

Basi Di Dati e di conoscenza

Basi di Dati e Big Data

Contenuti della lezione

ÿ Memorizzare e gestire i Big Data

Hadoop & Map-reduce

ÿ Cloud computing

ÿ NoSQL DBMS

Big Data in azione



Obiettivo:

per prendere decisioni
strategiche efficaci
sfruttando la
disponibilità dei big data

Big Data in azione



Richiede:

ÿ selezione

ÿ filtraggio

ÿ metadati

generazione

ÿ Gestione

provenienza

Big Data in azione



Richiede:

ÿ trasformazione
ÿ normalizzazione
ÿ pulizia ÿ
aggregazione ÿ
gestione degli errori

Big Data in azione



Richiede:
standardizzazione
Gestione
dei conflitti
riconciliazione
Definizione
della mappatura

Big Data in azione



Richiede:

- ÿ esplorazione
- ÿ data mining
- ÿ machine learning
- ÿ visualizzazione

Big Data in azione



Richiede:

- ÿ Conoscenza del dominio
- ÿ conoscenza della provenienza
- ÿ identificazione di modelli di interesse
- ÿ Flessibilità del processo

Big Data in azione



Richiede:

ÿ capacità
manageriali ÿ

Miglioramento
continuo del processo

Sfide

ÿ Prestazioni, prestazioni, prestazioni! ÿ

Scalabilità ÿ

Eterogeneità ÿ

Efficacia

ÿ Flessibilità

ÿ Privacy

ÿ Proprietà

ÿ Collaborazione umana

Volumi di dati

La mole di dati aumenta ogni giorno
Alcuni numeri (2012):
Dati elaborati da Google ogni giorno: 100+ PB
Dati elaborati da Facebook ogni giorno: 10+ PB
Per analizzarli, sistemi che scalano rispetto al volume di dati sono necessario

Da <https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/>

Volumi di dati: Esempio di Google

ÿ Analizzare **10 miliardi di pagine Web** ÿ

Dimensione media di una pagina Web: **20 KB** ÿ

Dimensione della raccolta: $10 \text{ miliardi} \times 20 \text{ KB} = \text{200 TB}$

ÿ Larghezza di banda di lettura del disco rigido **HDD : 550 MB/sec**

ÿ Tempo necessario **per leggere tutte le pagine web (senza analizzarle)**: 2 milioni di secondi

= più di **15 giorni** ÿ Un'architettura a nodo singolo

non è adeguata

Volumi di dati: Esempio di Google

ÿ Analizzare **10 miliardi di pagine Web** ÿ

Dimensione media di una pagina Web: **20 KB** ÿ

Dimensione della raccolta: 10 miliardi x 20 KB = **200 TB**

ÿ Larghezza di banda di lettura del disco rigido **SSD : 550 MB/sec**

ÿ Tempo necessario **per leggere tutte le pagine web (senza analizzarle)**: 2 milioni di secondi

= più di **4 giorni** ÿ Un'architettura a nodo singolo

non è adeguata

Fallimenti

- ÿ I guasti fanno parte della vita quotidiana, specialmente nei data center
 - ÿ Un singolo server resta attivo per 3 anni (~1000 giorni) ÿ 10 server ~ 1 guasto ogni 100 giorni (~3 mesi) ÿ 100 server ~ 1 guasto ogni 10 giorni ÿ 1000 server ~ 1 guasto/giorno ÿ Origini dei guasti
 - ÿ Hardware/Software
 - ÿ Elettrico,
 - ÿ Raffreddamento, ...
 - ÿ Indisponibilità di una risorsa per sovraccarico

Fallimenti

ÿ Dati LALN [DSN 2006] ÿ

Dati per 5000 macchine, per 9 anni

ÿ Guasti hardware: 60%, software: 20%, rete 5% ÿ

Analisi degli errori DRAM [Sigmetrics 2009]

ÿ Dati per 2,5 anni

ÿ 8% dei DIMM interessati da

errori ÿ Analisi dei guasti delle unità disco

[FAST 2007] ÿ Utilizzo e temperatura principali cause di guasti

Fallimenti

ÿ Tipi di guasto

Permanente

ÿ Ad esempio, scheda madre
rotta

ÿ Transitorio

ÿ Ad esempio, indisponibilità di una risorsa a causa del sovraccarico

Larghezza di banda della rete

- ÿ La rete diventa il collo di bottiglia se è necessario scambiare grandi quantità di dati tra nodi/server ÿ Larghezza di banda della rete
 - (in un data center): 10 Gbps ÿ Lo spostamento di 10 TB da un server all'altro richiede più di 2 ore
 - ➔ **I dati** dovrebbero essere **spostati tra i nodi solo quando** è **indispensabile**
 - ÿ Solitamente i codici/programmi sono piccoli (pochi MB)
 - ➔ **Sposta il codice** (programmi) e **il calcolo** nei dati

Larghezza di banda della rete

ÿ La rete diventa il collo di bottiglia se è necessario scambiare grandi quantità di dati tra nodi/server
ÿ Larghezza di banda della rete

(in un data center): 10 Gbps
ÿ Lo spostamento di 10 TB
da un server all'altro richiede più di 2 ore

→ **I dati** dovrebbero essere **spostati tra i nodi solo quando** è **indispensabile**

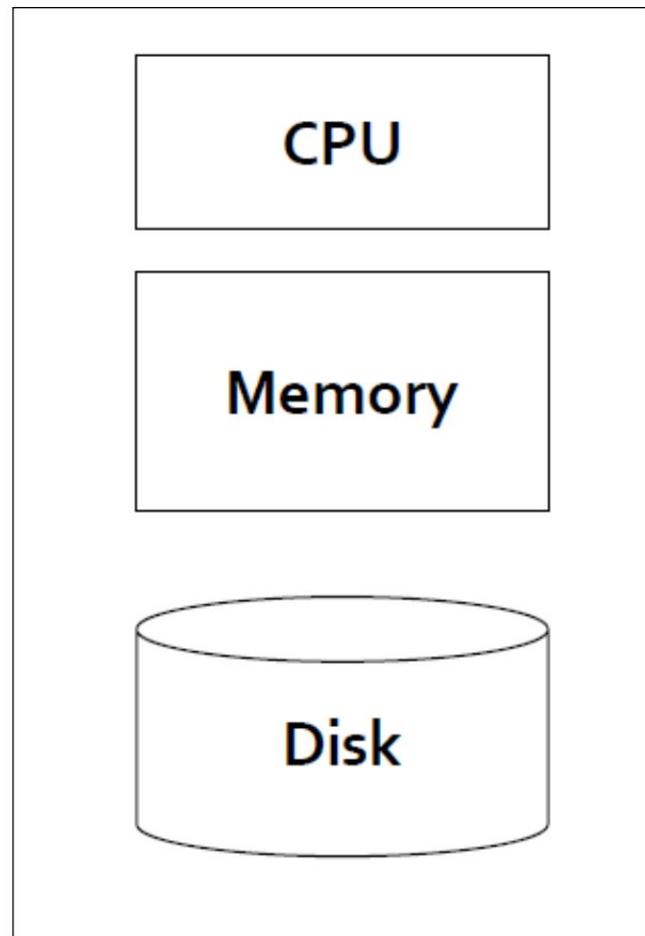
ÿ Solitamente i codici/programmi sono piccoli (pochi MB)

→ **Sposta il codice** (programmi) e **il calcolo** nei dati



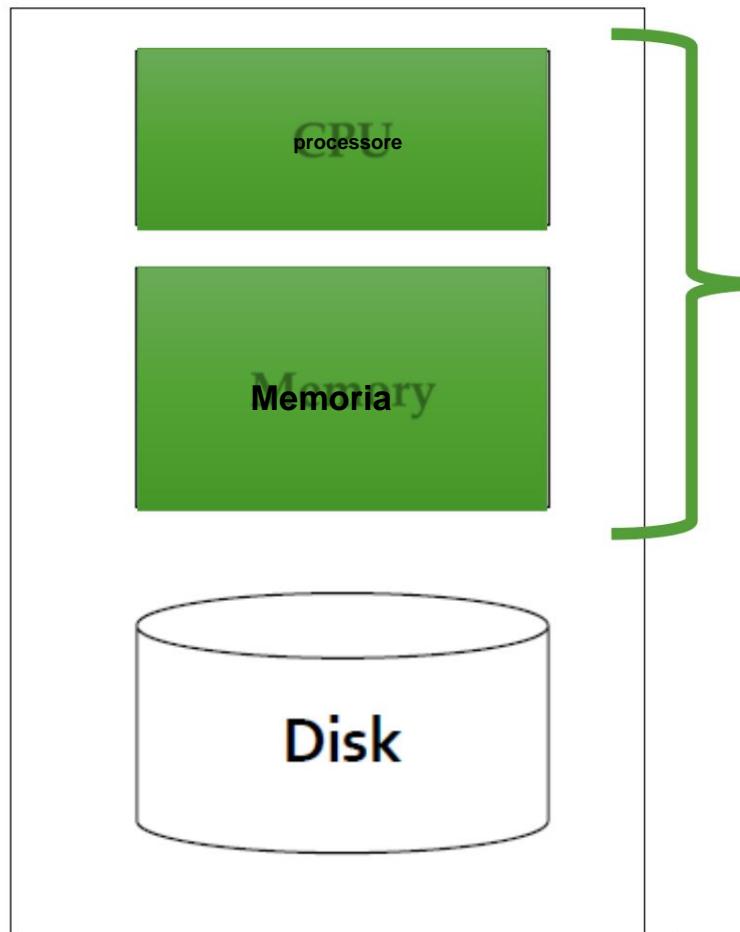
Località dei dati

Architettura a nodo singolo



Server (nodo singolo)

Architettura a nodo singolo



Server (nodo singolo)

Apprendimento automatico, statistiche

ÿ **Piccoli dati**

ÿ I dati possono essere completamente caricati
nella memoria principale

Architettura a nodo singolo



Server (nodo singolo)

Data mining “classico”.

ÿ Dati di grandi

dimensioni ÿ I dati non possono essere caricati completamente nella memoria principale

ÿ Carica nella memoria principale un blocco di dati alla volta tempo

ÿ Elaboralo e memorizza alcune statistiche

ÿ Combina le statistiche per calcolare il risultato finale

Architettura a grappolo

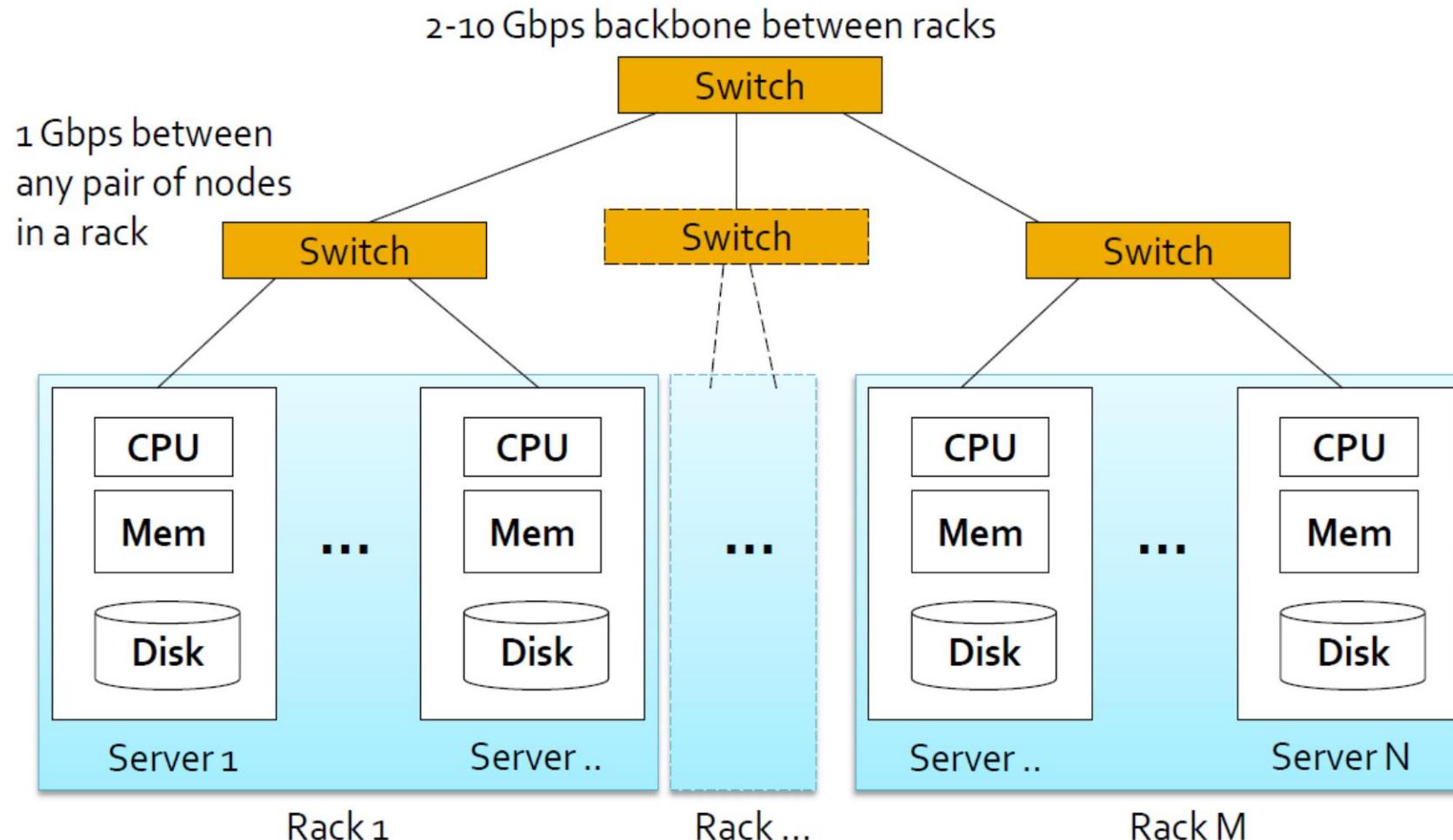
ÿ **Cluster di server** (data center) ÿ

Il calcolo è distribuito tra i server ÿ I dati
sono archiviati/distribuiti tra i server

ÿ Architettura standard nel contesto dei Big Data (**ÿ 2012**) ÿ

Cluster di nodi/server Linux commodity ÿ 32
GB di memoria principale per
nodo ÿ Interconnessione Gigabit Ethernet

Architettura dei cluster di merci



Scalabilità

- ÿ I sistemi attuali devono adattarsi alle esigenze
 - ÿ La crescente quantità di dati da analizzare
 - Il crescente numero di utenti da servire
 - ÿ La crescente complessità dei problemi
- ÿ Di solito vengono utilizzati due approcci per affrontare i problemi di scalabilità
 - ÿ Scalabilità verticale
(scale up)
 - ÿ Scalabilità orizzontale (scale out)

Ridimensionamento e ridimensionamento

ÿ **Scalabilità verticale (scale up)**

- ÿ Aggiungere più potenza/risorse (memoria principale, CPU) a un singolo nodo (alto performante server)
- ÿ Il costo dei supercomputer non è lineare rispetto alle loro risorse

ÿ **Scalabilità orizzontale (scale out)** ÿ

- Aggiungere più nodi (server di base) a un sistema ÿ Il costo scala approssimativamente linearmente rispetto al numero di nodi aggiunti
- ÿ Ma l'efficienza del data center è un problema difficile da risolvere

Ridimensionamento e ridimensionamento

- ÿ Per i carichi di lavoro ad alta intensità di dati, è preferibile un numero elevato di server di base rispetto a un numero ridotto di server ad alte prestazioni ÿ Allo stesso costo, possiamo implementare un sistema che elabora i dati più efficiente ed è più tollerante ai guasti
- ÿ La scalabilità orizzontale (scale out) è preferita per le applicazioni di big data ÿ Ma il calcolo distribuito è difficile ÿ Sono necessari nuovi sistemi che nascondano agli sviluppatori la complessità della parte distribuita del problema

Sfide del cluster computing

ÿ **La programmazione distribuita è difficile**

ÿ Scomposizione e parallelizzazione dei problemi

Sincronizzazione delle

attività

ÿ **La pianificazione delle attività delle applicazioni distribuite è fondamentale**

ÿ Assegna le attività ai nodi cercando di

ÿ Velocizzare l'esecuzione dell'applicazione

ÿ Sfruttare (quasi) tutte le risorse disponibili

ÿ Ridurre l'impatto dei guasti dei nodi

Sfide del cluster computing

ÿ Archiviazione distribuita dei

dati ÿ Come archiviare i dati in modo persistente su disco e mantenerli disponibili se i nodi può fallire?

ÿ **La ridondanza** è la soluzione, ma aumenta la complessità del sistema

ÿ Collo di bottiglia della rete

ÿ Ridurre la quantità di dati inviati attraverso la rete ÿ

Spostare il calcolo e il codice nei dati

Sfide del cluster computing

- ÿ Il calcolo distribuito non è un argomento nuovo ÿ HPC (High Performance Computing)
~1960 ÿ Grid computing
~1990 ÿ Database distribuiti ~1990
- ÿ Quindi, molte soluzioni alle sfide citate sono già disponibile
- ÿ Ma ora stiamo affrontando problemi legati ai big data
 - ➔ Le prime soluzioni non sono adeguate per gestire grandi volumi di dati

Tipico problema dei Big Data

ÿ Problema tipico dei Big Data ÿ

Iterare su un gran numero di record/oggetti
ÿ Estrarre qualcosa di interessante da ciascun record/oggetto
ÿ Aggregare i risultati intermedi
ÿ Generare l'output finale
ÿ **Le sfide:** ÿ

Parallelizzazione

ÿ Archiviazione distribuita di set di dati di grandi dimensioni (Terabyte, Petabyte)
ÿ Gestione dei guasti dei nodi
ÿ Collo di bottiglia della rete
ÿ Diversi formati di input (diversità ed eterogeneità dei dati)

Parallel Data Processing

Elaborazione di Big Data

ÿ Framework per elaborare Big Data

ÿ DBMS relazionale: old fashion

ÿ Archivi dati NoSQL e database NewSQL ÿ **Elaborazione batch:**

archivia ed elabora set di dati su vasta scala (in particolare

Volume+Variety)

ÿ **MapReduce** e **Hadoop**

ÿ **Spark**



La nostra attenzione

ÿ **Elaborazione del flusso di dati:** elabora i dati velocemente (in tempo reale) mentre i dati vengono elaborati generato, **senza memorizzazione**

FRAMEWORKS

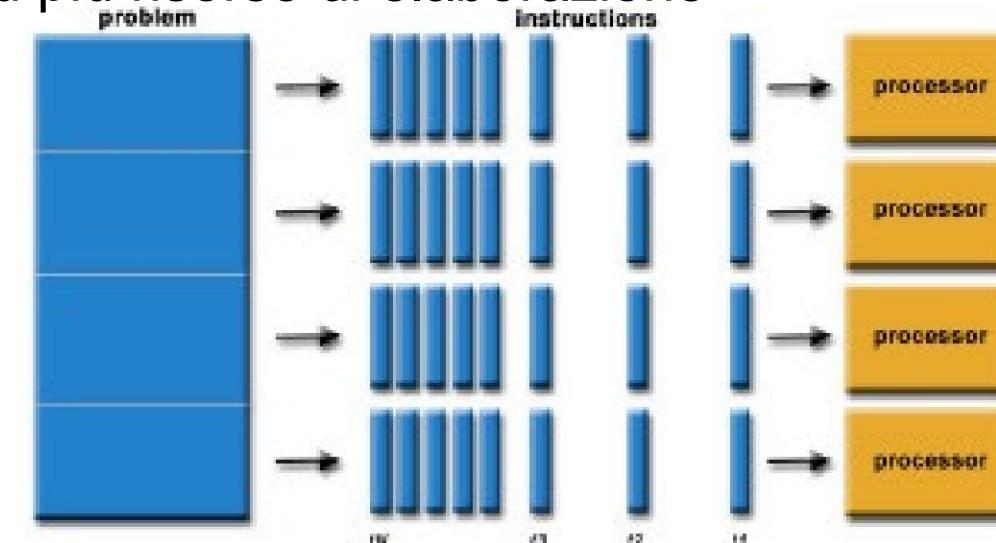


Programmazione parallela: cenni

ÿ Programmazione parallela

ÿ Uso simultaneo di più risorse informatiche (ad es. processori) per risolvere un problema

Come? Suddividi l'elaborazione in **parti** che possono essere **eseguite contemporaneamente** su più risorse di elaborazione



Programmazione parallela: cenni

ÿ L'ambiente più semplice per la programmazione parallela

ÿ Architettura **master/worker**

ÿ Maestro

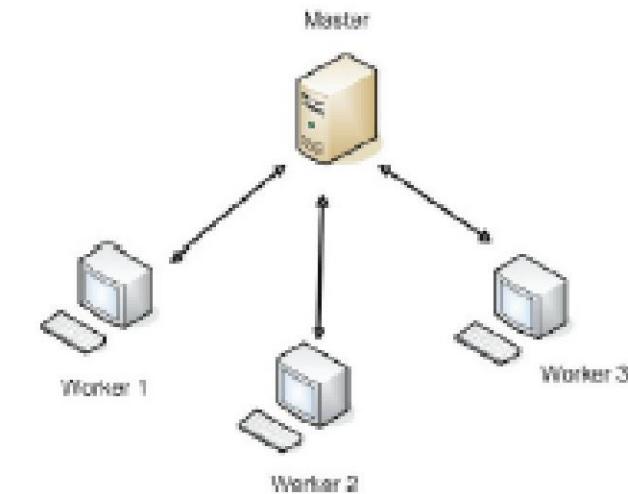
ÿ Prende i dati e li divide in blocchi in base al numero di lavoratori ÿ Invia a ciascun lavoratore un numero uguale di blocchi ÿ Riceve i risultati da ciascun lavoratore

ÿ Lavoratori:

ÿ Ricevi alcuni blocchi di dati dal master

ÿ Eseguire l'elaborazione

ÿ Inviare i risultati al master



Programmazione parallela: cenni

ÿ Esistono diversi stili di programmazione parallela ÿ

Single Program, Multiple Data (SPMD) è il più comunemente usato ÿ **Single Program:** tutte le risorse di calcolo eseguono lo stesso programma

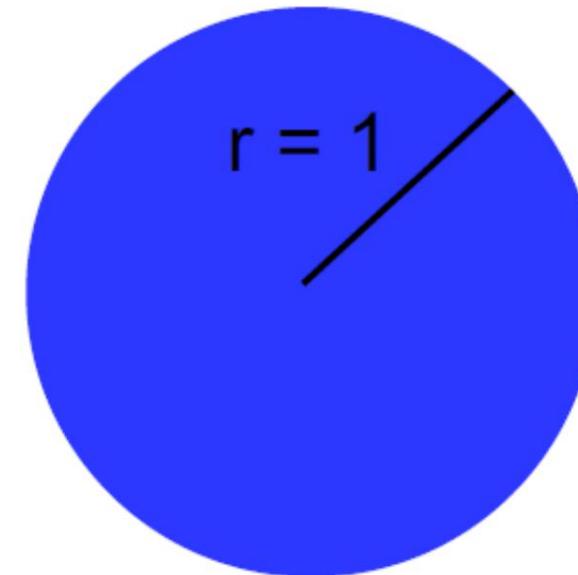
contemporaneamente ÿ **Multiple Data:** tutte le risorse di calcolo possono utilizzare da

Esempio: stima Pi

- ÿ Algoritmo di stima per il calcolo di ÿ
Si basa sul **metodo Monte Carlo**
- ÿ Consideriamo prima la versione sequenziale dell'algoritmo
Poi, come realizzare **una versione parallela e più veloce**

Esempio: stima Pi

↳ Per definizione, π è l'area di un cerchio di raggio uguale a 1



Esempio: stima Pi

ÿ Come stimare ÿ?

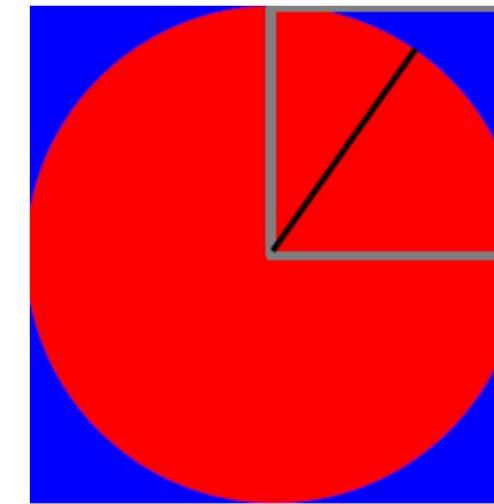
1. Scegli casualmente un gran numero di punti all'interno dell'unità circoscritta quadrato ÿ Un certo numero di questi punti finirà all'interno dell'area descritta dal cerchio, mentre il restante numero di questi punti giacerà all'esterno di essa (ma all'interno del quadrato)
2. Conta la frazione di punti che finiscono all'interno del cerchio su una popolazione totale di punti lanciati casualmente nel quadrato circoscritto

Esempio: stima Pi

ÿ Nelle formule:

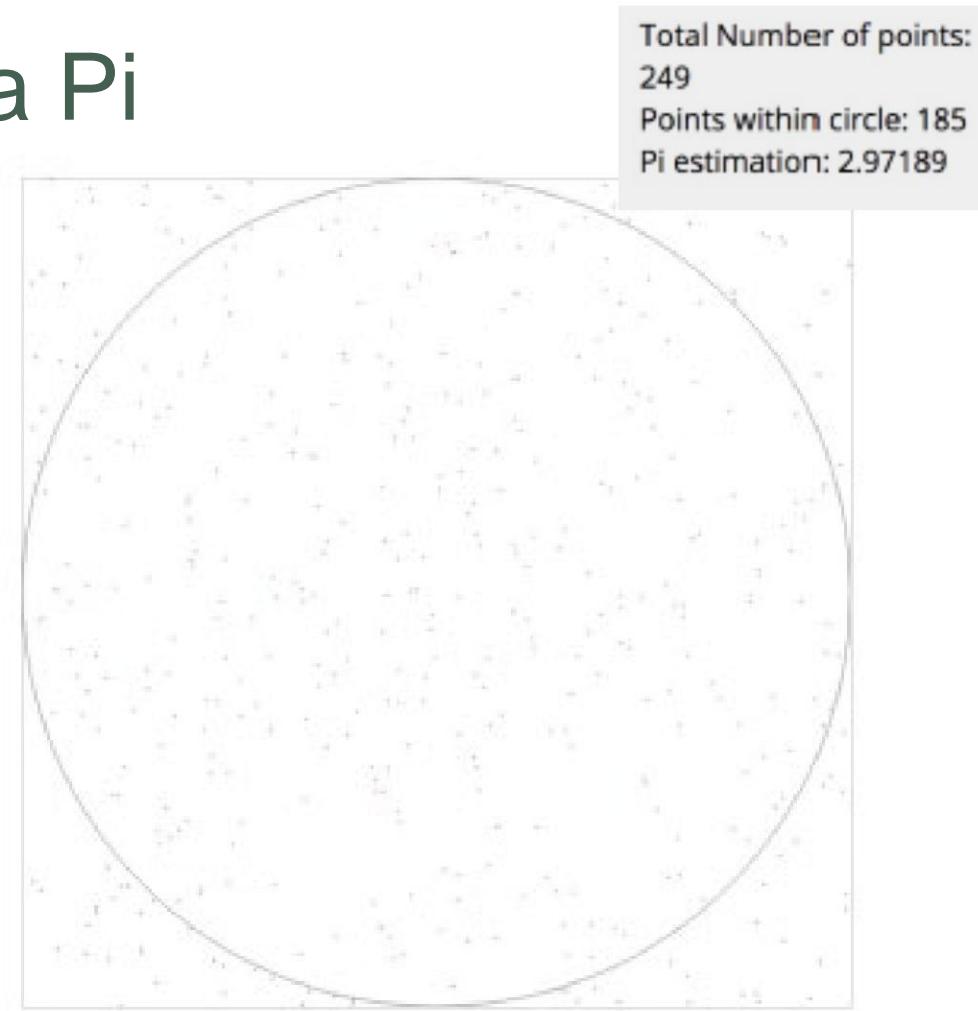
$$\frac{\pi}{4} \approx \frac{N_{inner}}{N_{total}}$$

$$\pi \approx 4 \frac{N_{inner}}{N_{total}}$$



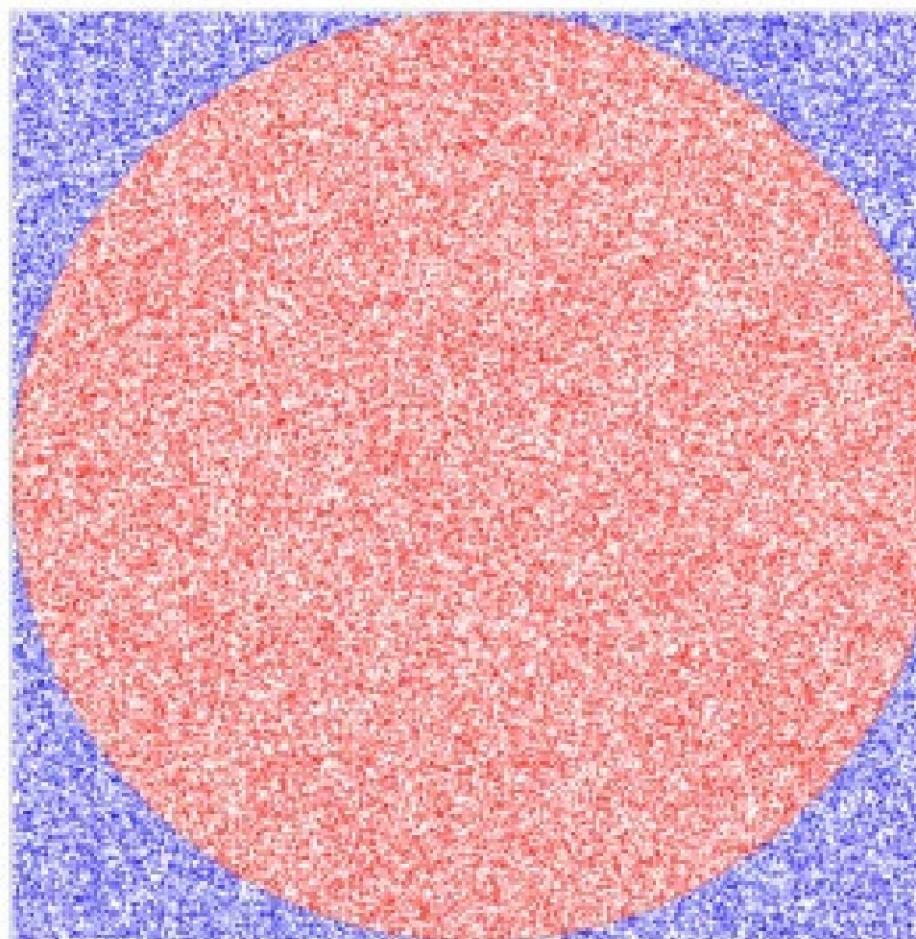
ÿ Maggiore è il numero di punti generati, maggiore è l'accuratezza della stima

Esempio: stima Pi



Guarda l'animazione su <https://academo.org/demos/estimating-pi-monte-carlo/>

Esempio: stima Pi



Total Number of points:
95770
Points within circle: 75212
Pi estimation: 3.14136

Maggiore è il numero di punti generati, maggiore è l'accuratezza della stima

Esempio: stima Pi

ÿ Come ottenere una stima accurata e veloce di π ? ÿ

Dal calcolo sequenziale a **quello parallelo**

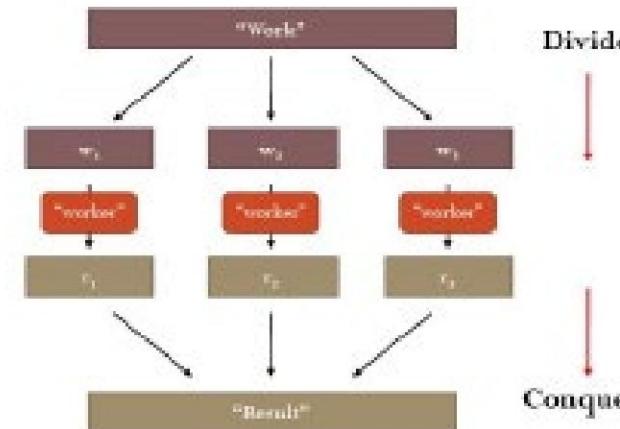
ÿ Utilizzare l'approccio **master/worker**

ÿ Ogni lavoratore esegue l'algoritmo per generare una serie di punti casuali, classificarli e contare quanti finiscono all'interno del cerchio

ÿ Il maestro raccoglie dai lavoratori il numero totale di punti generati e il numero totale di punti nel cerchio. Calcola il rapporto tra i due numeri e lo moltiplica per 4 per ottenere una stima più accurata di π

Idea chiave dietro MapReduce e Scintilla: Dividi e conquista

- ÿ Approccio fattibile per affrontare problemi di dati di grandi dimensioni
 - ÿ Suddividere un problema di grandi dimensioni in sottoproblemi più piccoli
 - ÿ Risolvere sottoproblemi indipendenti in parallelo
 - ÿ Combinare i risultati intermedi di ogni singolo lavoratore



Dividi et impera: come?

ÿ **Scomporre** il problema originale in attività parallele più piccole

Pianificare le attività sui lavoratori distribuiti in un cluster, tenendo account:

ÿ **Località dei**

dati ÿ **Disponibilità delle**

risorse ÿ Garantire che i lavoratori ottengano i dati

di cui hanno bisogno ÿ Coordinare la sincronizzazione tra

i lavoratori ÿ **Condividere**

risultati parziali ÿ Gestire **gli errori**

Idea chiave alla base di MapReduce e Spark: ridimensionare, non aumentare!

ÿ Per i carichi di lavoro a uso intensivo di dati, è disponibile **un gran numero di server commodity** preferito rispetto a un numero limitato di server di fascia alta

ÿ Il costo dei supercomputer non è lineare

ÿ Efficienza del data center

ÿ **L'elaborazione** dei dati è **veloce**, l'I/O è lento ÿ

Niente condiviso è preferibile alla condivisione

ÿ **Nulla condiviso:** ogni nodo è completamente indipendente dagli altri nodi del sistema, nessuna memoria o archiviazione condivisa

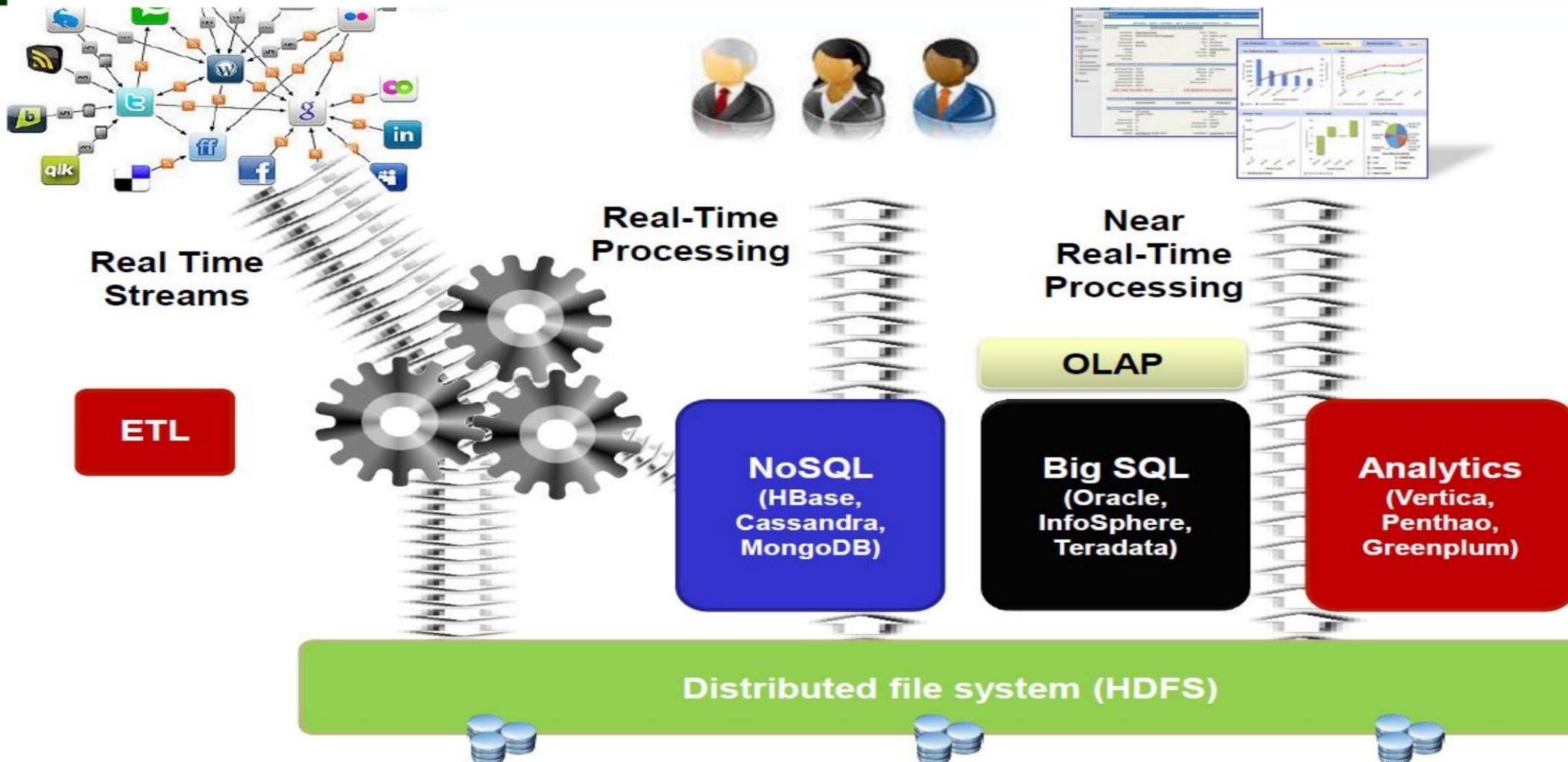
ÿ Scalabilità e tolleranza ai guasti

ÿ **Condivisione:** i nodi condividono uno stato comune/globale che deve essere gestito

ÿ Richiede la sincronizzazione, possono verificarsi deadlock, possono diventare risorse condivise colli di bottiglia (ad es. larghezza di banda per accedere ai dati archiviati)

HADOOP –MAP REDUCE

Grande flusso di dati



Tecnologia: Hadoop e MapReduce

ÿ Cos'è Hadoop? ÿ

Un framework software open-source (progetto Apache) ÿ

Originariamente sviluppato da Yahoo!

ÿ **Obiettivo:** archiviazione ed elaborazione di set di dati su vasta

scala ÿ **Infrastruttura:** cluster di hardware di base ÿ

Core: ÿ

HDFS, un file system distribuito ÿ

MapReduce, un modello di programmazione per l'elaborazione di dati
su larga scala ÿ Include una serie di progetti correlati

ÿ Apache Pig, Apache Hive, Apache HBase, ecc.

ÿ Utilizzato in produzione da Google, Facebook, Yahoo! e molti altri



Il nucleo di Hadoop

ÿ HDFS

- ÿ Un file system distribuito ÿ I server possono fallire e non interrompere il processo di calcolo
- I dati vengono replicati con ridondanza attraverso il cluster

MapReduce

Paradigma di programmazione per esprimere calcoli distribuiti su più server

- ÿ La centrale elettrica dietro la maggior parte dell'odierna elaborazione di big data
- ÿ Utilizzato anche in altri ambienti MPP e database NoSQL (ad esempio, Vertica e MongoDB)

Miglioramento della programmabilità: Pig e Hive

Miglioramento dell'accesso ai dati: HBase, Sqoop e Flume

ApacheHadoop

- ÿ Sistema distribuito scalabile e fault-tolerant per Big Data
 - ÿ Distributed Data Storage
 - ÿ Distributed Data Processing
 - ÿ Concetti/idee presi in prestito dai sistemi progettati da Google (Google File System per MapReduce di Google)
 - ÿ Progetto open source con licenza Apache ÿ Ma ci sono anche molte implementazioni commerciali (es. Cloudera, Hortonworks, MapR)

Storia dell'Hadoop

- ÿ Dicembre 2004 – Google pubblica un articolo su GFS
- ÿ Luglio 2005 – Nutch utilizza MapReduce
- ÿ Febbraio 2006 – Hadoop diventa un sottoprogetto Lucene
- ÿ Aprile 2007 – Yahoo! viene eseguito su un cluster da 1000 nodi
- ÿ gennaio 2008 – Hadoop diventa un progetto Apache Top Level
- ÿ luglio 2008 – Hadoop viene testato su un cluster da 4000 nodi

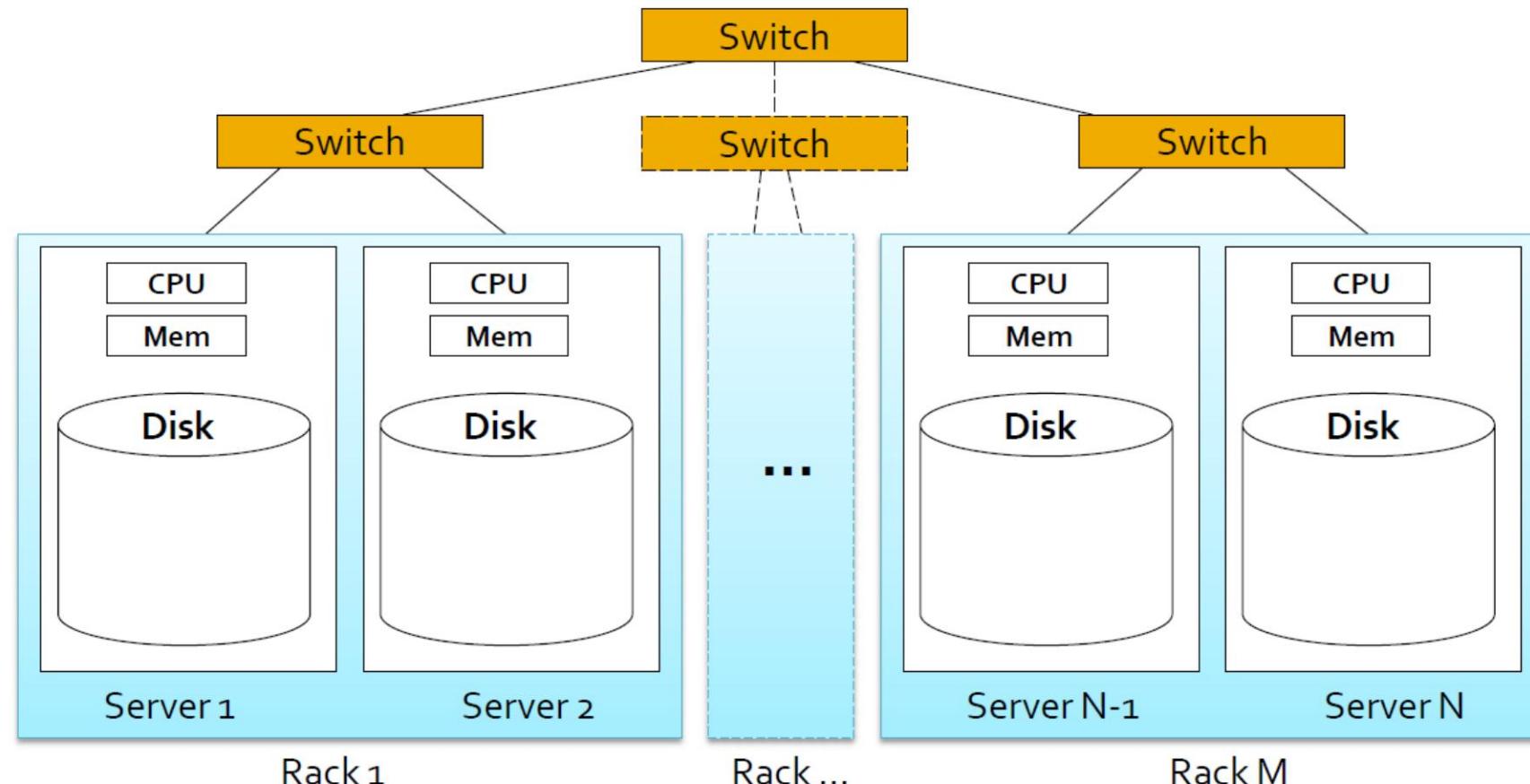
Storia dell'Hadoop

- ÿ Febbraio 2009 – Yahoo! Search Webmap è un'applicazione Hadoop che gira su più di 10.000 cluster Linux core ÿ Giugno
- 2009 – Yahoo! ha reso disponibile il codice sorgente della sua versione di produzione di Hadoop
- ÿ Nel 2010 Facebook ha affermato di avere il più grande cluster Hadoop al mondo con 21 PB di storage ÿ Il 27 luglio 2011 hanno annunciato che i dati sono cresciuti fino a 30 PB.

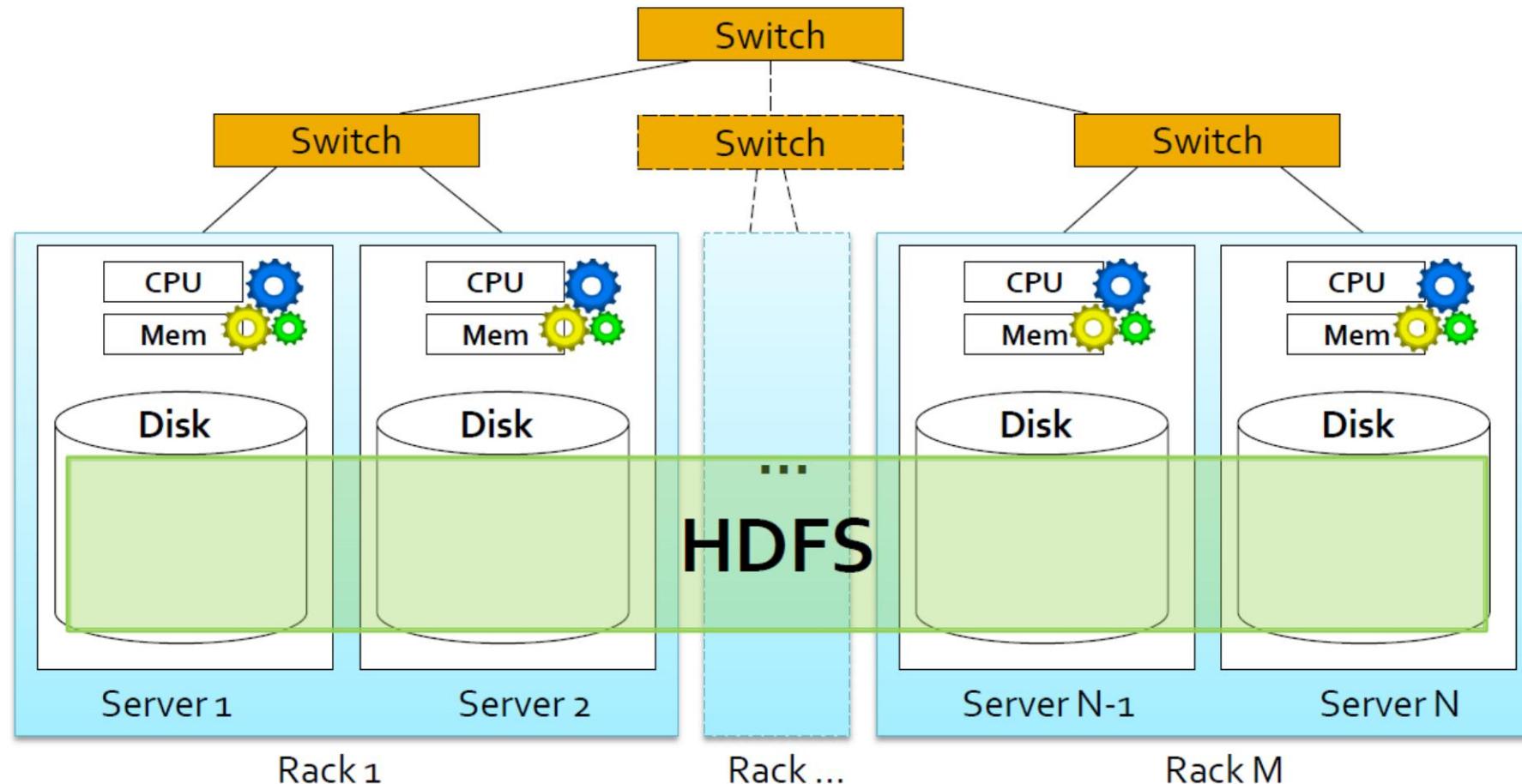
Hadoop: componenti principali

- ÿ Componenti principali di Hadoop:
 - ÿ **Infrastruttura di elaborazione di Big Data distribuita basata sul paradigma di programmazione MapReduce**
 - ÿ Fornisce una vista di astrazione di alto livello ÿ I programmatore non devono preoccuparsi della pianificazione e della sincronizzazione delle attività
 - ÿ Tolleranza ai guasti
 - ÿ Gli errori di nodi e attività vengono gestiti automaticamente dal sistema Hadoop
 - ÿ **HDFS (Hadoop Distributed File System)** ÿ Archiviazione distribuita ad alta disponibilità ÿ Tolleranza ai guasti

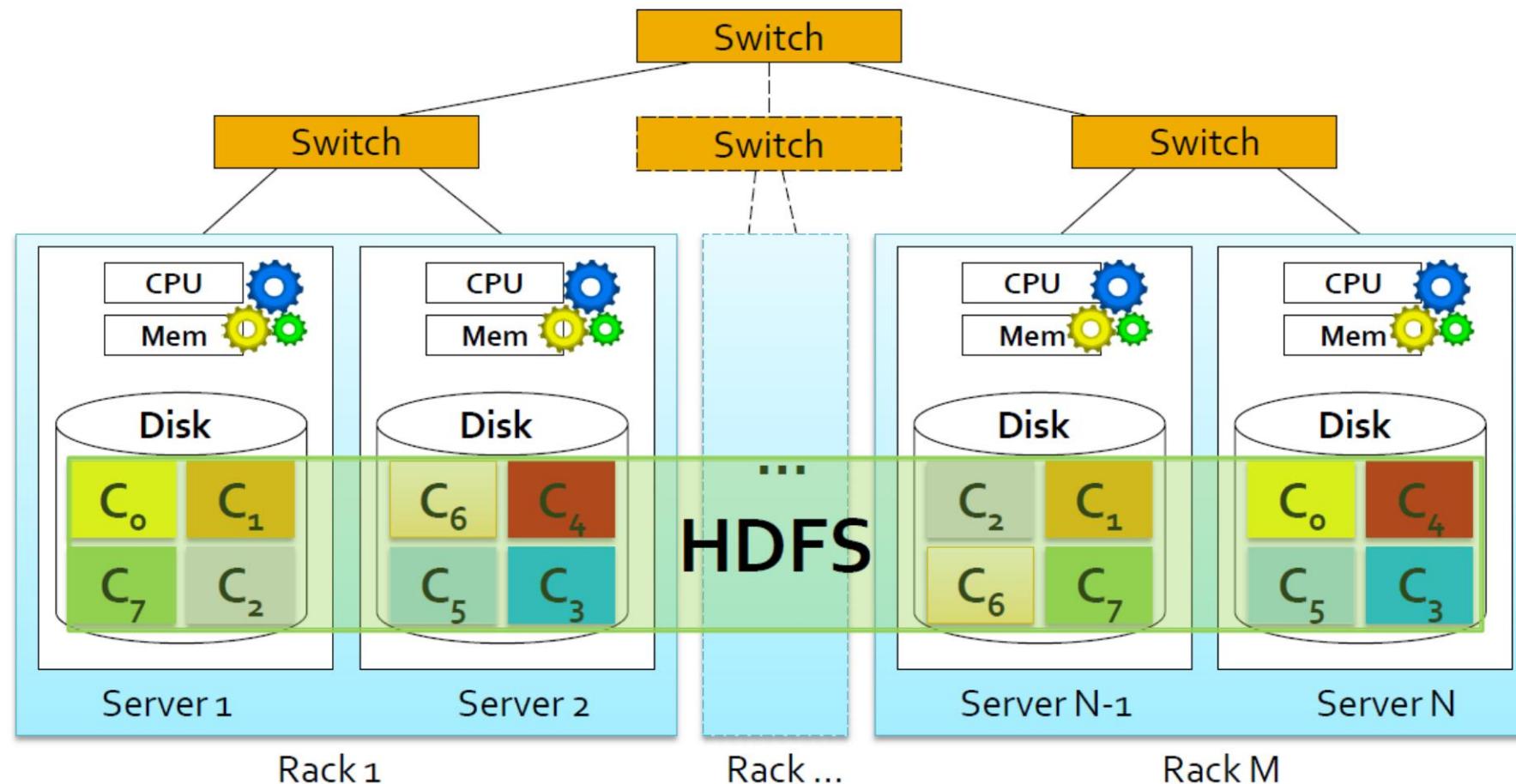
Hadoop: componenti principali



Hadoop: componenti principali



Hadoop: componenti principali



Elaborazione distribuita di Big Data Infrastruttura

ÿ **Separa il cosa dal come**

ÿ I programmi Hadoop sono basati sul paradigma di programmazione MapReduce
ÿ MapReduce astrae la parte “distribuita” del problema
(pianificazione, sincronizzazione, ecc.) ÿ I
programmatori si concentrano su cosa

ÿ La parte distribuita (pianificazione, sincronizzazione, ecc.) del problema è
gestita dal framework
ÿ L'infrastruttura Hadoop si concentra su come

Elaborazione distribuita di Big Data Infrastruttura

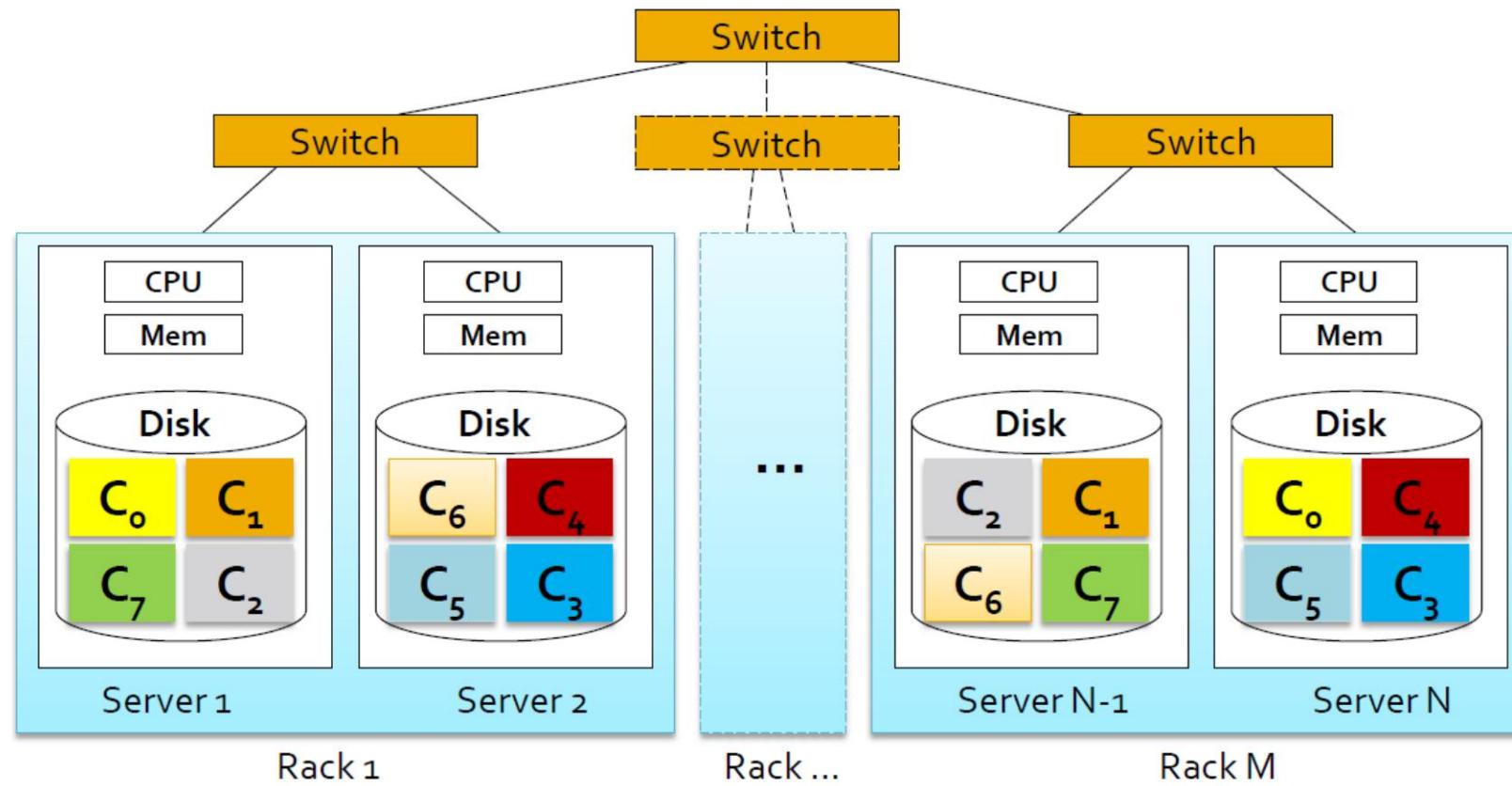
- ÿ Ma una conoscenza approfondita del framework Hadoop è importante per sviluppare applicazioni efficienti ÿ
 - Il design dell'applicazione deve sfruttare la località dei dati e limitare l'utilizzo della rete/la condivisione dei dati

HDFS

- ÿ Ogni file è suddiviso in "pezzi/blocchi" distribuiti sui server
 - ÿ Ogni mandrino viene replicato su server diversi (di solito ci sono 3 repliche per chunk)
 - ÿ Garantisce persistenza e disponibilità

Per aumentare la persistenza e la disponibilità, le repliche vengono archiviate in rack diversi, se possibile
 - ÿ Tipicamente ogni blocco è di 64-128 MB
 - ÿ Un file speciale (**il nodo master**) memorizza, per ogni file, le posizioni dei suoi
 - ÿ Chunk
 - ÿ Il nodo master è esso stesso replicato
 - ÿ Una directory per il file system sa dove trovare il nodo master
 - ÿ La directory stessa può essere replicata

HDFS



Example with number of replicas per chunk = 2

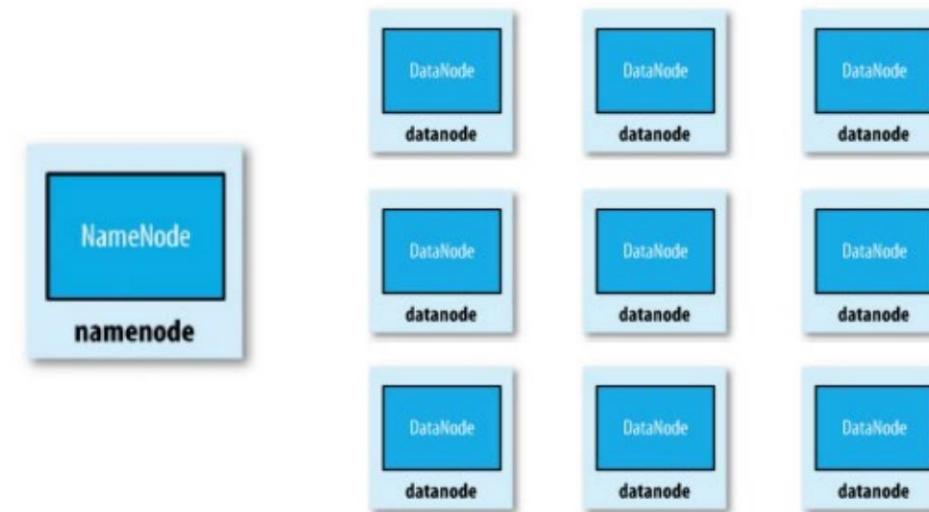
HDFS

- ÿ Il **nodo Master**, noto anche come Name Nodes in HDFS, è un nodo/server speciale
Quello
 - ÿ Memorizza i metadati HDFS
 - ÿ Ad esempio, la mappatura tra il nome di un file e la posizione dei suoi blocchi
 - ÿ Potrebbe essere replicato
 - ÿ **Applicazioni client:** accesso ai file tramite API HDFS
 - ÿ Parlare con il nodo master per trovare server dati/chunk associati al file di interesse
 - ÿ Connettersi ai chunk server selezionati per accedere ai dati

HDFS

ÿ Un cluster HDFS ha due tipi di nodi:

ÿ **Più DataNode** ÿ **Il NameNode**



HDFS

- ÿ I **datanode** semplicemente memorizzano e recuperano i blocchi quando gli viene richiesto (dai client o dal namenode) ÿ II
- namenode:**
 - ÿ Gestisce l'albero del filesystem ei metadati per tutti i file e directory
 - ÿ Conosce i datanode su cui si trovano tutti i blocchi per un dato file
 - ÿ Senza namenode HDFS non può essere utilizzato
 - ÿ È importante rendere il namenode resiliente al fallimento

I/O HDFS

- ÿ Un client dell'applicazione che desidera leggere un file (o una parte di esso) deve prima contattare il **namenode** per determinare dove sono archiviati i dati effettivi
- ÿ In risposta alla richiesta del client, namenode restituisce:
 - ÿ l'identificativo del blocco rilevante
 - ÿ la posizione in cui è contenuto il blocco (cioè quale datanode)
 - ÿ Il client contatta quindi il datanode per recuperare i dati.
 - ÿ I blocchi sono essi stessi archiviati su file system standard per singola macchina
 - ÿ HDFS si trova in cima allo stack del sistema operativo standard
 - ÿ Caratteristica importante del design:
 - ÿ i dati non vengono mai spostati attraverso il namenode
 - ÿ tutto il trasferimento dei dati avviene direttamente tra client e datanode
 - ÿ le comunicazioni con il namenode implicano solo il trasferimento di me

Ecosistema Hadoop

ÿ Sono disponibili molti progetti/sistemi relativi a Hadoop

ÿ **Alveare**

ÿ Un database relazionale distribuito, basato su MapReduce, per l'interrogazione dei dati memorizzati in HDFS tramite un linguaggio di interrogazione basato su SQL

ÿ **HBase**

ÿ Un database distribuito orientato alle colonne che utilizza HDFS per l'archiviazione dei dati

Pig ÿ Un linguaggio di flusso di dati e un ambiente di esecuzione, basato su MapReduce, per esplorare set di dati molto grandi

Ecosistema Hadoop

ÿ **Sqoop**

ÿ Uno strumento per spostare in modo efficiente i dati da database relazionali tradizionali e fonti di file flat esterni a HDFS

ÿ **ZooKeeper** ÿ

Un servizio di coordinamento distribuito. Fornisce primitive come lock distribuiti

ÿ

ÿ **Ogni progetto/sistema affronta una specifica classe di problemi**

MapReduce: introduction

Le "grandi idee" dell'informatica su larga scala

ÿ Ridimensiona "verso l'esterno", non "verso l'alto"

- ÿ Limiti del multiprocessore simmetrico e delle grandi macchine a memoria condivisa
- ÿ Nascondere i dettagli a livello di sistema allo sviluppatore dell'applicazione
 - ÿ È difficile ragionare sui programmi concorrenti e più difficile eseguire il debug
 - ÿ Spostare l'elaborazione sui dati
 - ÿ I cluster hanno una larghezza di banda limitata
 - ÿ Elaborare i dati in sequenza, evitare l'accesso casuale
 - ÿ Le ricerche sono costose, il throughput del disco è ragionevole
 - ÿ Scalabilità continua
 - ÿ Dal mitico mese-uomo all'ora-macchina negoziabile

Riscaldamento: conteggio delle parole

ÿ Input ÿ

Un file testuale di parole di grandi dimensioni
ÿ Problema

ÿ Contare il numero di volte in cui ciascuna parola distinta appare nel file

ÿ Output

ÿ Un elenco di coppie <parola,

contatore> ÿ contare il numero di occorrenze di ciascuna parola specifica nel file di input

Conteggio parole

- ÿ **Caso 1: l'intero file sta nella memoria principale** ÿ Un approccio tradizionale a nodo singolo è probabilmente la soluzione più efficiente in questo caso
 - ÿ La complessità e le spese generali di un sistema distribuito influiscono sulle prestazioni quando i file sono "piccoli"
 - ÿ "piccolo" dipende dalle risorse che hai
- ÿ **Caso 2: File troppo grande per entrare nella memoria principale**
 - ÿ Come suddividere questo problema in un insieme di **sub (quasi) indipendenti task** e ÿ
 - eseguirli in parallelo** su un cluster di server?

Conteggio parole

ÿ Supponiamo

che 1. Il cluster abbia **3 server**

2. Il contenuto del file di input è

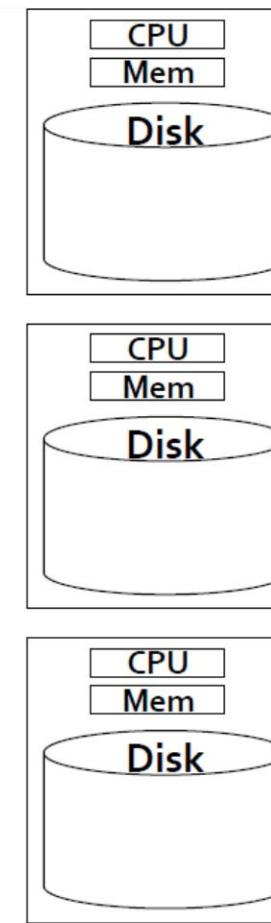
ÿ “File di esempio giocattolo per Hadoop. Esempio di esecuzione di Hadoop.”

3. Il file di input è suddiviso in **2 blocchi**

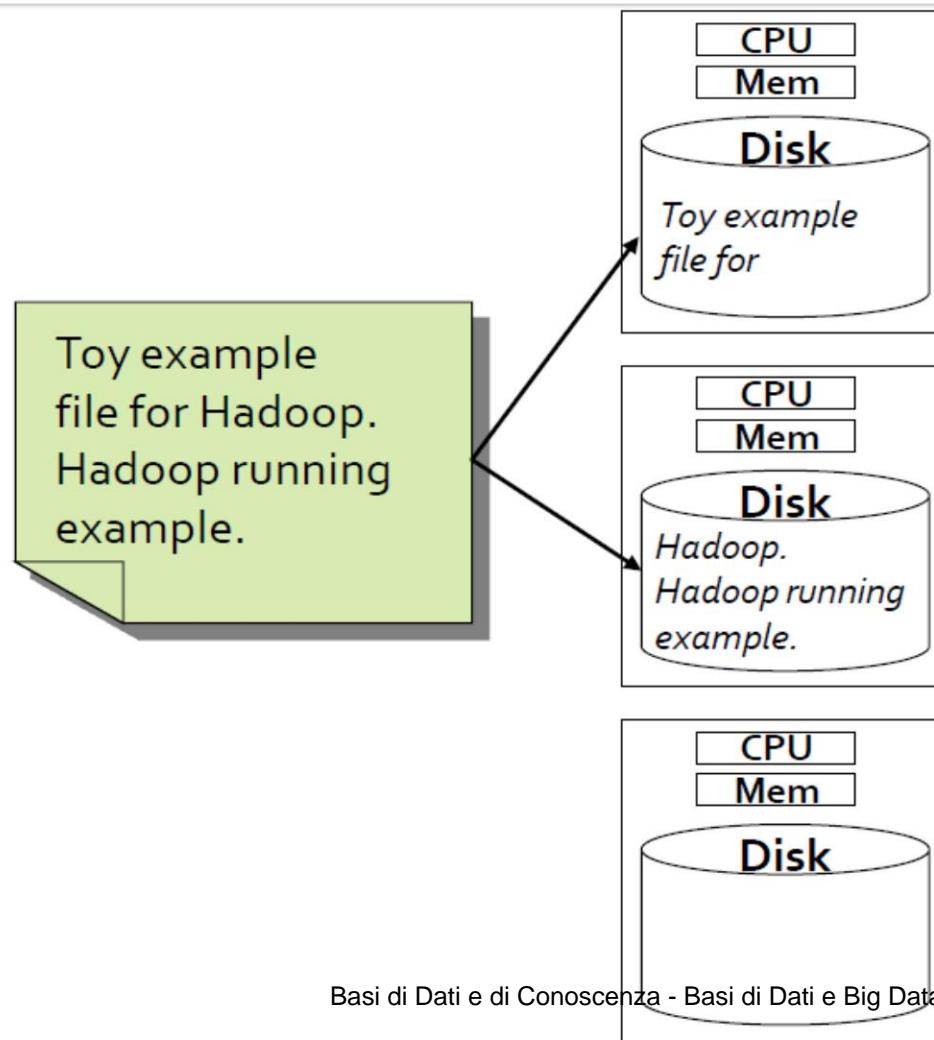
4. Il numero di **repliche** è **1**

Conteggio parole con un file molto grande

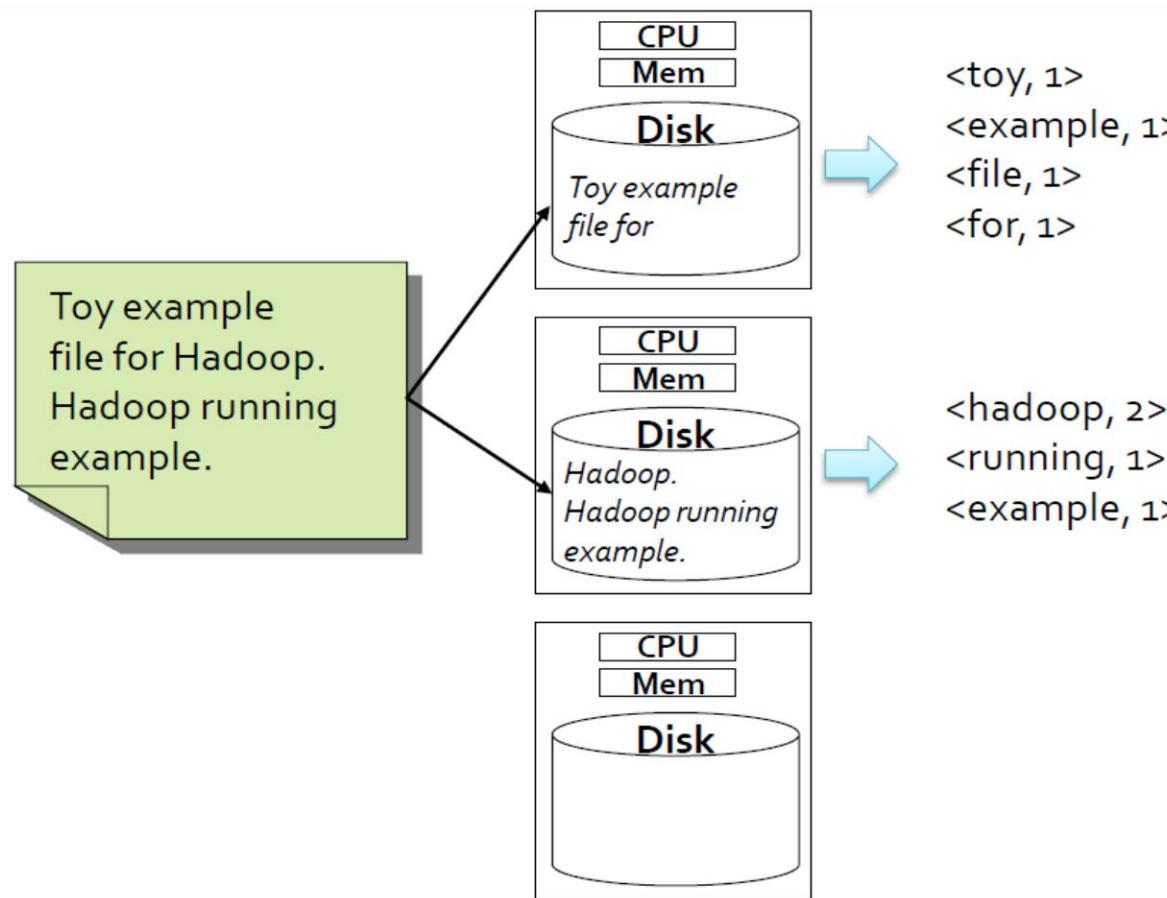
Toy example
file for Hadoop.
Hadoop running
example.



Conteggio parole con un file molto grande



Conteggio parole con un file molto grande

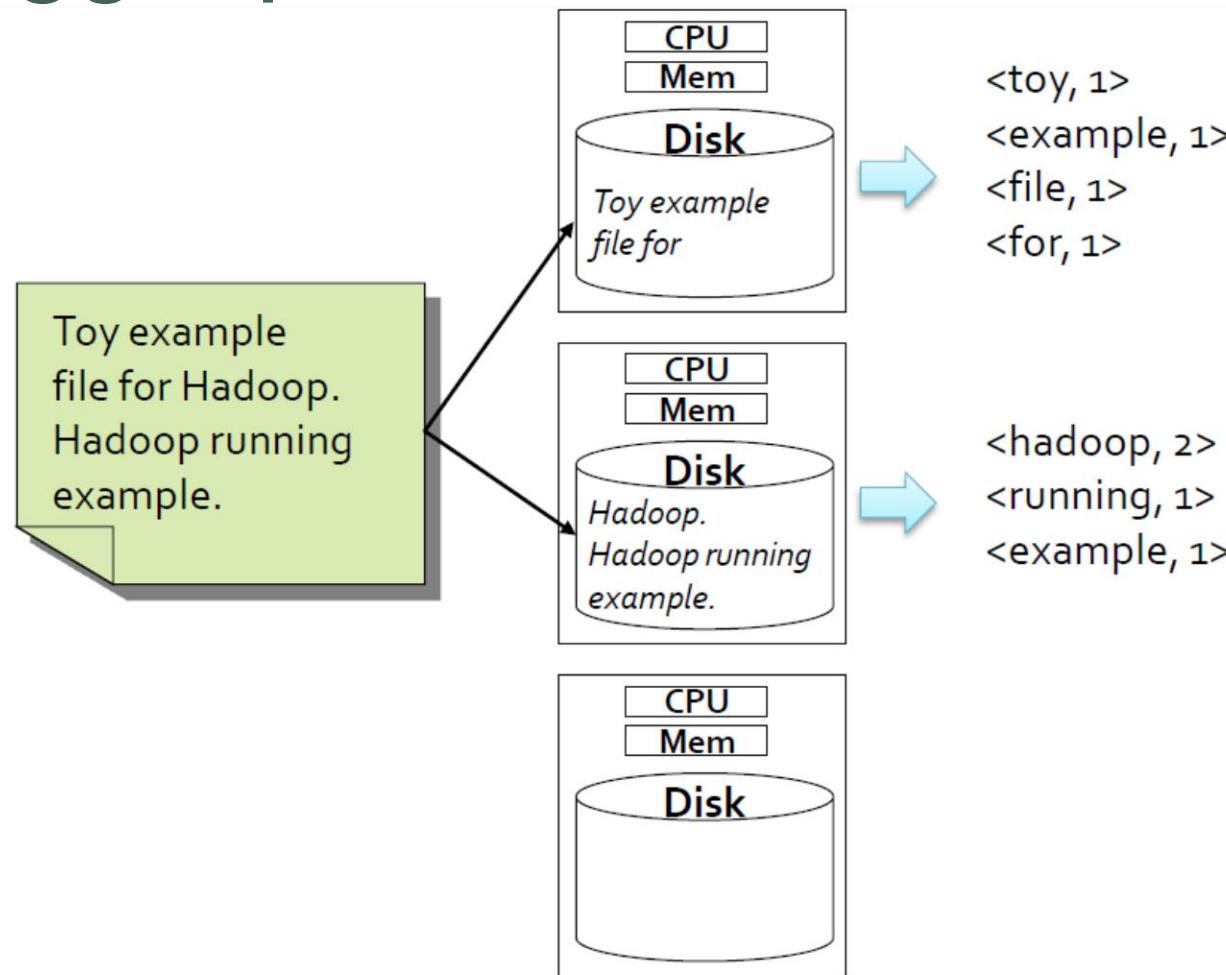


Conteggio parole con un file molto grande

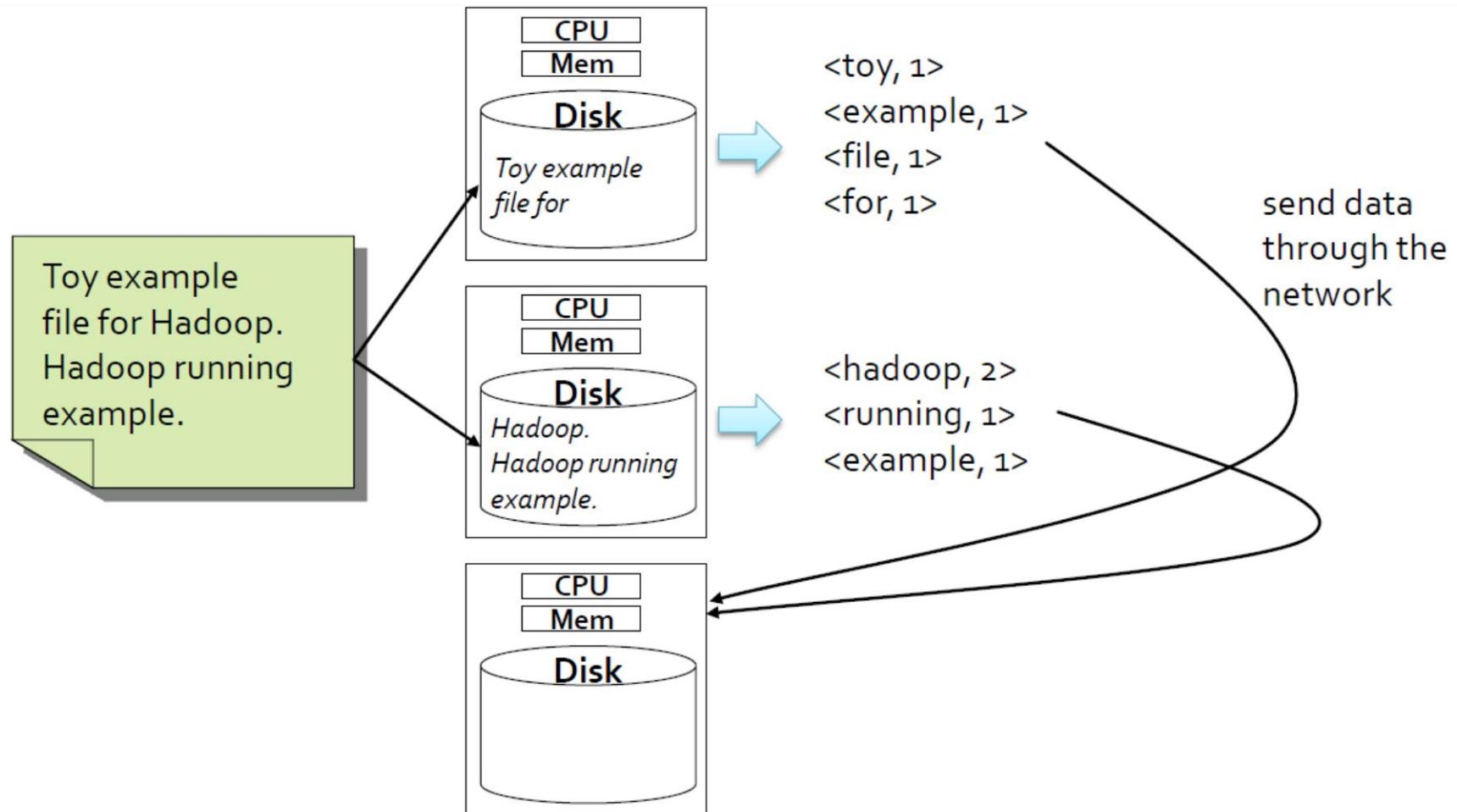
- ÿ Il problema può essere facilmente parallelizzato **1. Ogni server elabora il proprio blocco di dati** e conta il numero di ogni parola appare nel suo blocco
- ÿ Ogni server può eseguire la propria attività secondaria indipendentemente dagli altri server del grappolo
 - ÿ **la sincronizzazione non è necessaria in questa fase**

L'output generato da ciascun chunk da ciascun server rappresenta un **risultato parziale**

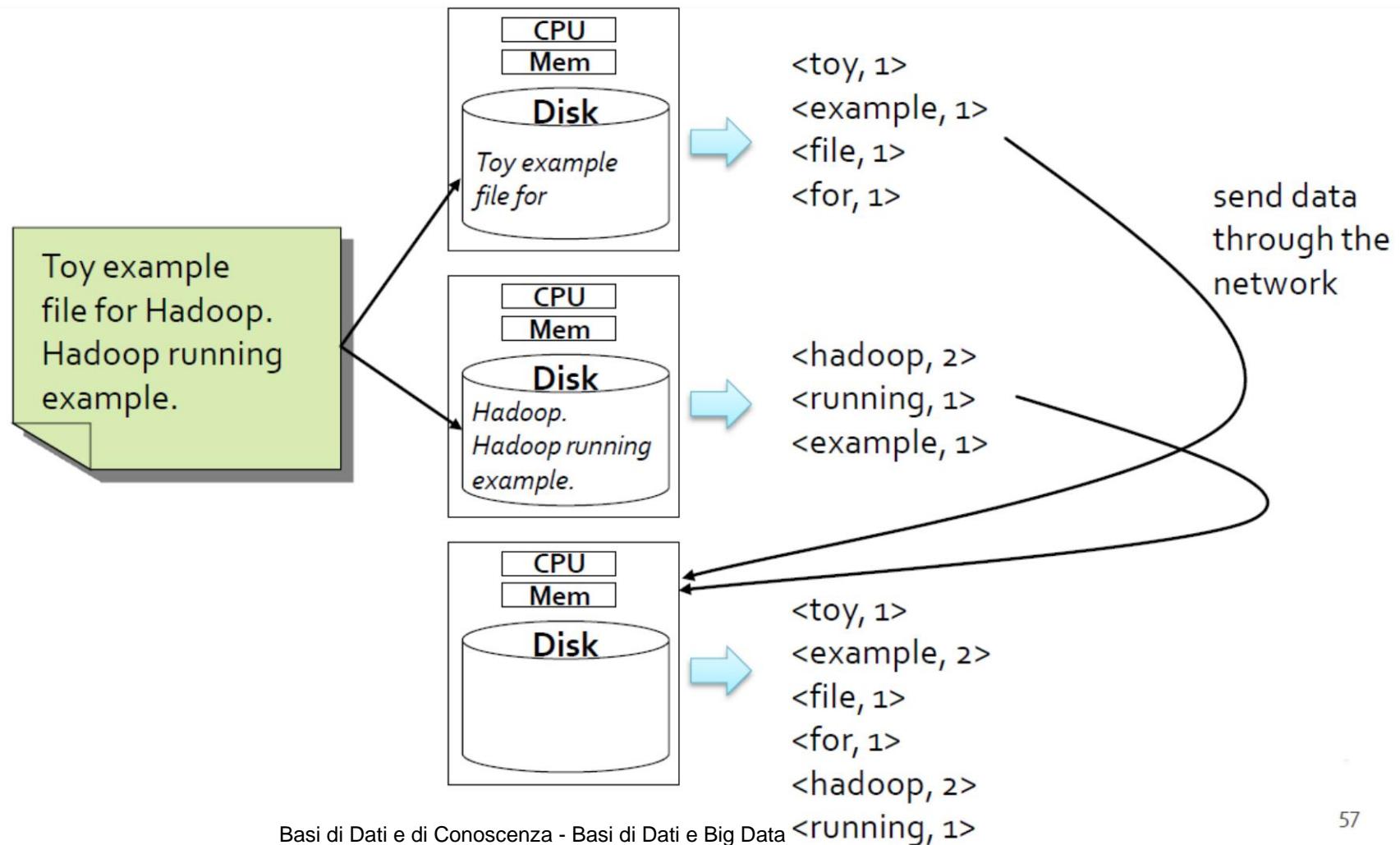
Conteggio parole con un file molto grande



Conteggio parole con un file molto grande



Conteggio parole con un file molto grande



Conteggio parole con un file molto grande

2. Ogni server invia il suo elenco locale (parziale) di coppie **<parola, numero di occorrenze nel suo blocco>** a un server che si occupa di **aggregare** tutti i risultati locali e calcolare il risultato globale
 - ÿ Il server incaricato di calcolare il risultato globale ha bisogno di ricevere tutto il locali Risultati (parziali) per calcolare ed emettere la lista finale
 - ÿ **In questa fase è necessaria un'operazione di sincronizzazione**

Conteggio parole: un esempio più realistico

Caso 2: File troppo grande per entrare nella memoria

principale

Supponiamo che
La dimensione del file sia 100 GB e il numero di parole distinte che ricorrono in esso
massimo

1.000
Il cluster ha 101 server

Il file è distribuito su 100 server e ognuno di questi server contiene una (diversa)
parte del file di input
cioè, il file è
distribuito in modo ottimale su 100 server (ogni server contiene 1/100 del file
nei suoi dischi rigidi locali)

Conteggio parole: complessità

- ÿ **Ogni server legge 1 GB** di dati dal proprio disco rigido locale (legge un blocco da HDFS)
ÿ **Pochi secondi**
- ÿ **Ogni lista locale è composta** da un massimo di 1.000 coppie (perché il numero di parole distinte è 1.000) ÿ **Pochi MB**
- ÿ **La quantità massima di dati inviati** sulla rete è 100 x la dimensione del locale list (numero di server x dimensione dell'elenco locale) ÿ **Alcuni MB**

Conteggio parole: scalabilità

ÿ Possiamo definire **la scalabilità** lungo due dimensioni

ÿ **In termini di dati:**

ÿ Dato il doppio della quantità di dati, l'algoritmo di conteggio delle parole richiede circa n più del doppio del tempo di esecuzione

ÿ Ogni server elabora $2 \times$ dati => $2 \times$ tempo di esecuzione per calcolare l'elenco locale

ÿ **In termini di risorse**

ÿ Dato il doppio del numero di server, l'algoritmo di conteggio delle parole impiega circa non più della metà del tempo di esecuzione

ÿ Ogni server elabora $\frac{1}{2} \times$ dati => $\frac{1}{2} \times$ tempo di esecuzione per calcolare l'elenco locale

Conteggio parole: scalabilità

- ÿ **Il tempo necessario per inviare i risultati locali al nodo incaricato di calcolare il risultato finale e il calcolo del risultato finale sono considerati trascurabili** in questo esempio in esecuzione
- ÿ Spesso questa ipotesi non è vera ÿ
 - Dipende ÿ
 - dalla complessità del problema
 - ÿ dalla capacità dello sviluppatore di limitare la quantità di dati inviati sulla rete

Idee chiave dell'approccio MapReduce

ÿ Ridimensiona "verso l'esterno", non "verso l'alto"

ÿ Aumentare il numero di server, evitando di aggiornare le risorse (CPU, memoria) di quelli attuali

ÿ **Sposta l'elaborazione sui dati** ÿ La rete ha una larghezza di banda limitata

ÿ **Elaborare i dati in sequenza, evitare l'accesso casuale** ÿ

Le operazioni di ricerca sono costose

ÿ Le applicazioni di big data di solito leggono e analizzano tutti i record/oggetti di input
ÿ L'accesso casuale è inutile

Località dei dati

- ÿ I tradizionali sistemi distribuiti (ad es. HPC) spostano i dati su nodi di elaborazione (server)
 - ÿ Questo approccio non può essere utilizzato per elaborare TB di dati
 - ÿ La larghezza di banda della rete è limitata
- ÿ Hadoop sposta il codice nei dati
 - ÿ Il codice (pochi KB) viene copiato ed eseguito sui server dove si trovano i chunk di dati sono memorizzati
 - ÿ Questo approccio si basa sulla “**località dei dati**”

Hadoop e MapReduce

- ÿ Hadoop/MapReduce è progettato per
 - ÿ Elaborazione batch che coinvolge (principalmente) scansioni complete dei dati di input
 - ÿ Applicazioni ad alta intensità di dati
 - ÿ Leggere ed elaborare l'intero Web (ad esempio, il calcolo del PageRank)
 - ÿ Leggere ed elaborare l'intero grafico sociale (ad esempio, LinkPrediction, noto anche come "amico suggerimento")
 - ÿ Analisi dei log (es. Tracce di rete, dati Smart-meter, ..)

Cosa possiamo fare con MapReduce?

ÿ Risolvere problemi complessi è difficile
ÿ Tuttavia, ci sono diversi problemi importanti a cui ci si può adattare
MapReduce

Analisi dei log

ÿ Calcolo del PageRank

Analisi dei grafici sociali

ÿ Analisi dei dati dei

sensori

ÿ Analisi dei dati
delle città intelligenti

ÿ Analisi della cattura della rete

Hadoop e MapReduce

- ÿ Hadoop/MapReduce non è la panacea per tutti i problemi dei Big Data
- ÿ Hadoop/MapReduce non funziona bene ÿ
 - Problemi iterativi
 - Problemi ricorsivi
 - Elaborazione dei dati in streaming
 - ÿ Elaborazione in tempo reale

MapReduce e programmazione funzionale

ÿ Il paradigma di programmazione MapReduce si basa sui concetti di base della programmazione funzionale ÿ

MapReduce “implementa” un sottoinsieme della programmazione

funzionale ÿ Il modello di programmazione appare piuttosto limitato e rigido

ÿ Tutto si basa su due “funzioni” con firme predefinite

MAPP

+

REDUCE

Elementi costitutivi: mappa e riduci

ÿ MapReduce si basa su due “mattoni” principali

ÿ Funzioni **Map** e **Reduce** ÿ

Funzione Map ÿ

Viene **applicata su ciascun elemento** di un set di dati di input ed **emette** un insieme di **copie**

(chiave, valore) ÿ **Funzione Reduce**

ÿ Viene **applicato su ogni insieme di copie (chiave, valore)** (emesse dalla mappa funzione) **con la stessa chiave** ed **emette** un **insieme di copie (chiave, valore)** ÿ
Risultato finale

Esempio in esecuzione di Conteggio parole

ÿ Input ÿ

Un file testuale (cioè un elenco di parole) ÿ **Problema**

ÿ Contare il numero di volte in cui ciascuna parola distinta appare nel file ÿ

Output ÿ Un elenco di coppie <parola, numero di occorrenze nell'input file>

Esempio in esecuzione di Conteggio parole

Il file testuale di input è considerato come un elenco di parole **L**

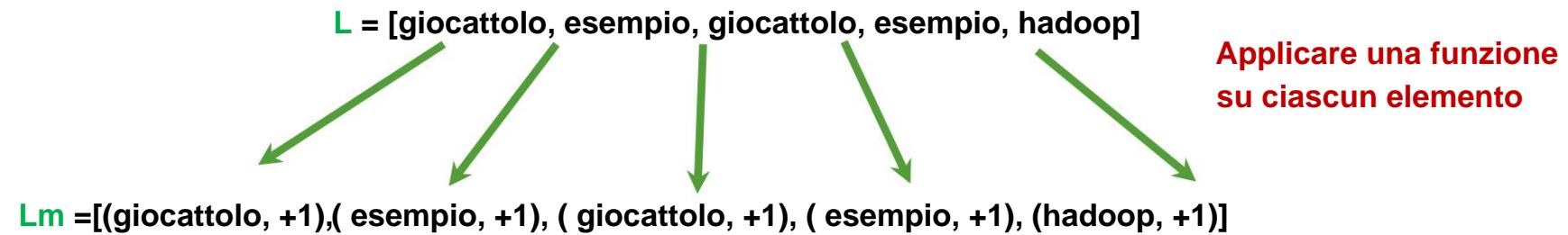
Esempio in esecuzione di Conteggio parole

L = [giocattolo, esempio, giocattolo, esempio, hadoop]

[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

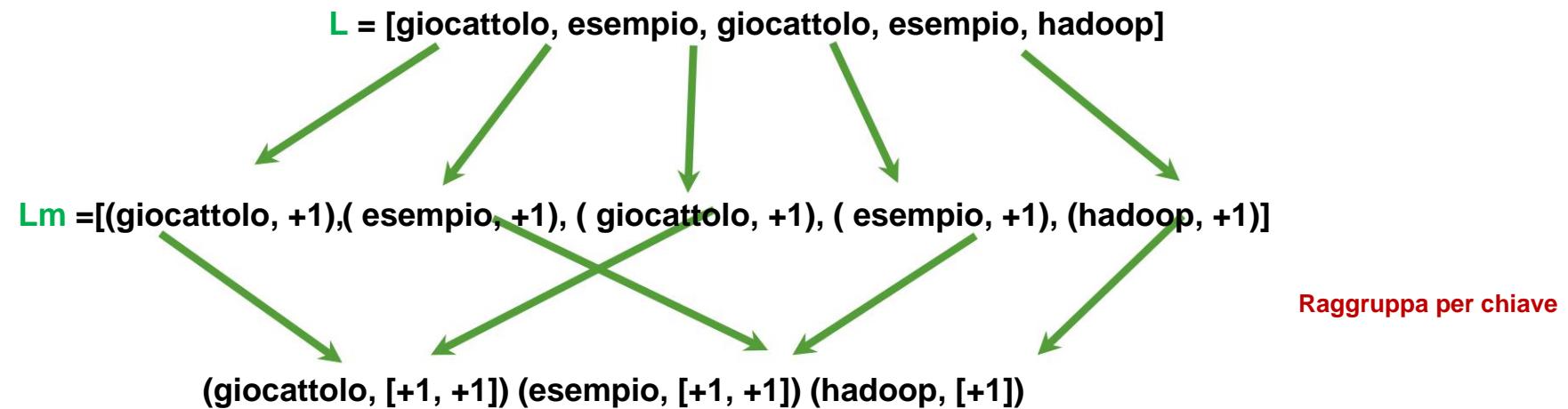
Esempio in esecuzione di Conteggio parole



[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

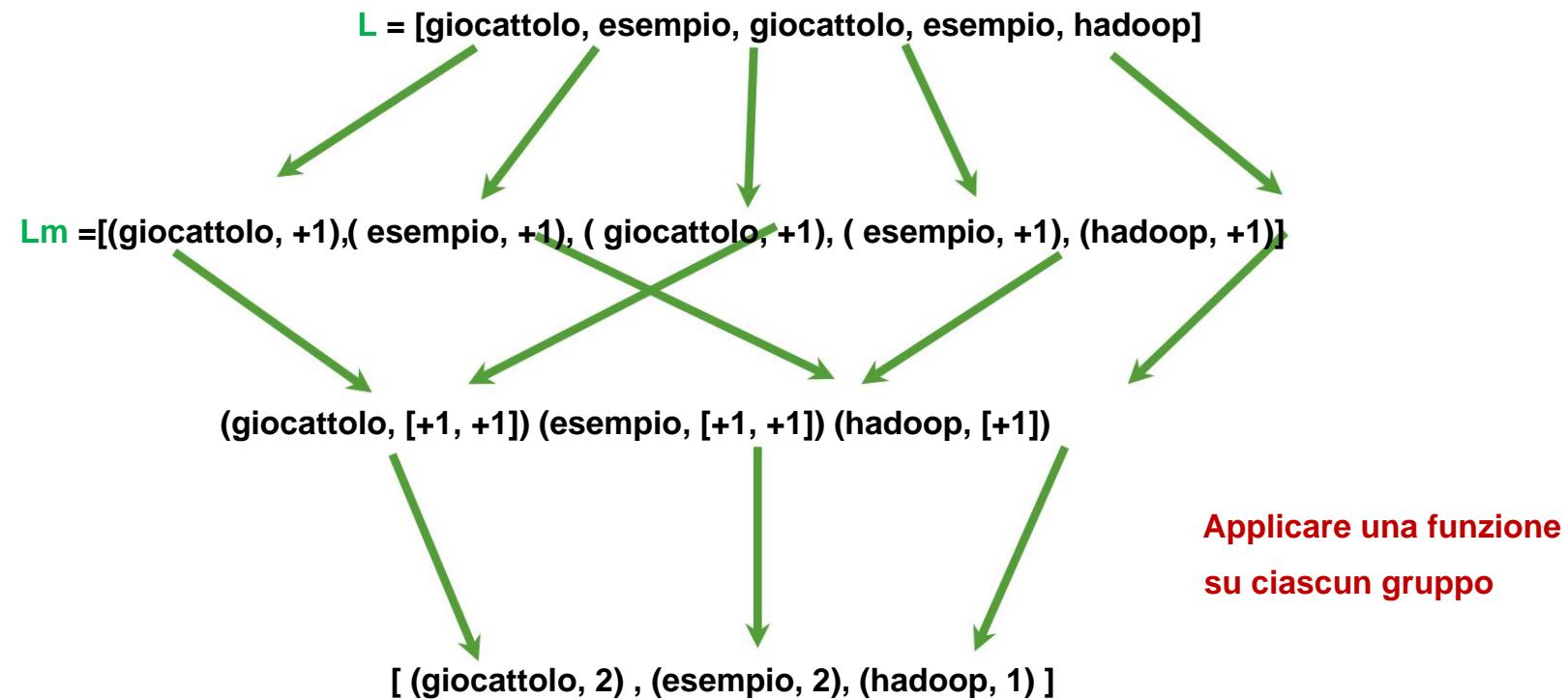
Esempio in esecuzione di Conteggio parole



[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

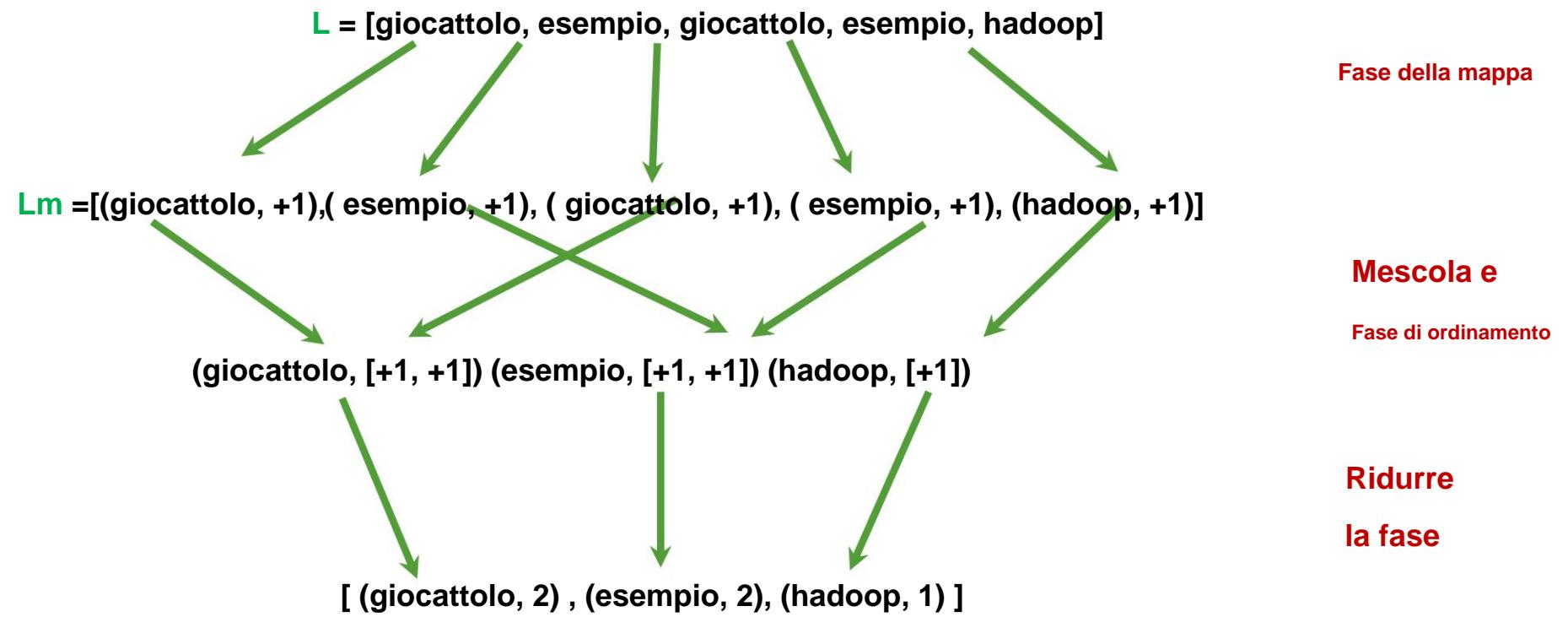
Esempio in esecuzione di Conteggio parole



[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

Esempio in esecuzione di Conteggio parole



[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

Esempio in esecuzione di Conteggio parole

Il file testuale di input è considerato come una lista di parole L . Una coppia chiave-valore $(w, 1)$ viene emessa per ogni parola w in L , cioè la funzione **map** è

$$m(w) = (w, 1)$$

Viene generata una nuova lista di coppie (chiave, valore) L_m

Esempio in esecuzione di Conteggio parole

- ÿ Le coppie chiave-valore in **Lm** sono aggregate per chiave (cioè per parola **w** in our esempio)
 - ÿ Viene generato un gruppo **Gw** per ogni parola **w**
 - ÿ Ogni gruppo **Gw** è una coppia di liste di chiavi (**w**, [lista di valori]) dove [lista di valori] contiene tutti i valori delle coppie associate alla parola **w** ÿ cioè, [lista di valori] è una lista di [1, 1, 1, ...] nel nostro esempio
 - ÿ Dato un gruppo **Gw** il numero di uno [1, 1, 1, ...] è uguale alle occorrenze della parola w in il file di input

Esempio in esecuzione di Conteggio parole

ÿ Per ogni gruppo viene emessa una coppia chiave-valore (**w**, somma **Gw** .[lista di valori])

Gw ÿ cioè, la funzione **di riduzione** è

$$r(\mathbf{Gw}) = (\mathbf{w}, \text{sum}(\mathbf{Gw}.\text{[lista di valori]}))$$

ÿ L'elenco delle coppie emesse è il risultato del problema del conteggio delle parole

ÿ Una coppia (parola w, num. di occorrenze) per ogni parola nella nostra corsa
esempio

MapReduce: Mappa

- ÿ La fase **Mappa** può essere vista come una **trasformazione** su ogni elemento di un set di dati
- ÿ Questa trasformazione è una funzione **m** definita dagli sviluppatori
- ÿ **m** viene invocata una volta per ogni elemento di input
- ÿ Ogni invocazione di **m** avviene **isolatamente**
- ÿ L'applicazione di **m** a ciascun elemento di un insieme di dati può essere parallelizzata in modo semplice

MapReduce: Ridurre

- ÿ La fase **Reduce** può essere vista come un'operazione **di aggregazione**
 - ÿ La funzione di aggregazione è una funzione **r** definita dagli sviluppatori ÿ **r** viene invocata una volta per ogni chiave distinta e aggrega tutti i valori ad essa associati
- ÿ Anche la fase di riduzione può essere eseguita in parallelo e in isolamento
 - ÿ Ogni gruppo di coppie chiave-valore con la stessa chiave può essere elaborato separatamente

MapReduce: Sposta e ordina

- ÿ La fase di shuffle e sort è sempre la stessa ÿ
 - Raggruppa l'output della fase della mappa per chiave
 - ÿ Non ha bisogno di essere definita dagli sviluppatori
 - ÿ È già fornita dal sistema Hadoop

Struttura dati

- ÿ **La coppia chiave-valore è la struttura dati di base in MapReduce**
 - ÿ Chiavi e valori possono essere: numeri interi, float, stringhe, ... ÿ Possono anche essere strutture dati (quasi) arbitrarie definite dal progettista
- ÿ Sia l'input che l'output di un programma MapReduce sono elenchi di valori-chiave coppie
 - ÿ Si noti che anche l'input è un elenco di coppie chiave-valore

Struttura dati

Il design di MapReduce prevede

L'imposizione della struttura chiave-valore sui set di dati di input e output

Ad esempio, per una raccolta di pagine Web, le chiavi di input possono essere URL ei valori possono essere i loro Contenuto HTML

Definizione formale delle funzioni Map e Reduce

↳ Le funzioni map e reduce sono formalmente definite come segue:

map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$

reduce: $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

Poiché l'input il set di dati è un elenco di coppie chiave-valore, l'argomento della funzione map è una coppia chiave-valore

[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

Definizione formale delle funzioni Mappa e Riduci

ÿ Funzione mappa

ÿ **mappa: (k₁, v₁) ÿ [(k₂, v₂)]**

ÿ L'argomento della funzione mappa è una coppia chiave-valore

ÿ Si noti che la funzione mappa emette un elenco di coppie chiave-valore per ogni input

documentazione

ÿ L'elenco può anche essere vuoto

[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

Definizione formale delle funzioni Mappa e Riduci

ÿ Ridurre la funzione

ÿ **ridurre: (k2, [v2]) ÿ [(k3, v3)]**

ÿ Notare che la funzione di riduzione

ÿ Riceve l'elenco completo dei valori [v2] associati a una specifica chiave k2 ÿ

Emette un elenco di coppie chiave-valore

[...] denota un elenco.

(k, v) denota una coppia chiave-valore.

Algoritmi MapReduce

- ÿ In molte applicazioni, la parte fondamentale del set di dati di input viene ignorata
 - ÿ cioè, solitamente la funzione map non considera la chiave del suo valore-chiave argomento coppia
 - ÿ Ad esempio, problema di conteggio delle parole
 - ÿ Alcune applicazioni specifiche sfruttano anche le chiavi dei dati di input
- ÿ Ad esempio, le chiavi possono essere utilizzate per identificare univocamente record/oggetti

Conteggio parole utilizzando MapReduce: Pseudocodice

- ÿ **File di input:** un documento testuale con una parola per riga
- ÿ La funzione map viene invocata su ogni parola del file di input

```
map(key, value): //
key: offset della parola nel file // value: una parola
del documento di input emit(value, 1)
```

```
reduce(chiave, valori): //
chiave: una parola; valori: un elenco di occorrenze di
numeri interi = 0
per ogni c in valori:
    occorrenze = occorrenze + c
emit(chiave, occorrenze)
```