MAPREDUCE

UN MODELLO DI PROGRAMMAZIONE PER I BIG DATA

Problemi Big Data

Map

Iterare su un elevato numero di record

Estrarre qualcosa di interessa da ogni record

Shuffle (mischiare) e sort (ordinare) risultati intermedi

Aggregare risultati intermedi

Reduce

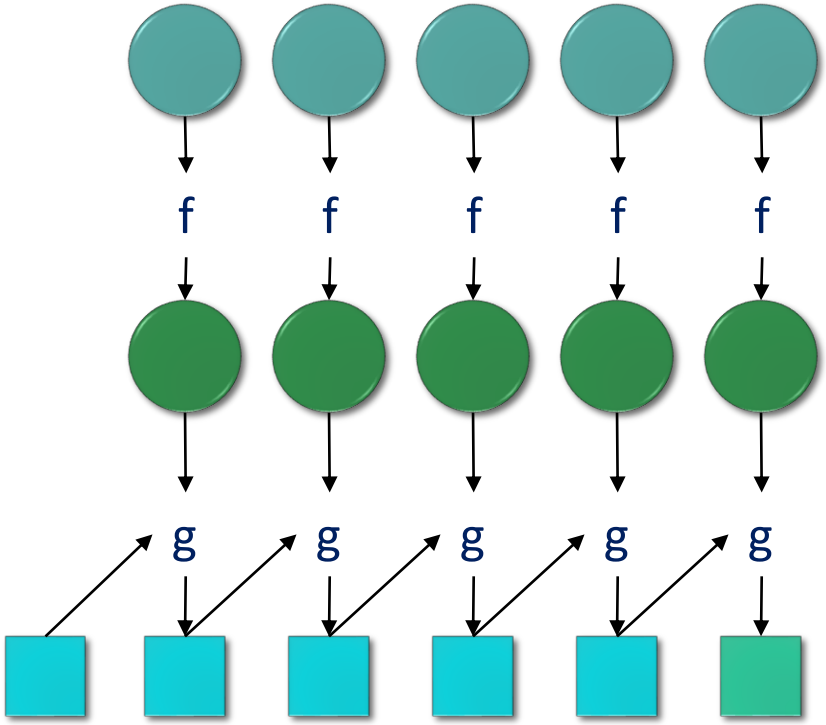
Generare file di output

Idea: incapsulare Map e Reduce in paradigma funzionale

Programmazione Funzionale

MAP

FOLD



MAP applica una funzione f ad ogni elemento in una lista

FOLD applica iterativamente una funzione g per aggregare i risultati

Esempio di programmazione funzionale

Programmazione imperativa

```
a = 0
b = a + 1

### Map example
names = ['Mary', 'Isla', 'Sam']
name_lengths = []
for i in range(len(names)):
  name_lengths[i] = len(names[i])

### Reduce example
sentences = ['Mary read a story to Sam and
Isla.', 'Isla cuddled Sam.', 'Sam
chortled.']
sam_count = 0
for sentence in sentences:
    sam_count += sentence.count('Sam')
```

Programmazione funzionale

```
a = 0
b = increment(a)
def increment(a):
  return a + 1;

### Map example
names = ['Mary', 'Isla', 'Sam']
name_lengths = map(len, names)

### Reduce example
sentences = ['Mary read a story to Sam and
Isla.', 'Isla cuddled Sam.', 'Sam
chortled.']
sam_count = reduce(
  lambda a, x: a + x.count('Sam'),
  sentences, 0)
```

Parallelizzazione MapReduce

map (vale a dire, l'applicazione di *f* a ogni voce in un elenco) può essere parallelizzata in modo semplice, ogni applicazione funzionale avviene in isolamento

- In un cluster, queste operazioni possono essere distribuite in molti computer diversi

reduce ha più restrizioni sulla località dei dati (data locality)

- Gli elementi nell'elenco devono essere "riuniti" prima che la funzione *g* possa essere applicata

Tuttavia, molte applicazioni reali non applicano *g* a tutti gli elementi dell'elenco. Se gli elementi nell'elenco possono essere suddivisi in gruppi, le aggregazioni possono procedere in parallelo.

MapReduce

"MapReduce is a programming model and an associated implementation for processing and generating large data sets.

Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

-- Dean J., Ghemawat S. (Google)

MapReduce di Hadoop è un'implementazione open source del modello di programmazione MapReduce

MapReduce: programma

Un programma MapReduce, denominato job, consiste in:

- Codice per la Map e Reduce
- Parametri di configurazione (directory di input/output sul file System distribuito sottostante)

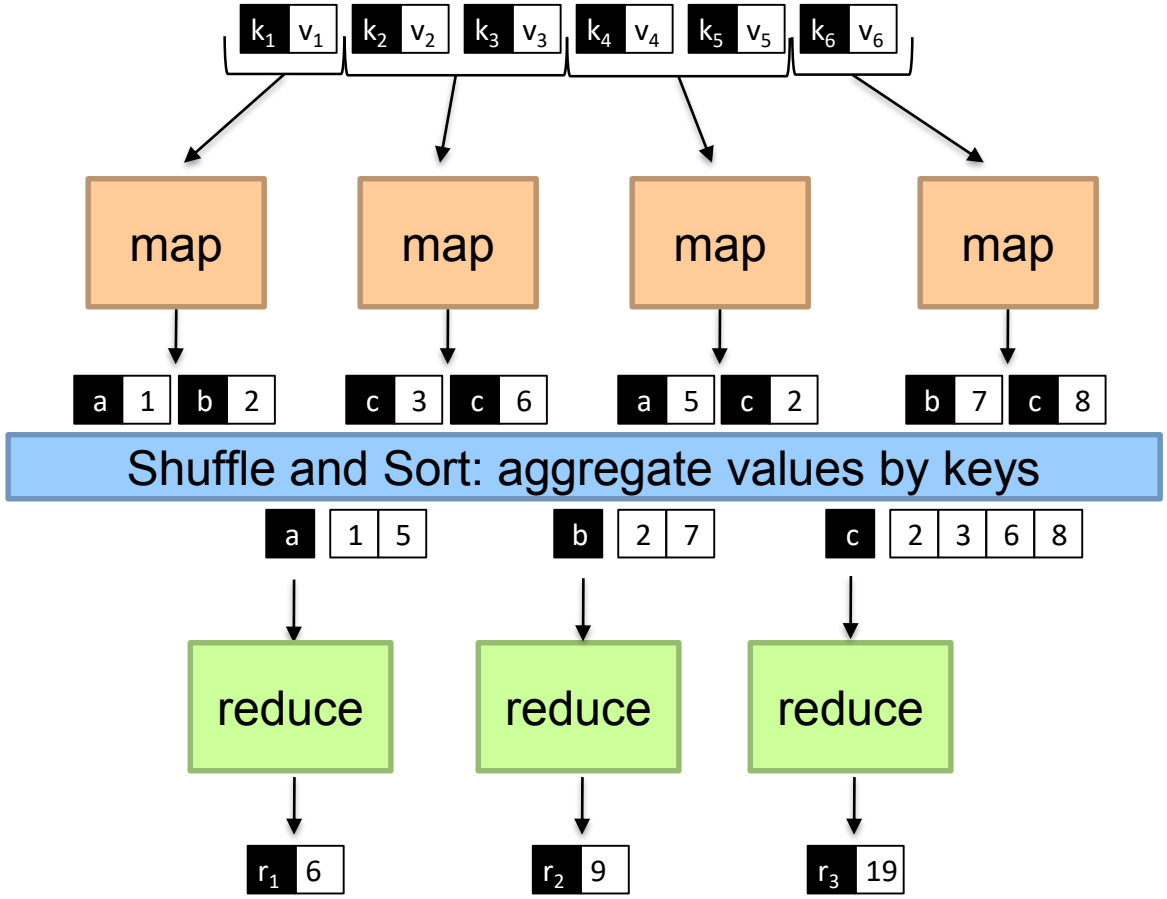
Ogni job MapReduce è diviso dal sistema in unità più piccole chiamate tasks

- Map tasks
- Reduce tasks

Task pianificati utilizzando YARN ed eseguiti sui nodi del cluster

- Se task fallisce, verrà automaticamente riprogrammato per l'esecuzione su un nodo diverso

MapReduce: esempio



Esempio: Word Count

Problema: conteggio del numero di occorrenze per ogni parola in una raccolta di documenti

Input: un repository di documenti; ogni documento è un valore nelle coppie di input

Map function: leggere documento, emettere una sequenza di coppie chiave-valore

- Le chiavi sono le parole che si trovano nei documenti; valori sono uguali a 1:

(w1, 1), (w2, 1), (w1, 1), ... , (wn, 1)

Shuffle and Sort: gruppo per chiave e generare coppie del modulo:

(w1, [1, 1, ... , 1]) , ... , (wn, [1, 1, ... , 1]) ← *Map output*

Reduce function: aggiungere tutti i valori per una determinata chiave ed emette una coppia del modulo:

(wn, l)

Output: *(w, m) coppie; w è una parola che appare almeno una volta tra tutti i documenti di input; m è il numero totale di occorrenze di w tra tutti i documenti*

Ottimizzazione del numero di Reduce task

Hadoop utilizzato per creare un unico Reduce task globale per impostazione predefinita
- Questo sarebbe ovviamente un collo di bottiglia per un lavoro di grandi dimensioni

Parallelismo massimo: un task reduce per nodo e per key
- In realtà, ci sono spesso più chiavi da elaborare che nodi di calcolo disponibili

Una soluzione: un task reduce per nodo, tante chiavi per task reduce
- Ci sono spesso variazioni significative nelle lunghezze degli elenchi di valori per le diverse chiavi e nella quantità di tempo assunta da ogni task reduce
- Se le chiavi sono inviate in modo casuale per i task reduce, possiamo aspettarci che ci sarà un tempo medio diverso richiesto dai task reduce

Una soluzione diversa: tanti task reduce per nodo
- Task reduce possono occupare completamente un nodo di calcolo, brevi task reduce possono essere eseguite in sequenza in un singolo nodo
- Tuttavia, c'è un overhead associato a ogni task che creiamo

Scegliere il numero di reducer per un lavoro è più un'arte che una scienza
- Regola empirica: task reduce deve essere eseguito per 5 minuti (circa) e produrre almeno un valore di output del blocco HDFS

MapReduce: l'immagine completa

I programmatori specificano due funzioni:

map (k1, v1) → list(k2, v2)

reduce (k2, list(v2)) → list(k3, v3)

- Tutti i valori con la stessa chiave vengono ridotti insieme

Di solito, i programmatori specificano anche:

combine (k2, list(v2)) → list(k3, v3)

- Mini-reducer che corrono dopo la fase della Map
- Utilizzato come ottimizzazione per ridurre il traffico di rete

partition (k2, number_of_partitions) → partition_for_k2

- Divide lo spazio chiave per operazioni di riduzione parallele

Il Framework di esecuzione gestisce tutto il resto...

Ambiente MapReduce

MapReduce runtime

Un'idea importante alla base di MapReduce è separare il cosa dal come

Lo sviluppatore avvia il processo sulla JVM del client, che contatta il RM per inviare l'applicazione

Il Framework di esecuzione (il "Runtime") si occupa di tutto il resto

- Gestisce in modo trasparente gli aspetti dell'esecuzione di codice distribuito
- Su cluster che vanno da un singolo nodo a qualche migliaio di nodi

MapReduce "Runtime"

1. Gestisce la pianificazione: assegna i lavoratori per task map e reduce
2. Gestisce la "distribuzione dei dati": ottiene i dati ai lavoratori
3. Sincronizzazione: Raccoglie, Ordina e mescola i dati intermedi
4. Gestisce errori e guasti: Rileva gli errori del container e riavvia
5. Tutto accade in cima a un DFS

Algoritmi MapReduce

Algoritmi per MapReduce

MapReduce è un Framework, non uno strumento

- È necessario adattare la soluzione a funzioni di map e reduce
- In alcune situazioni, potrebbe essere difficile
- Tradurre gli algoritmi di machine learning e data mining nel paradigma MapReduce non è banale
- Si noti che a volte potremmo aver bisogno di più fasi di map/reduce

Algoritmi di filtraggio

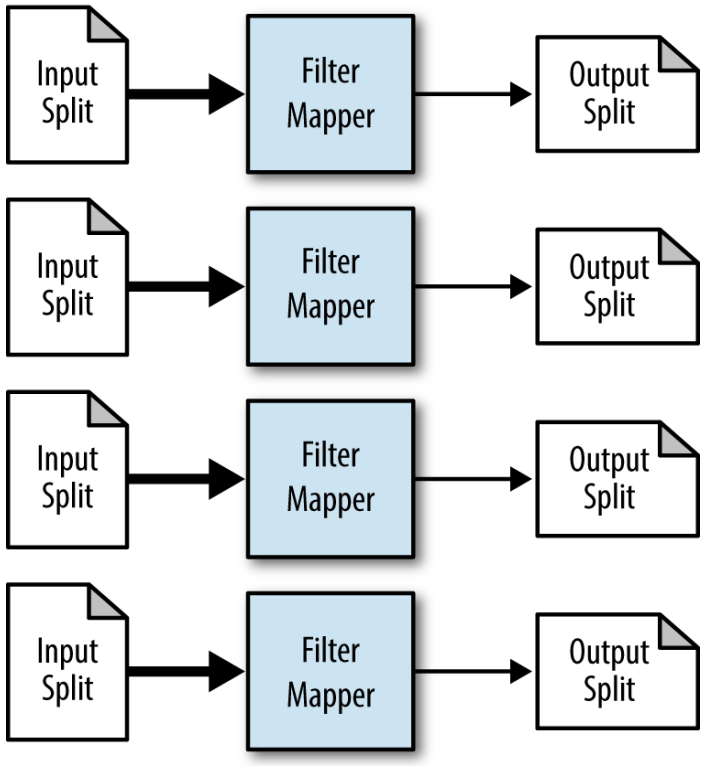
Obiettivo: trovare linee/file/tuple con una particolare caratteristica

- Recupera i log Web per le richieste a un determinato dominio (o da un determinato IP)
- Recuperare un campione del DataSet
- Trova i 10 clienti con la massima reputazione
- Dato che la reputazione è già memorizzata nei dati

General pattern

- map (key, record) → if (critieria is met) then emit(key, record)
- reduce (key, record) → if (critieria is met) then emit(key, record)
  - Reduce può anche essere omesso

Modello di filtraggio



Algoritmi di summarization

Obiettivo: calcolare il massimo/somma/media/... su un insieme di valori

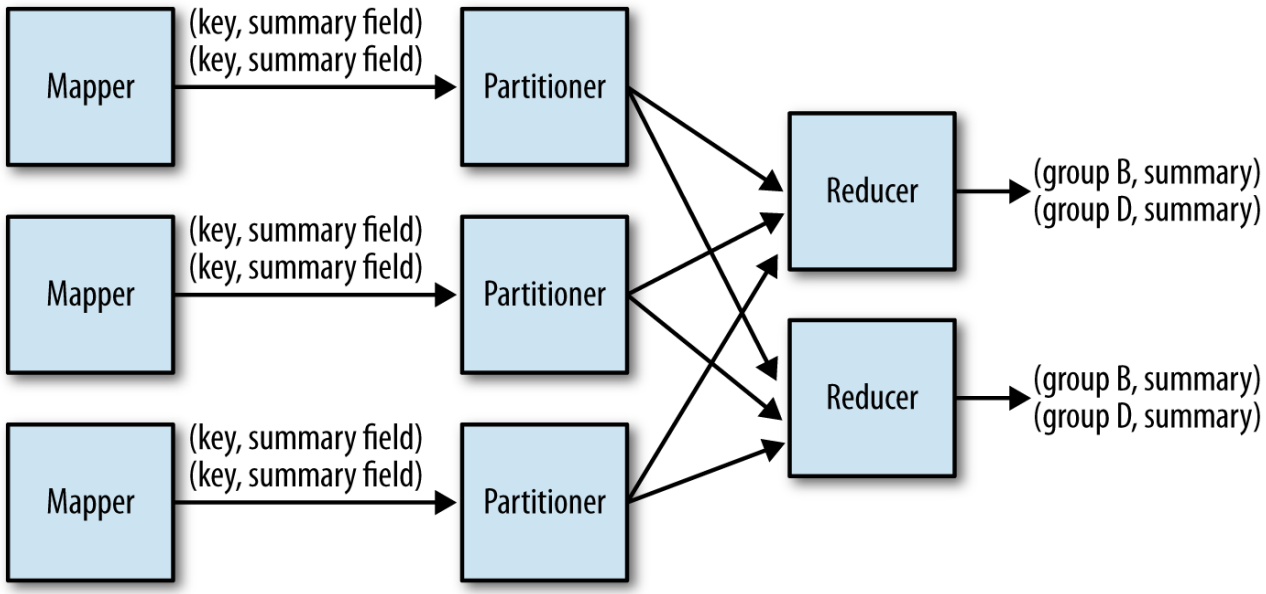
- Contare il numero di richieste per ogni sottodominio di *. csr.unibo.it
- Trova il dominio più popolare (!)
- Costruire un indice invertito di parole nei documenti

Pattern generale:

- map (key, record) → foreach group_criteria in record :
  emit (group_criteria, value)
- reduce (group_criteria, values) → emit (group_criteria, agg(values))

Un combinatore può essere utilizzato per eseguire una pre-aggregazione

Summarization



Join

Obiettivo: combinare diversi input su alcuni valori condivisi

- Dato un elenco di professori (con i corsi che insegnano) e un elenco di studenti (con i corsi che seguono), trovare tutte le triple < professor, corso, studente > dove il corso è in comune

Pattern generale:

- map (key, record) → if (dataset1)
  then emit (sharedKey, (flag1, record))
  else emit (sharedKey, (flag2, record))
- reduce (sharedKey, records) → for r1 in records with flag1 :
  for r2 in records with flag2 :
  emit (someKey, <r1, sharedKey, r2>)

Join

Esempio:

- Prof(Name, Course) = {(Enrico, BigData), (Andrea, BigData)}
- Stud(Name,Course) = {(Mario, BigData), (Lucia, DataMining)}
- Prof JOIN Stud ON (Prof.Course=Stud.Course)

- MAP: {(BigData, (Prof, Enrico)), (BigData, (Prof, Andrea)),
  (BigData, (Stud, Mario)), (DataMining, (Stud, Lucia))}
- SHUFFLE AND SORT: {(BigData, [(Prof, Enrico), (Prof, Andrea), (Stud, Mario)])
  (DataMining, [(Stud, Lucia)])
- REDUCE: {(k1, (Enrico, BigData, Mario)), (k2, (Andrea, BigData, Mario))}

Sort

Obiettivo: ordinare l'input

- Restituire tutti i domini indicizzati da Google e il numero di pagine in ciascuno, ordinati per il numero di pagine

Il modello di programmazione non supporta questo per sé

- Ma le implementazioni fanno: la fase shuffle esegue il raggruppamento e l'ordinamento!

Pattern generale:

- map (key, record) → emit (sortKey, record)
- reduce (sortKey, records) → emit (sortKey, records[1]), ...

Map e Reduce non fanno nulla

- Con 1 reducer, otteniamo l'uscita ordinata
- Con molti reducer, otteniamo parzialmente ordinato output (a meno che: TotalOrderPartitioner)

Due fasi di MapReduce

Dato che i calcoli di MapReduce sono più complessi, è utile suddividere in fasi

- L'output della prima fase funge da input per il successivo
- La stessa uscita può essere utile per diverse fasi successive
- L'output può essere memorizzato nel DFS, formando una vista materializzata

Le prime fasi delle operazioni di MapReduce spesso rappresentano la quantità più pesante di accesso ai dati, in modo da costruire e salvarli una volta come base per molti usi downstream fa risparmiare un sacco di lavoro!

Due fasi di MapReduce

Esempio: confrontare le vendite di prodotti per ogni mese in 2011 a 2010

Prima fase: produrre record che mostrano le vendite per un singolo prodotto in un solo mese di un singolo anno

- map (key, sales) → for s in sales :

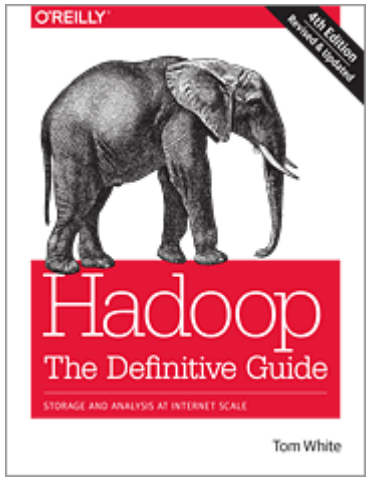
  emit(concat(s.month, s.year, s.product), s.qty)
- reduce (keyMYP, quantities) → emit (keyMYP, sum(quantities))

Seconda fase: produrre record che mostrano l'aumento delle vendite per un singolo prodotto in un singolo mese confrontando vendite 2011 con quelle 2010

- map (keyMYP, totalQty) →

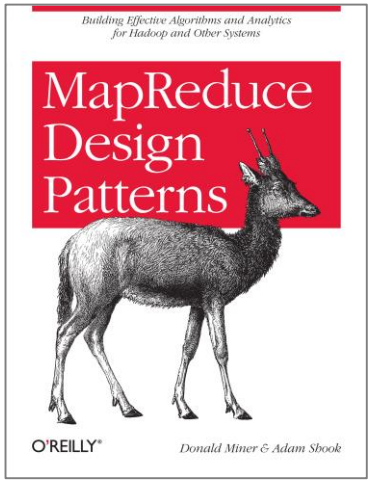
  emit(concat(s.month, s.product),(s.year, s.totalQty))
- reduce (keyMP, totals) → emit (keyMP, (totals[2011]/totals[2010]))

Letture e risorse suggerite



T. White
Hadoop The Definitive Guide
4th edition (O'Reilly)

D. Miner, A. Shook
MapReduce Design Patterns
O'Reilly



Hadoop official documentation
https://hadoop.apache.org/docs/stable/hadoop-project-dist