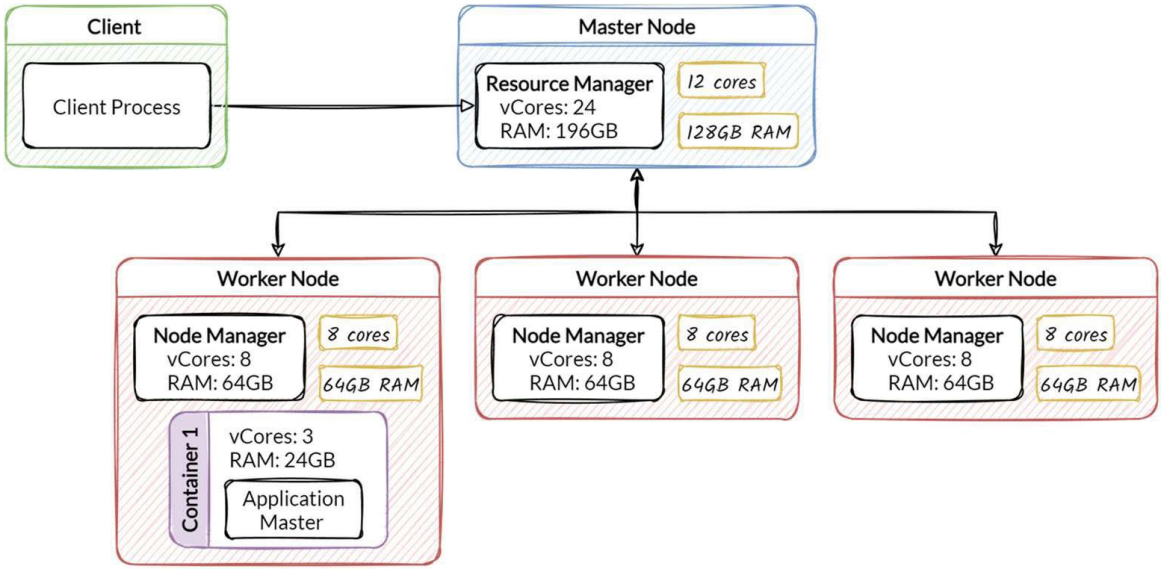
- **Esecuzione** di un processo MapReduce
  1) Il client avvia il processo (connessione con il `Resource Manager`)



Hadoop V2 – Esecuzione di un processo MapReduce

- Esecuzione di un processo MapReduce

  2) Il Resource Manager alloca un singolo contenitore in cui viene eseguito l'application master

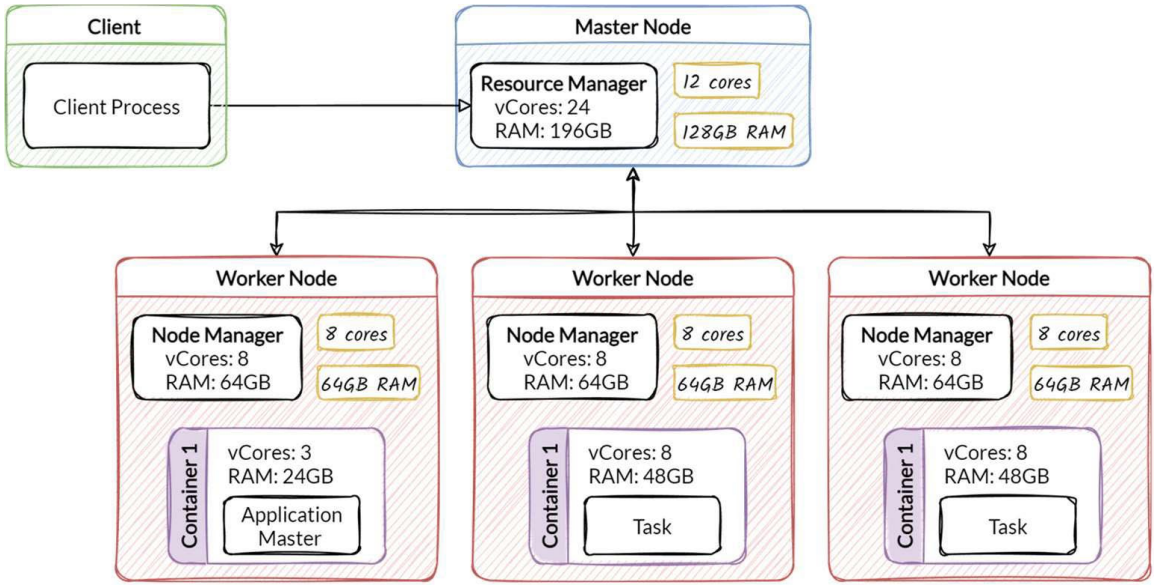


Hadoop V2 – Esecuzione di un processo MapReduce

- **Esecuzione** di un processo MapReduce

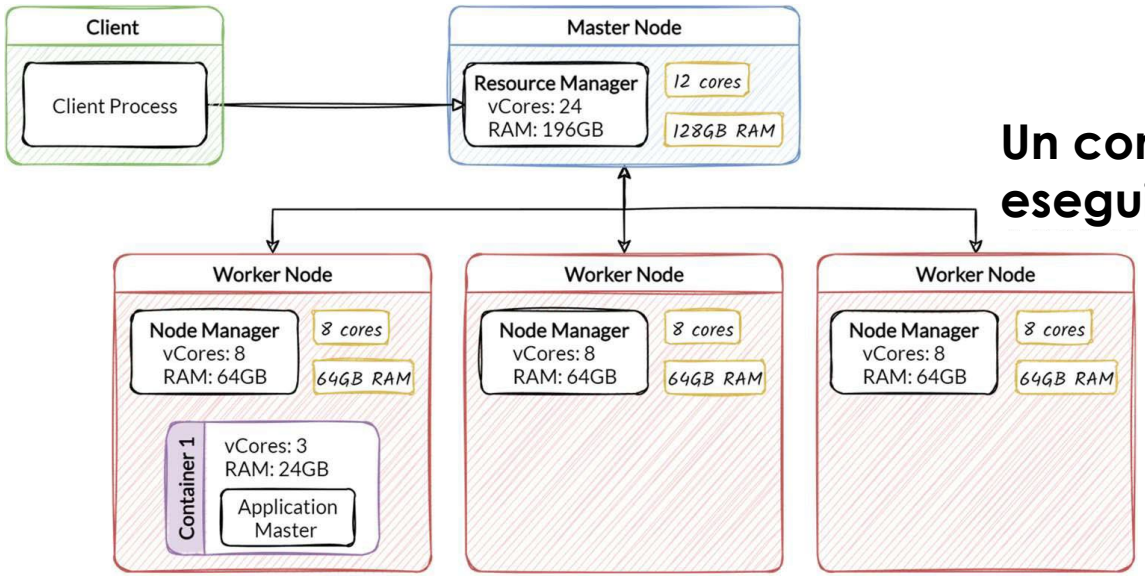
  3) L'Application Master richiede ai container di eseguire tutte le attività (in nodi diversi)

I contenitori potrebbero non usare tutta la memoria disponibile in un nodo



Hadoop V2 – Executing a MapReduce process

- **Execution** of a MapReduce process
  4) Tutte le attività vengono eseguite nel container. Container vengono rilasciati una volta terminati i suoi compiti

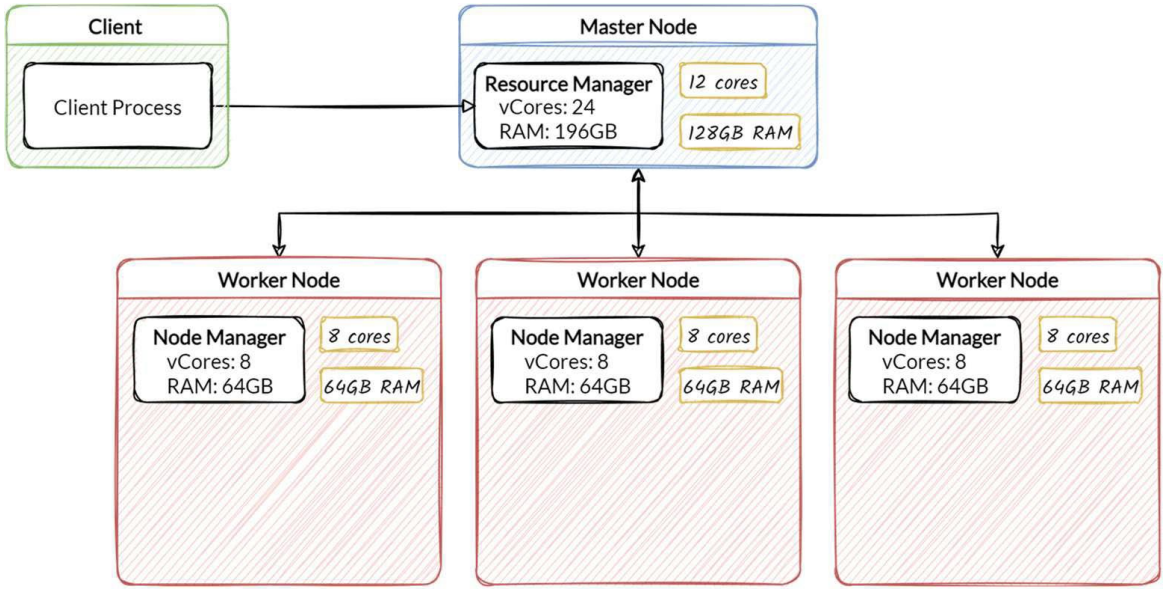


Un contenitore può eseguire più attività

Hadoop V2 – Executing a MapReduce process

- **Esecuzione** di un processo MapReduce

  5) L' **`Application Master`** nfinisce quando tutti i task sono stati completati e i container rilasciati



Come istalare Hadoop su Docker

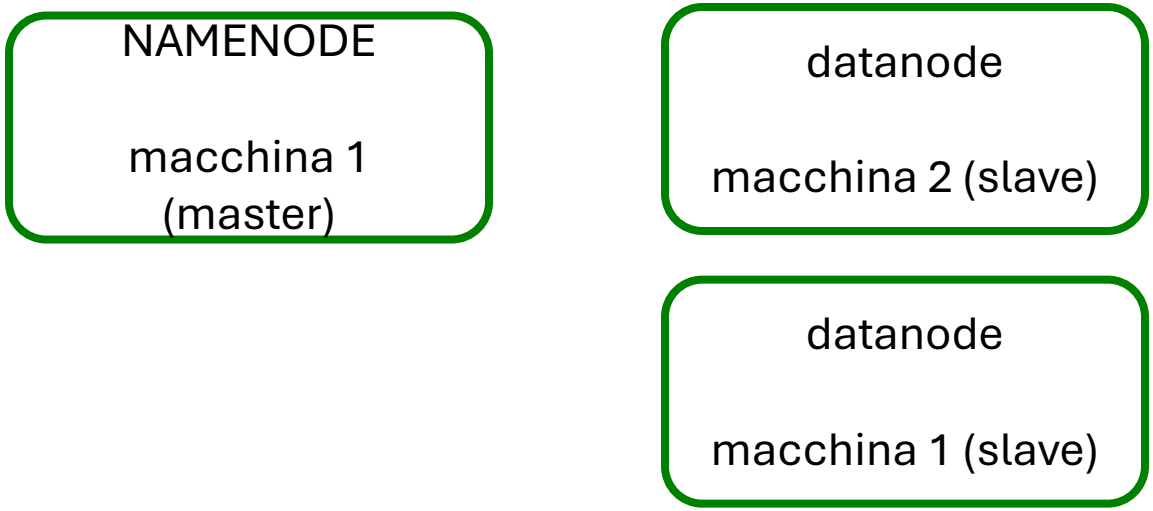
- Istallare Docker Desktop:
  https://www.docker.com/products/docker-desktop

  Richiederà preliminarmente di istallare (solo su Windows) il sottosistema Linux
- A questo url troverete le istruzioni per istallare hadoop.

  https://medium.com/analytics-vidhya/how-to-easily-install-hadoop-with-docker-ad094d556f11

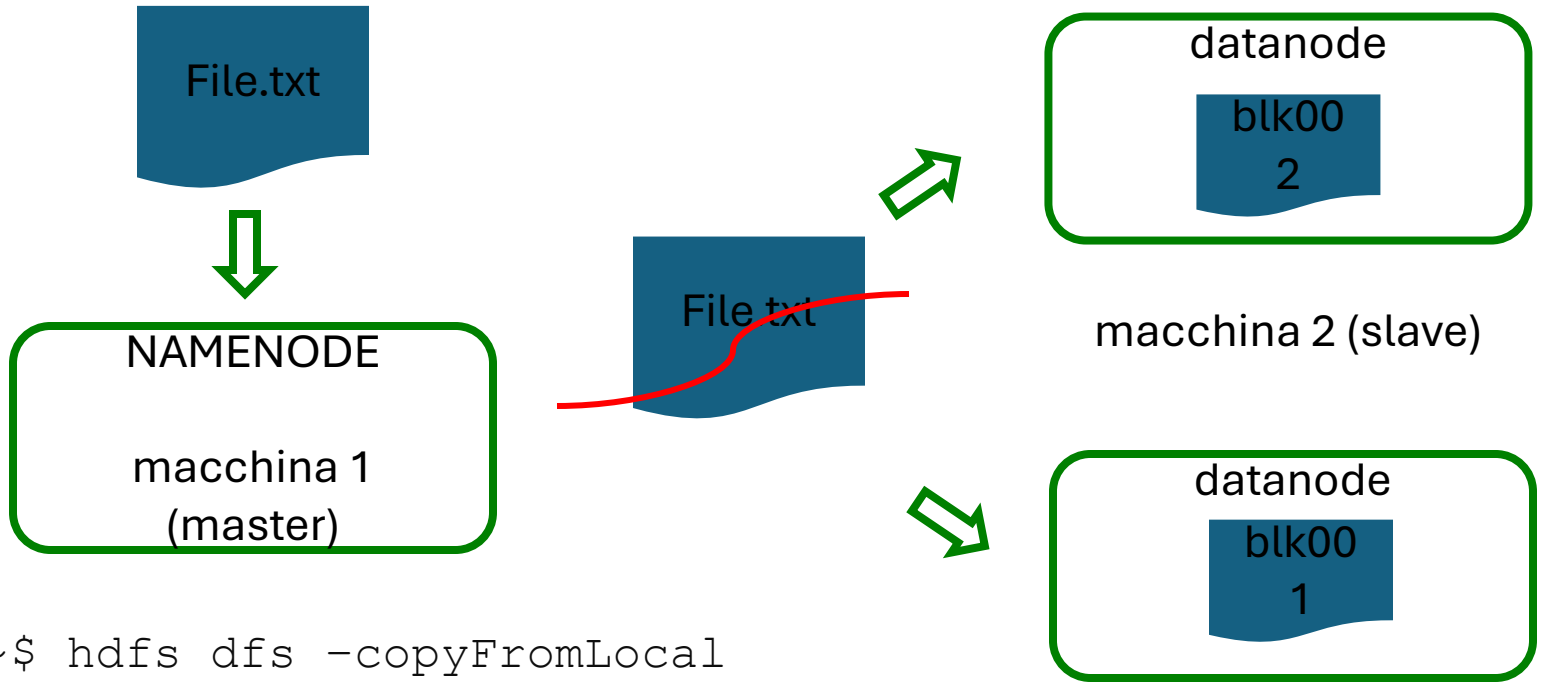
HDFS

- file system distribuito che viene gestito da dei demoni con un'architettura master/slave
- il contenuto del file system NON è (in generale) replicato sui vari nodi, ma spezzettato tra i vari nodi



copio un file dal file system locale in HDFS

Hadoop si occupa dello split in modo trasparente all'utente.




```
:~$ hdfs dfs –copyFromLocal
/path/to/file.txt
/path/in/hdfs/file.txt
```

Verifico il contenuto di HDFS

- Il file mi viene mostrato come se fosse tutto intero su un unico supporto

```
:~$ hdfs dfs -ls /
Found x items
...
drwxr-xr-x - hadoop root 2010-03-16 11:36 /user/hadoop/file.txt!
...
```

- i comandi sono shell-like

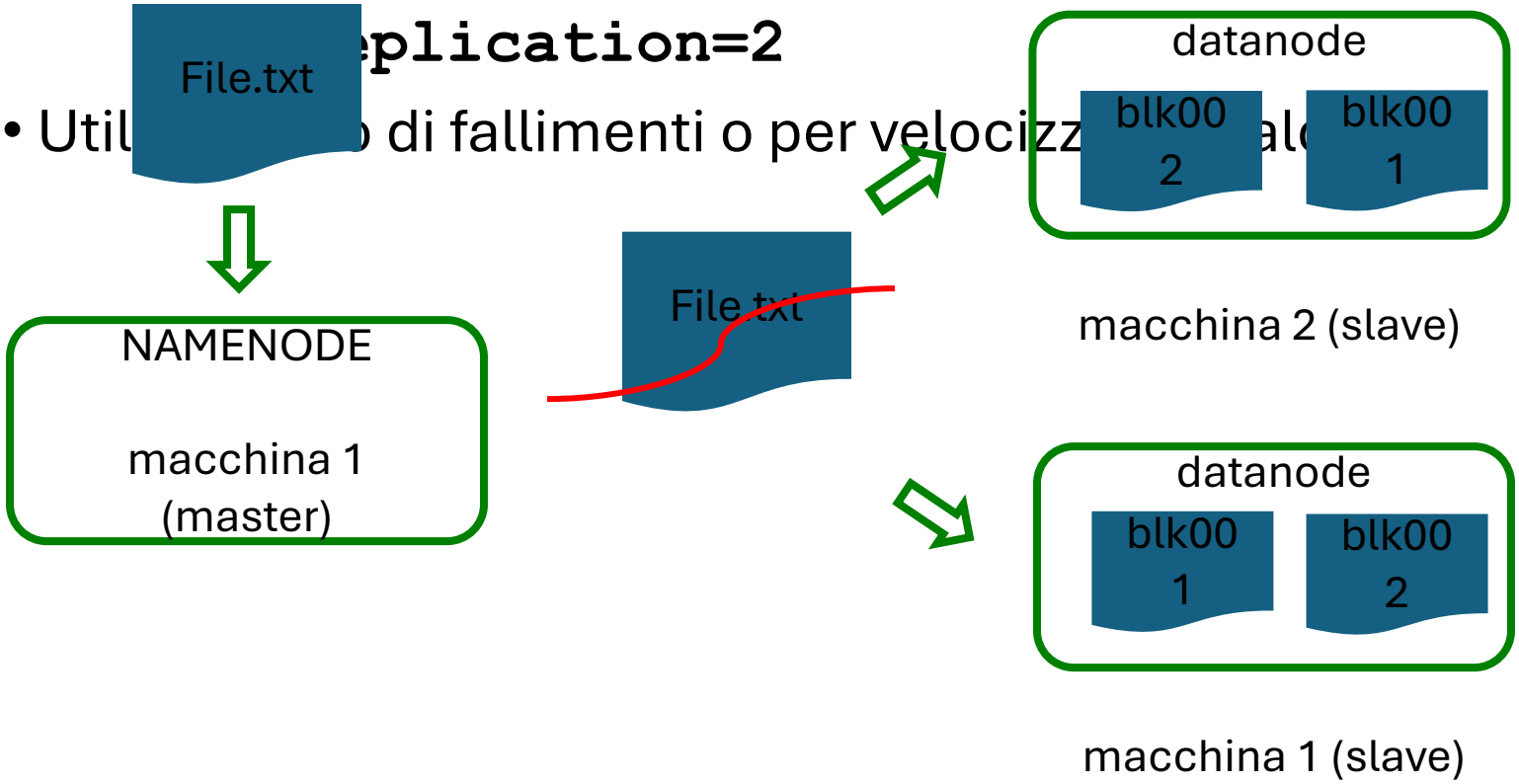
```
:~$ hdfs dfs –mkdir mydir
:~$ hdfs dfs –rmr mydir
```

HDFS

- Può gestire autonomamente anche la replicazione dei dati.
- Parametro di configurazione:

`eplication=2`

- Util o di fallimenti o per velocizz al



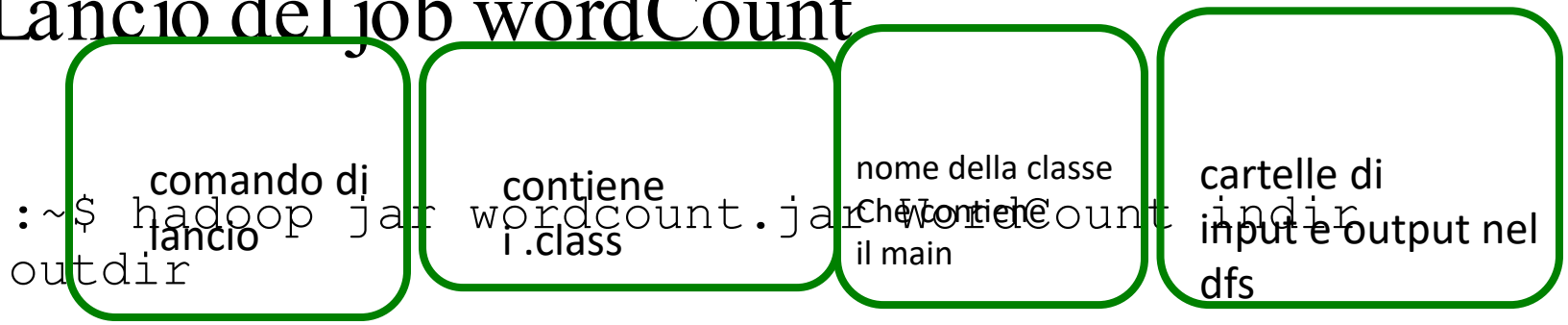
HDFS – Utilizzo

- Operazioni di base (con il commando Hadoop):

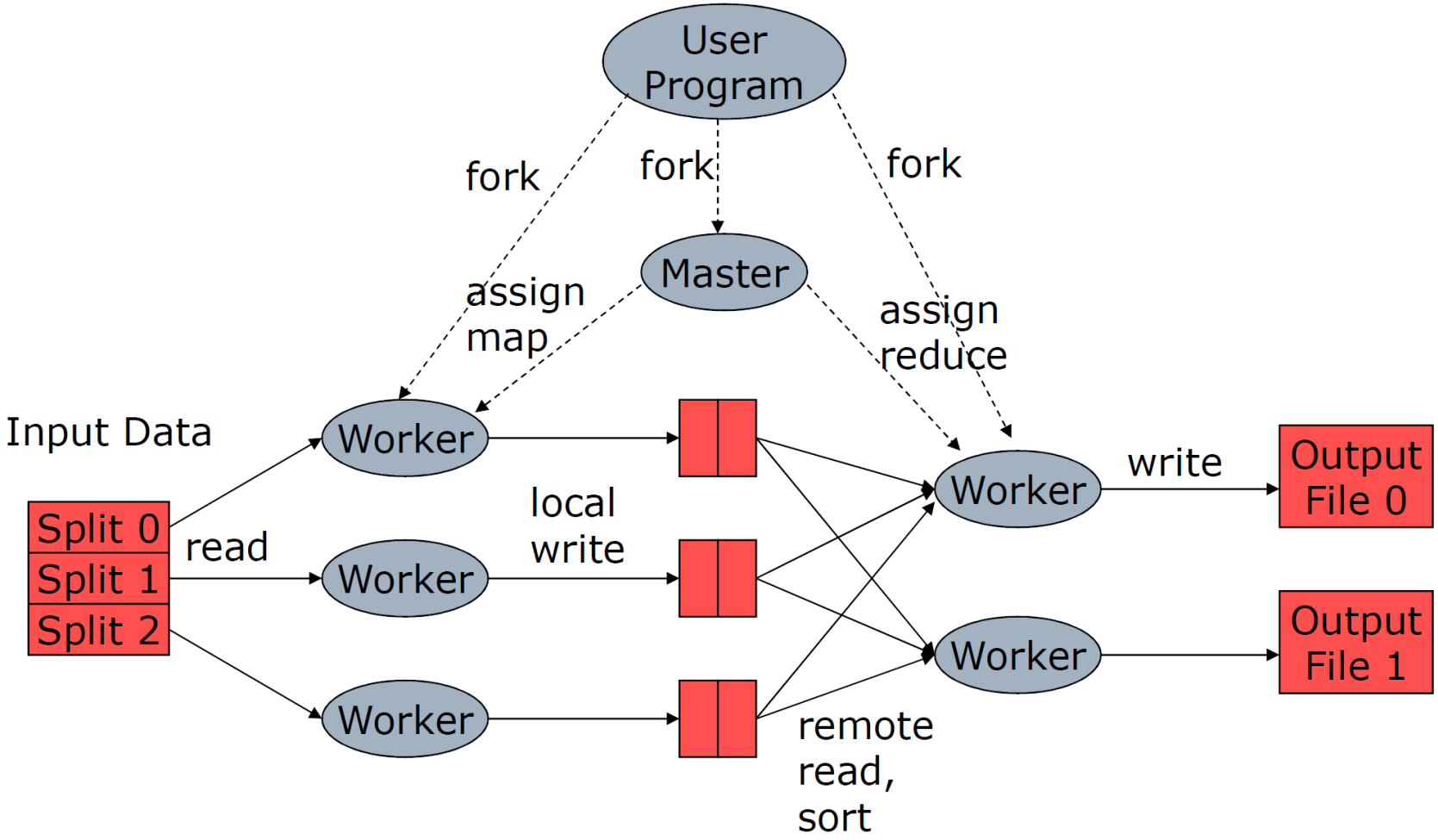
`hadoop fs -ls <path>`	Elencare i file
`hadoop fs -cp <src> <dst>`	Copiare file da HDFS a HDFS
`hadoop fs -mv <src> <dst>`	Spostare i file da HDFS a HDFS
`hadoop fs -rm <path>`	Rimuovere i file in HDFS
`hadoop fs -rmr <path>`	Rimuovi in modo ricorsivo in HDFS
`hadoop fs -cat <path>`	Visualizzare il contenuto di un file in HDFS
`hadoop fs -mkdir <path>`	Creare una cartella HDFS
`hadoop fs -put <localsrc> <dst>`	Copiare i file da Locale a HDFS

`hadoop fs -copyToLocal <src> <localdst>`
 	
Anche:
 	
`hadoop fs -get <src>`
	Copiare i file da HDFS in locale.

Lancio del job wordCount



- Cosa succede in Hadoop quando lancio questo comando?
- Viene chiamato il componente MapReduce runtime.

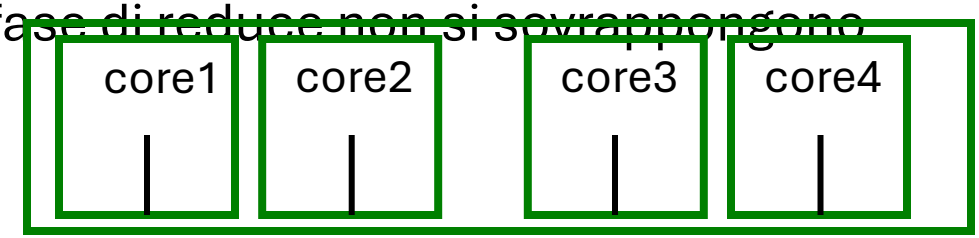

User Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Worker
Split 0
Split 1
Split 2
read
Worker
Worker
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort

Quanti Me R task?

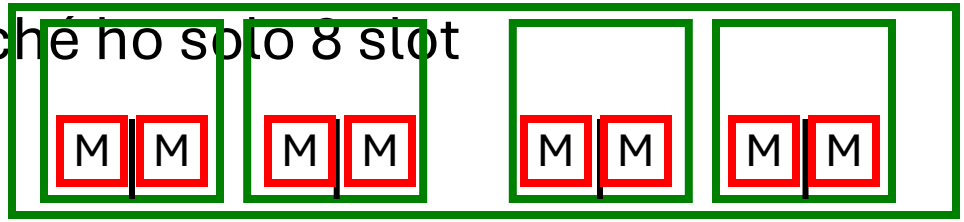
- Vorremmo il maggior grado di parallelismo possibile
  - la risposta dipende da:
    1. quale grado di parallelismo ho nella mia architettura? Quanti core?
    2. quanti dati devo processare?

Quanti M e R?

- Hadoop permette di specificare il grado di parallelismo di ogni macchina in configurazione esprimendo il numero di slot.
- uno slot è un contenitore in cui può finire un map/reduce task in esecuzione
- su una macchina quad-core specificherò:
  - n° map slots = 8
  - n° reduce slot = 8
- poiché la fase di map e la fase di reduce non si sovrappongono (quasi) mai



- Supponiamo di voler fare wordcount su un file da 1GB e di avere 4 slave dual-core
- 8core totali 8 slot
- Soluzione di default di Hadoop:
  - ogni map processa dei chunk di massimo 64MB
  - il file viene spezzato in blocchi di 64MB
- Hadoop lancerà 1GB/64MB = 16 map task tutti insieme?
- NO, perché ho solo 8 slot



- è la soluzione migliore? forse no… 1GB/8core = 128MB
- Se avessi avuto dei chunk da 128MB, avrei fatto tutto in parallelo con soli 8 task e meno cambi di contesto.

- Hadoop permette di cambiare il valore del chunck di default in configurazione, ma
  - occorrono nozioni di computazione distribuita
  - effettivi miglioramenti si notano solo con molti TB

⇒ solitamente hadoop decide da solo il numero di map

- numero di map task = numero di chunck da 64MB in input

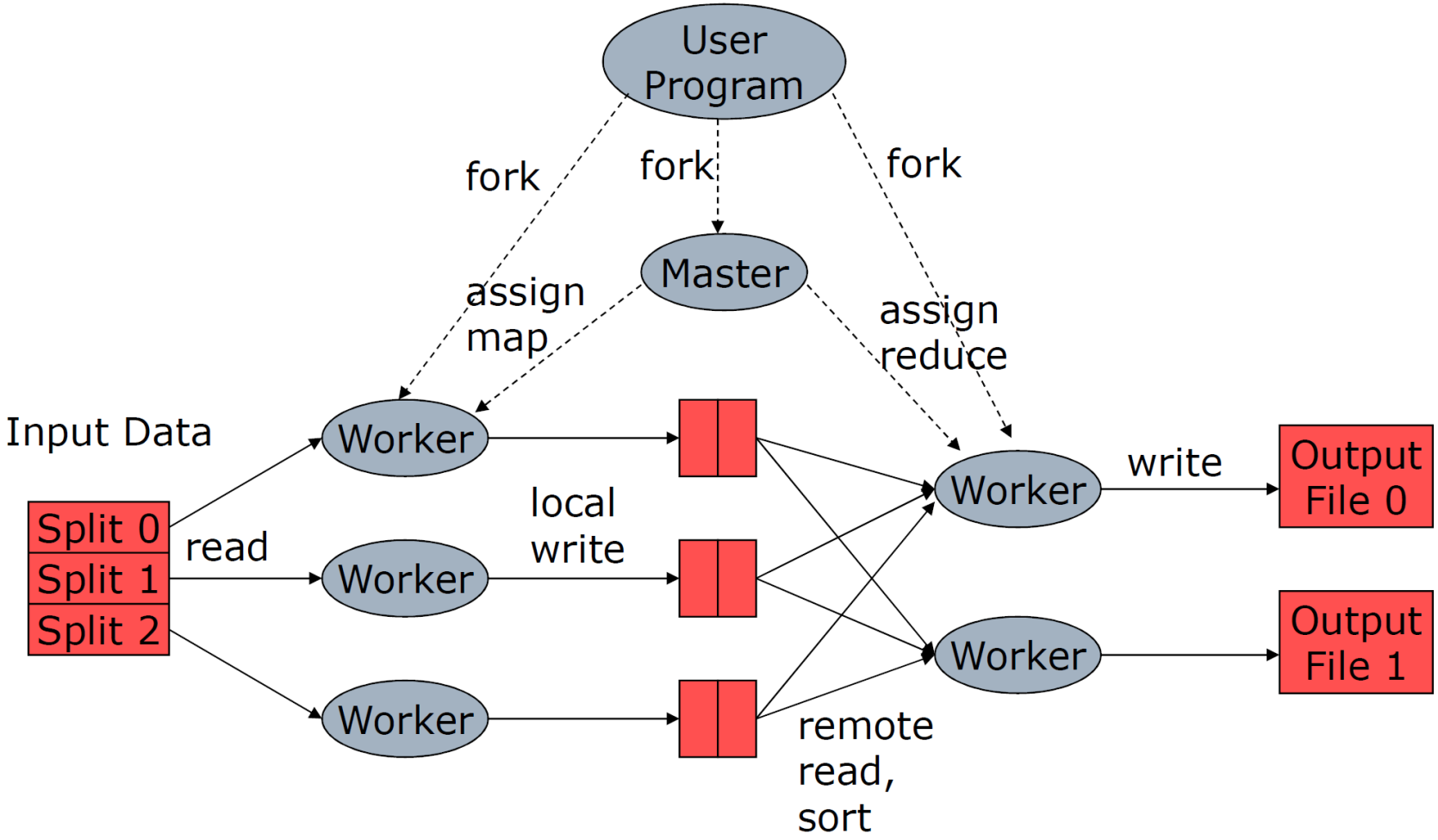
- Questo lo decide l'utente!
  `job.setNumReduceTasks(n);`
- Gli sviluppatori di hadoop consigliano due valori ottenuti statisticamente:

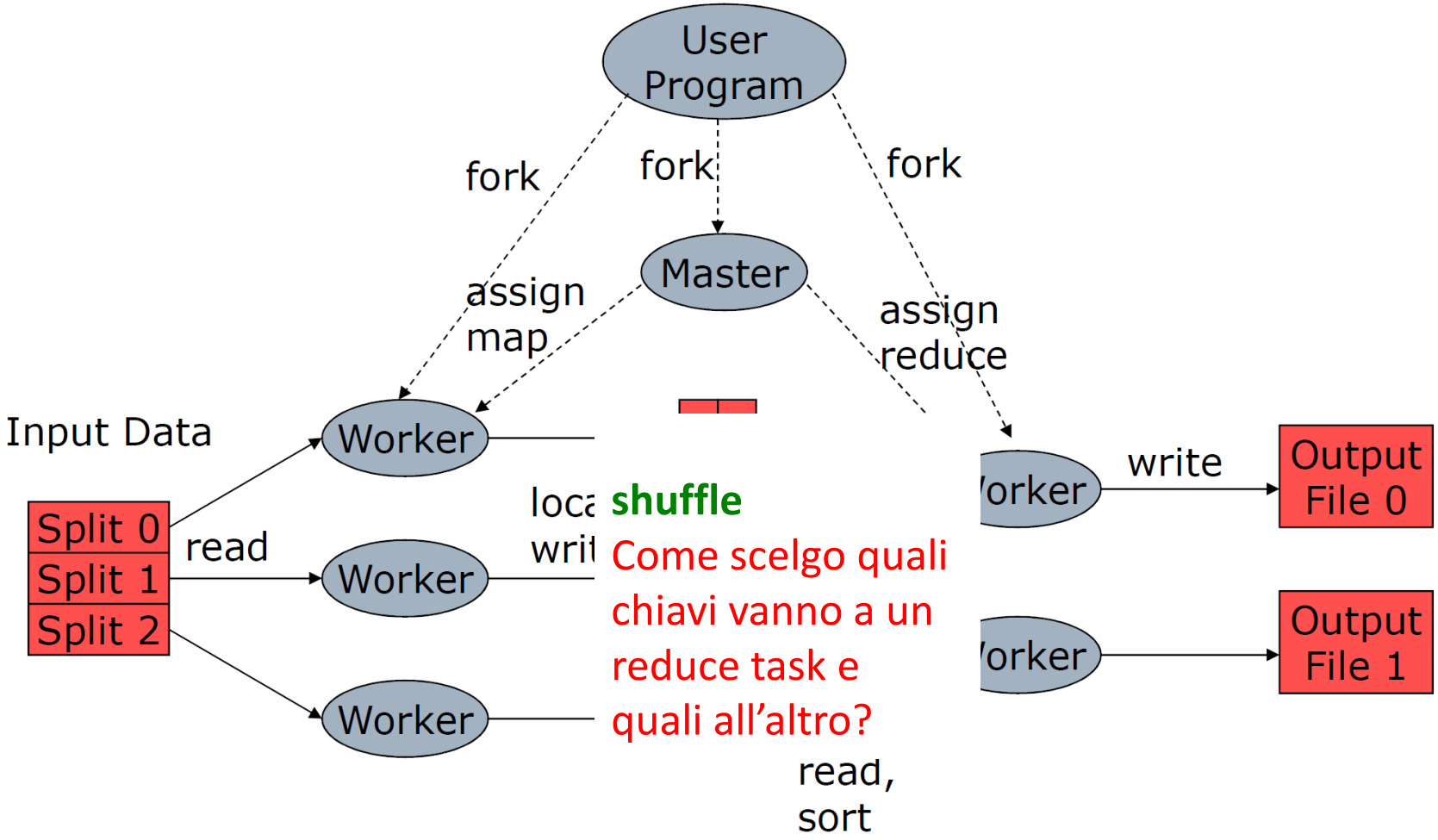
```
0.95 * n° tot di reduce slot
```

se i core sono tutti uguali, oppure

```
1,75 * n° tot di reduce slot
```

se c'è qualche differenza di velocità tra i core delle varie macchine


User Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Worker
Split 0
Split 1
Split 2
read
Worker
Worker
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort


User Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Worker
Split 0
Split 1
Split 2
read
Worker
Worker
shuffle
Come scelgo quali chiavi vanno a un reduce task e quali all'altro?
read, sort
write
Output File 0
Output File 1

Hadoop default partitioning

- Usare una hash function. Per cui Hadoop implementa questo partitioner di default:

```
public class HashPartitioner<K, V>
         extends Partitioner<K, V> {
    public int getPartition(K key, V value,
                int numReduceTasks) {
         return (key.hashCode() & Integer.MAX_VALUE)
                % numReduceTasks;
    }
}
```

bitwise AND per avere solo valori positivi

modulo per avere sempre un risultato nell'intervallo [0,numReduceTasks-1]

- nessun ordinamento le coppie possono essere inviate ai R man mano che vengono prodotte dai M, senza attendere che si sia definita tutta la lista

```
getPartition(“pluto”,”1”,2) = 0
getPartition(“topolino”,”1”,2) = 1
getPartition(“paperino”,”1”,2) = 1
getPartition(“topolino”,”1”,2) = 1
getPartition(“pippo”,”1”,2) = 0
getPartition(“people”,”1”,2) = 1
getPartition(“poeta”,”1”,2) = 1
```

sui grandi numeri questo garantisce anche un certo bilanciamento del carico

NB: **garanzia di ordinamento**.
Il reducer emette sempre i risultati ordinati per chiave

Combiner

- In certi casi può essere vantaggioso far fare qualcosa di più ai map task anticipando il lavoro dei reducer.
- Es: wordcount classico
  - map emette per ogni w nel chunk , <w,1>
  - reduce emette <w,sum(values)>
- Così ho un sacco di traffico tra M e R: una coppia <w,1> per ogni w nel documento.
- Il reducer deve fare tutte le somme

- Se la funzione del reduce è associativa e commutativa, posso parzialmente anticiparla, facendola eseguire sulla stessa macchina del map task.

I Writables

- Hadoop richiede che tutti gli oggetti in output estendano l'interfaccia Writable

- Un writable è una ottimizzazione dell'oggetto Serializable
  - Nessun metadato è memorizzato
  - Evita di creare nuovi oggetti durante le operazioni

- Quali oggetti:
  - BooleanWritable, ByteWritable, ShortWritable, IntWritable, FloatWritable, LongWritable, DoubleWritable, ObjectWritable, NullWritable
  - Text
  - BytesWritable => byte[]
  - ArrayWritable, TwoDArrayWritable
  - MapWritable, SortedMapWritable

Cosa dobbiamo importare

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.IOException;
import java.util.StringTokenizer;
import java.util.regex.Pattern;
```

Struttura del Word Count

```
public class WordCount extends Configured implements Tool {

    private WordCount() {
    }

    public static class TokenizerMapper extends Mapper<Object, Text, Text, LongWritable> {...}

    public static class SumReducer extends Reducer<Text, LongWritable, Text, LongWritable> {...}

    @Override
    public int run(String[] args) throws Exception {...}

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new WordCount(), args);
        System.exit(res);
    }
}
```

Il Mapper

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

    private final static Pattern cleaner = Pattern.compile("[^a-z0-9\\s]");

    private final static LongWritable one = new LongWritable( value: 1);

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(
                cleaner.matcher(value.toString().toLowerCase())
                        .replaceAll( replacement: ""));
        while (itr.hasMoreTokens()) {
            context.write(new Text(itr.nextToken()), one);
        }
    }
}
```

Struttura di un mapper

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    public Mapper() {
    }

    protected void setup(Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
    }

    protected void map(KEYIN key, VALUEIN value, Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
        context.write(key, value);
    }

    protected void cleanup(Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
    }
}
```

Il Reducer

```
public static class SumReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
```

Struttura di un reducer

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    public Reducer() {
    }

    protected void setup(Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
    }

    protected void reduce(KEYIN key, Iterable<VALUEIN> values, Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
        Iterator i$ = values.iterator();

        while(i$.hasNext()) {
            VALUEIN value = i$.next();
            context.write(key, value);
        }

    }

    protected void cleanup(Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>.Context context) throws IOException, InterruptedException {
    }
}
```

Definizione del job

```
@Override
public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    Job job = Job.getInstance(conf, jobName: "word-count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumReducer.class);
    job.setReducerClass(SumReducer.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion( verbose: true) ? 0 : 1;
}
```

Join

Cust ID	First Name	Last Name	Age	Profession
4000001	Kristina	Chung	55	Pilot
4000002	Paige	Chen	74	Teacher
4000003	Sherri	Melton	34	Firefighter
4000004	Gretchen	Hill	66	Engineer
.....	.....	.....	.....	.....

Fig: cust_details

Trans ID	Date	Cust ID	Amount	Game Type	Equipment	City	State	Mode
0000000	06-26-2011	4000001	40.33	Exercise & Fitness	Cardio Machine Accessories	Clarksville	Tennessee	credit
0000001	05-05-2011	4000002	198.44	Exercise & Fitness	Weightlifting Gloves	Long Beach	California	credit
0000002	06-17-2011	4000002	5.58	Exercise & Fitness	Weightlifting Machine Accessories	Anaheim	California	credit
0000003	06-14-2011	4000003	198.19	Gymnastics	Gymnastics Rings	Milwaukee	Wisconsin	credit
0000004	12-28-2011	4000002	98.81	Team Sports	Field Hockey	Nashville	Tennessee	credit
0000005	02-14-2011	4000004	193.63	Outdoor Recreation	Camping & Backpacking & Hiking	Chicago	Illinois	credit
0000006	10-17-2011	4000005	27.89	Puzzles	Jigsaw Puzzles	Charleston	South Carolina	credit
.....	.....	.....	.....	.....	.....	.....	.....	.....

Fig: transaction_details

Matrix-Vector multiplication

- An $n \times n$ matrix M with m_{ij} the element in row i and column j
- A vector $\boldsymbol{v}$ of length n whose jth element is v_j
- The matrix-vector product is the vector x of length n where

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

- Key process of the ranking of Web pages where *n* is tens of billions

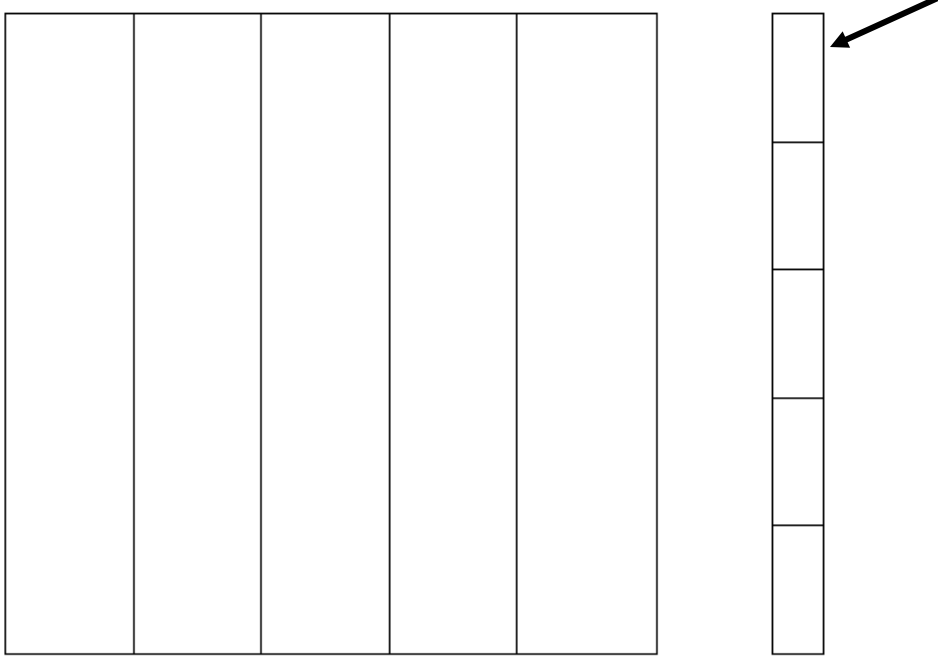
Matrix-Vector multiplication

- The matrix M and the vector v stored in a file of the DFS.
- Entry of the matrix can be stored with explicit coordinates (i,j m_{ij}) and therefore can be discovered.
- The same for v

Matrix-Vector multiplication on MapReduce

- Map:
  - Operates in a chunk of of the matrix M
  - For each element m_{ij} it produces the key-value pair $(i, m_{ij} \times v_j)$
  - All terms of the sum that make up the component x_i will get the same key i
- Reduce:
  - Sums all the values associeated with a given key i. The result will be the pair (i, x_i)

If the vector v cannot fit in Main Memory



The matrix and the vector are dvided into stripes

Each map task is assigned a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector

Matrix multiplication

- A matrix M and a matrix N with the columns of M equals to the number of rows of N. P = MN

$$p_{ik} = \sum_{j} m_{ij} n_{jk}$$

Matrix multiplication

- The matrix can be view as a relation with three attributes. M(I,J,V) with tuples (i, j, m_{ij}) and N(J,K,W) with tuples (j, k, n_{jk})
- The product NM is almost a natural join followed by gruping and aggregation.
- M(I,J,V) natural join N(J,K,W) $(i, j, k, m_{ij}, n_{jk})$
- We want a four component tuple $(i, j, k, m_{ij}, n_{jk})$ → $(i, j, k, m_{ij} \times n_{jk})$

- Once we have $(i, j, k, m_{ij} \times n_{jk})$ we can perform a grouping on i and k and sum all $m_{ij} \times n_{jk}$

Matrix multiplication with 2 map reeduce Steps

- Map:
  - For each entry m_{ij} produce the key value pair $(j, (M, i\ m_{ij}))$ the same we comput for n_{jk} $(j, (N, k\ n_{jk}))$
- Reduce:
  - For each key j examin the list of associated values and produce a list of key-value pairs $((i, k), m_{ij} \times n_{jk})$
- Map:
  - The identity function
- Reduce:
  - For each (i, k) produce the sum of the list of values associated with this key $((i, k), v)$ where v is the falue of the element in row i and column k of the matrix P= MN

Problem

- Design MapReduce algorithms to take a very large file of integers and produce as output:
  - The largest integer
  - The average of all the integers
  - The same set of integers, but with each integer appearing only once
  - The count of the number of distinct integers in the input
  - Prodotto di due matrici con una sola coppia M/R

Cost Measures for Algorithms

- In MapReduce we quantify the cost of an algorithm using

1. Communication cost = total I/O of all processes
2. Elapsed communication cost = max of I/O along any path
3. (Elapsed) computation cost analogous, but count only running time of processes

Note that here the big-O notation is not the most useful (adding more machines is always an option)

Example: Cost Measures

- For a map-reduce algorithm:
  - Communication cost = input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - Elapsed communication cost is the sum of the largest input + output for any map process, plus the same for any reduce process

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other

- Total cost tells what you pay in rent from your friendly neighborhood cloud

- Elapsed cost is wall-clock time using parallelism

Cost of Map-Reduce Join

- Total communication cost
  = O(|R|+|S|+|R ⋈ S|)
- Elapsed communication cost = O(s)
  - We're going to pick k and the number of Map processes so that the I/O limit s is respected
  - We put a limit s on the amount of input or output that any one process can have. s could be:
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost