

Elaborazione dati parallela con map/reduce

Roberto Congiu
rcongiu@yahoo.com

Indice delle slide

- Introduzione a Map/Reduce
- Descrizione del modello
- Implementazione
- Ottimizzazioni

Introduzione

- Map/Reduce e' un *modello di programmazione e la sua implementazione* per elaborare grandi quantita' di dati
- Le applicazioni scritte usando map reduce possono essere eseguite in cluster di macchine economiche con una curva di apprendimento brevissima

Tipo di applicazioni

- Web Indexing
- Reverse Web-Link graph
- Distributed sort
- URL access counter

Vantaggi

- Elaborazione distribuita su un cluster 'commodity'
- Scalabile quasi linearmente
- Paradigma facile da imparare per il programmatore

Modello

- Input: insieme di coppie chiave/valore
- Output: insieme di coppie chiave/valore
- Elaborazione in due stadi: Map e Reduce
- Map produce un insieme intermedio di coppie chiave/valore
- Map e Reduce sono scritte dallo sviluppatore

Esempio (1)

- **Contare il numero di occorrenze di una parola**

```
Map(String key, String value):  
  // key: document name  
  // value: document contents  
  for each word w in value:  
    EmitIntermediate(w, "1" );
```

Esempio (2)

```
Reduce (String key, Iterator values):
```

```
    // key: a word
```

```
    // values: a list of counts
```

```
    int result=0
```

```
    for each v in values:
```

```
        result += ParseInt(v)
```

```
    Emit(AsString(result))
```

Esempio (3)

- Map emette le parole con un semplice “1” associato
- Reduce somma tutti gli “1” prodotti per ottenere il totale
- Il programmatore indica anche i nomi dei file di ingresso e di uscita e i parametri (opzionali) di ottimizzazione.
- Il codice e' linkato con la libreria

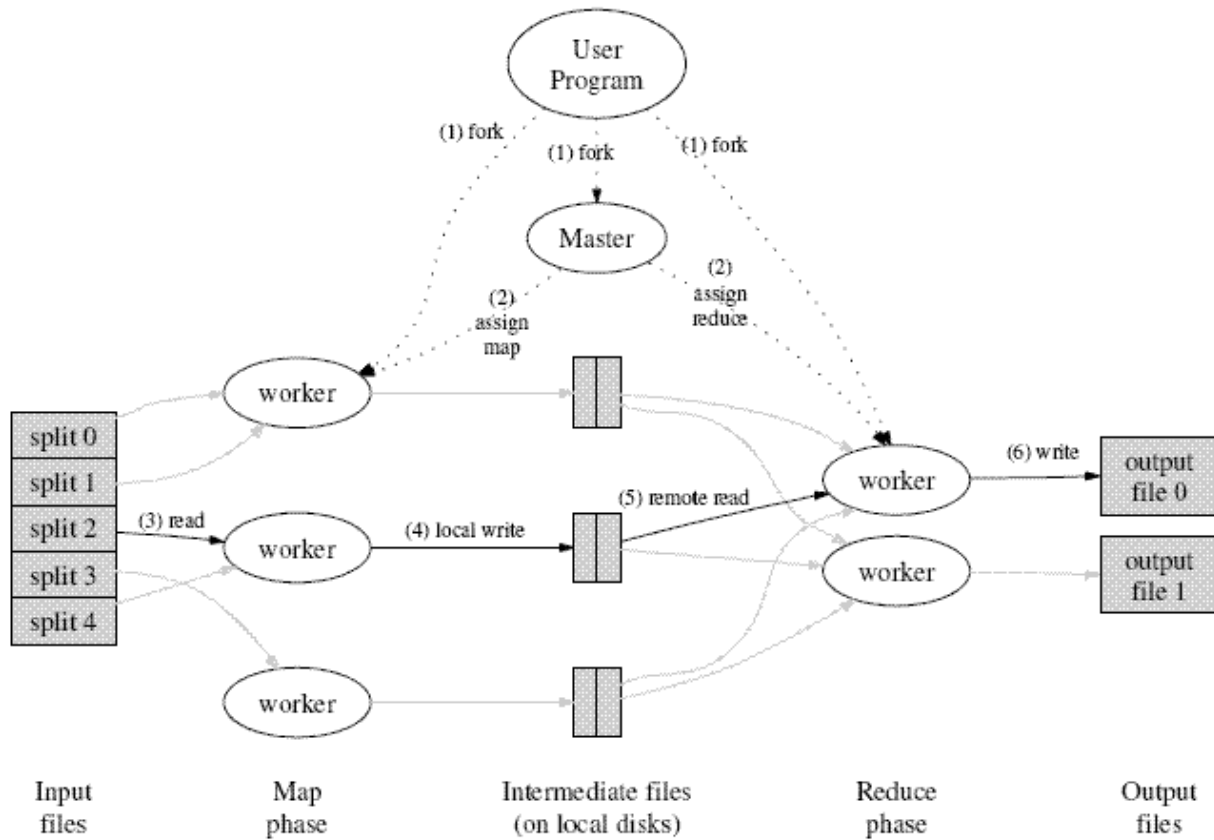
Types

- Map: $(k1, v1) \rightarrow \text{list}(k2, v2)$
- Reduce: $(k2, \text{list}(v2)) \rightarrow \text{list}(v2)$

Implementazione

- Diverse implementazioni sono possibili a seconda dell'hardware disponibile
- Ci focalizzeremo su cluster di macchine di basse prestazioni (single/dual proc x86 con 2G di RAM e disco IDE) piuttosto che su supercomputer

Implementation schema



Partizionamento

- I dati in ingresso vengono divisi in M partizioni
- Le M partizioni verranno elaborate in parallelo dai mapper su macchine diverse
- L'operazione di reduce è parallelizzata partizionando lo spazio delle chiavi intermedie in R partizioni ($\text{hash}(\text{key}) \bmod R$)

Partizionamento

- Il numero di partizioni M e' in genere scelto tale che ogni partizione sia dai 16 ai 64 Mb (controllabile come parametro)
- Il numero di partizioni R e la funzione di partizionamento sono specificate dall'utente.

Master e Workers

- Su ogni macchina del cluster gira uno o più processi controllati dalla libreria
- Un processo è il master, che assegna ai workers un'operazione di map o di reduce, passandogli la partizione da elaborare
- Un worker che elabora un map legge la partizione in ingresso e emette chiave/valore in M partizioni usando la funzione di hash specificata dall'utente.

Master e workers

- Il map worker bufferizza su disco il risultato. Quando termina, notifica il master con la locazione dei file prodotti (un numero $\leq M$)
- Quando tutti i map sono terminati, il master assegna le operazioni di reduce passando la lista di n file per una delle R partizioni ($n \leq M$).

Reduce

- Il reduce worker ordina tutti i file in ingresso per chiave
- Per ogni chiave incontrata, passa tutti i valori corrispondenti alla funzione scritta dall'utente
- Il reduce worker quindi scrive in un file system globale i risultati emessi per quella partizione

Output

- L'output e' partizionato in R file
- Non e' necessario ricombinarli se sono a loro volta l'input di un'altro processo di map/reduce

Master

- Per ogni operazione di map o reduce, mantiene lo stato (idle, in-progress, completed)
- Propaga le locazioni dei file di input, intermedi, e di output

Fault tolerance

- Il master 'pinga' periodicamente i worker
- Se il worker non risponde entro un certo tempo, viene considerato fallito e quindi re-schedulato (impostato a idle)
- Le operazioni di map salvano il loro output su dischi locali e quindi vengono reschedulate anche se complete se la macchina sulla quale sono state eseguite ha un guasto

Fault tolerance

- Le operazioni di reduce salvano il loro risultato su un disco globale quindi non hanno bisogno di essere re-eseguite.
- Il fallimento del master e' improbabile, ma puo' periodicamente salvare su disco lo stato ed essere rieseguito.

Operazioni di backup

- In un cluster, alcune macchine possono essere molto piu' lente di altre (problemi di rete, di carico eccessivo): il task di map o reduce puo' quindi essere lento e ritardare il completamento dell'elaborazione
- Quando l'elaborazione e' vicina al completamento, tutte le operazioni in progress vengono rischedulate. Il primo worker che completa viene preso, l'altro scartato
- Questo meccanismo generalmente migliora le prestazioni di circa il 40%

Combiner

- Nell'esempio del conteggio delle parole, si potrebbe calcolare un sottotale per ogni mapper per rendere più piccoli i file in uscita
- Ottimizzazione: il combiner calcola il sottotale usando la stessa funzione usata per il reduce
- L'operazione di reduce deve essere associativa e commutativa.

Formato di input e output

- Qualsiasi formato che possa essere trattato come chiave/valore puo' essere utilizzato
- Nell'implementazione, si puo' astrarre in un reader/writer generico

Errori di elaborazione

- Se un file provoca un errore in maniera deterministica (per esempio un segfault) in un worker, l'elaborazione non verra' mai portata a termine
- Si puo' implementare un meccanismo che marca i record problematici e permette di saltarli, loggarli e completare comunque l'elaborazione

Bibliografia

- **MapReduce: Simplified Data Processing on Large Clusters (J.Dean e Sanjay Ghemewat)**