Twister: Runtime for Iterative MapReduce

Jaliya Ekanayake, Hui Li, Bingjing Zang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, Geoffrey Fox

Valerio Massimi

Sapienza Università di Roma

4 giugno 2012

Agenda

Motivazioni

1. Incremento della quantità di dati in tutti i campi (Peta-Scale)
2. Incremento della capacità computazionale per risolvere problemi reali
3. Anche usando sistemi paralleli la capacità computazionale non basta
4. MapReduce come soluzione per sistemi distribuiti:
   1. Facilità di apprendimento
   2. Facilità di utilizzo
   3. Buon supporto

Caratteristiche di MapReduce

1. MapReduce segue un approccio incentrato sui dati
2. Quando il volume dei dati è grande è necessario un infrastruttura di comunicazione piu semplice possibile per ottenere buone performance
3. MapReduce ha inoltre un modello di sincronizzazione rilassato
4. Supportare esecuzioni di MapReduce iterativo può ampliare il suo range di applicazione

Architettura MapReduce

- Le scelte architetturali sono basate sulla scala dei problemi e sulla dimensione delle infrastrutture usate
- La maggior parte delle implementazioni di MapReduce rispettano questa architettura
- Le implementazioni di MapReduce sono focalizzate sull' esecuzione di un singolo passo

Twister

Twister is an enhanced MapReduce runtime with an extended programming model that support iterative MapReduce computations efficiently

Twister tramite alcuni miglioramenti e ottimizzazioni riesce a gestire computazioni iterative di MapReduce più efficacemente delle altre implementazioni quali Hadoop e DriadLINQ

MapReduce Handling Data

Input/Output data handling:

1. Viene usato un file system distribuito resistente ai fault
2. Il D.F.S. è implementato sui dischi locali dei nodi
3. Utilizza il metodo della duplicazione dei dati per resistere ai fault

Intermediate data handling:

1. Dopo un passo di map i risultati vengono memorizzati nei dischi locali
2. Il **master scheduler** assegna questi dati ai reducers
3. I worker cercano e ricevono i dati tramite protocolli di comunicazione (HTTP)
4. Questa gestione semplifica il recupero dai fault ma introduce un alto overhead

MapReduce Scheduling

Dynamic scheduling mechanism :

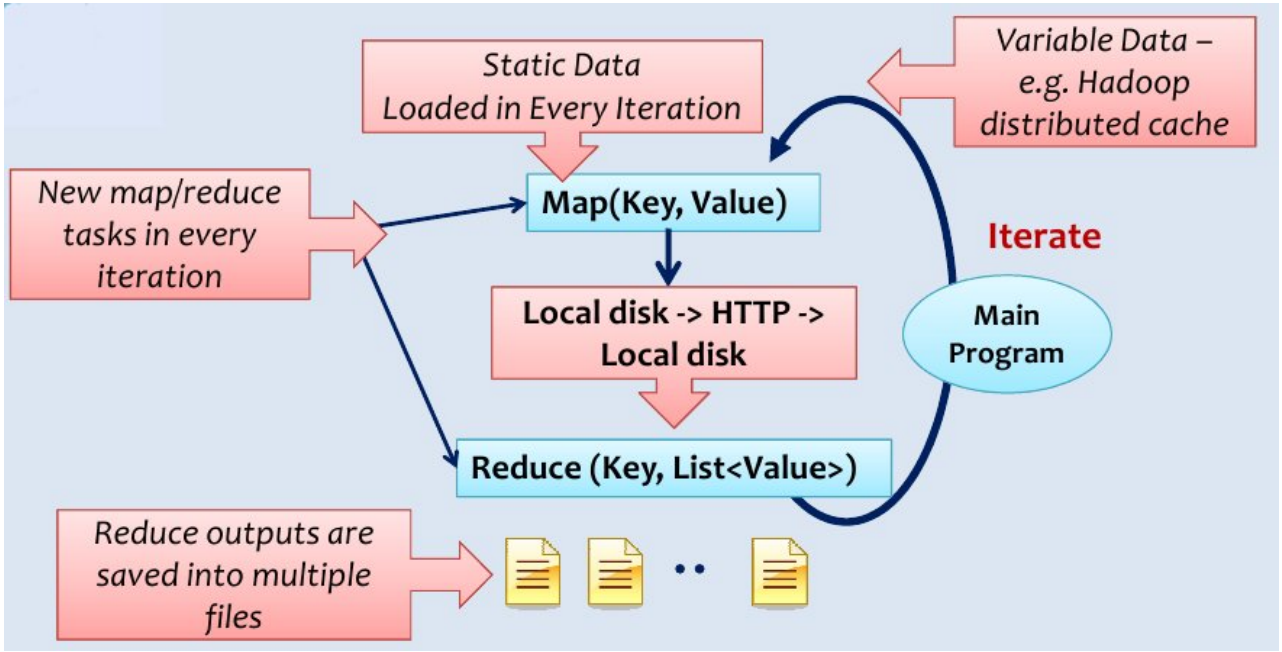
1. Assegna i vari task alle risorse disponibili
2. Questo approccio facilita l'ottimizzazione dell' utilizzo di sistemi eterogenei
3. Assegnamento iniziale dei map task viene effettuato sulla base della località dei dati

MapReduce fault handling

Faillure:

1. Fallimento di un Nodo
   1. Scrivere tutti i dati prodotti su storage semplifica la gestione dei faillure
   2. Con il meccanismo della replicazione dei dati si possono recuperare i dati in caso di fallimento di un disco
2. Fallimento Tasks
   1. Map: gestito semplicemente riavviando il task
   2. Reduce: Si riesegue il task dopo aver ricaricato il risultato appropriato
3. Si assume che il master node non possa fallire

Iterazione con MapReduce



- Focalizzato principalmente su un passo di MapReduce
- Introducono molti overhead:
  - Reinizializzare i task
  - Ricaricare i dati statici
  - Comunicazioni e Trasferimenti ottimizzabili

Map reduce iterativo con Twister

Si osserva che molti algoritmi paralleli hanno una semplice struttura iterativa:

1. Usano dati statici e dinamici
2. Hanno bisogno di più iterazioni per arrivare alla convergenza
3. Decidono se continuare o no l'iterazione dopo il passo di reduce

Per supportare questi presupposti viene implementato Twister

Twister Definition

Twister

Twister is a distributed MapReduce runtime optimized for iterative MapReduce computations. It reads data from local disks of the worker nodes and handles the intermediate data in distributed memory of the worker nodes. All communication and data transfers are performed via a publish/subscribe messaging infrastructure.

Static Data vs. Variable data

Static Data:

- Rimangono immutati per tutta la durata della computazione
- Sono usati in tutte le iterazioni
- Esempio: Pagerank graph

Variabile Data:

- Sono il valore computato
- Vengono consumati dalla prossima iterazione
- Esempio: Pagerank score

Long MapReduce tasks

1. Molte applicazioni iterative hanno una quantità di dati che entra nella memoria distribuita (cachable)
2. Utilizzando Long-running MapReduce tasks diminuiamo la necessità di caricare nuovi dati
3. E' comunque possibile utilizzare Twister anche con applicazioni con dati che eccedono la capacità della memoria distribuita

Granularity of tasks

1. Il documento di Google MapReduce presenta una granularità dei compiti molto fine
2. Incrementare la granularità dei map tasks riduce il volume dei dati intermedi
3. In Twister si deve dare un maggiore peso computazionale alla fase di map
4. Con la possibilità di configurare i map task questi possono accedere ad una quantità di dati più grande

Combine operation

- In Google MapReduce l'output dei reducer è memorizzato in file separati.
- Molte applicazioni iterative hanno bisogno degli output combinati per decidere se continuare ad iterare
- Twister introduce una nuova fase di combine che aggrega i dati in un unico file

Twister API

```
1.configureMaps(PartitionFile partitionFile)
2.configureMaps(Value[] values)
3.configureReduce(Value[] values)
4.runMapReduce()
5.runMapReduce(KeyValue[] keyValues)
6.runMapReduceBCast(Value value)
7.map(MapOutputCollector collector, Key key, Value val)
8.reduce(ReduceOutputCollector collector, Key key,List<Value>
  values)
9.combine(Map<Key, Value> keyValues)
```

Programming extension

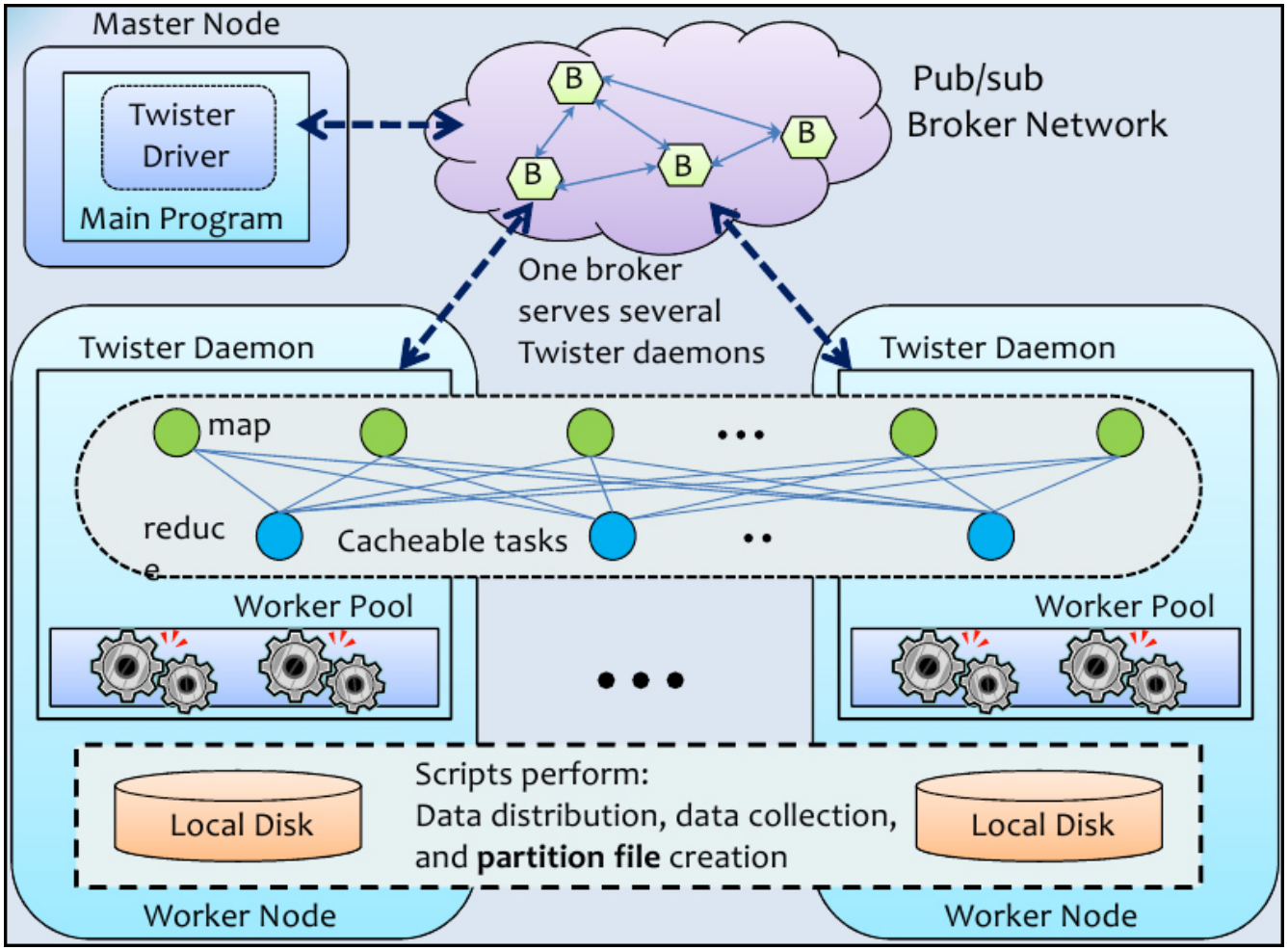
- MapReduceBcast(Value value)
  1. Invia un *Value* a tutti i mappers
  2. *Value* è un tipo di dato generico
- ConfigureMap(PartitionFile partitionfile)
- ConfigureMap(Value[] value)
  1. Permettono la configurazione dei passi di Map e Reduce
  2. Possono essere inizializzati tramite partiton file o tramite valori
- ConfigureReduce(Value[] value)

Twister Architecture

L'architettura di Twister è formata da 3 entità principali

1. I driver del lato Client (Twister driver)
2. I Twister Daemon eseguiti su ogni worker node
3. La broker network

Twister Architecture



Client Side drivers

- Coordinano l'intera computazione
- Provvedono a tradurre i comandi dati dall'utente tramite le API in messaggi ed eventi da inviare ai daemon
- Si assume che questi non possano fallire

Twister Daemons

- Durante l'inizializzazione Twister avvia un demone per ogni working node
- Il Twister daemon si occupa di:
  - Aprire una connessione con la broker network
  - Gestire i passi si MapReduce assegnatogli
  - Mantenere il Worker pool
  - Notificare il suo stato al master node
  - Rispondere agli eventi di controllo

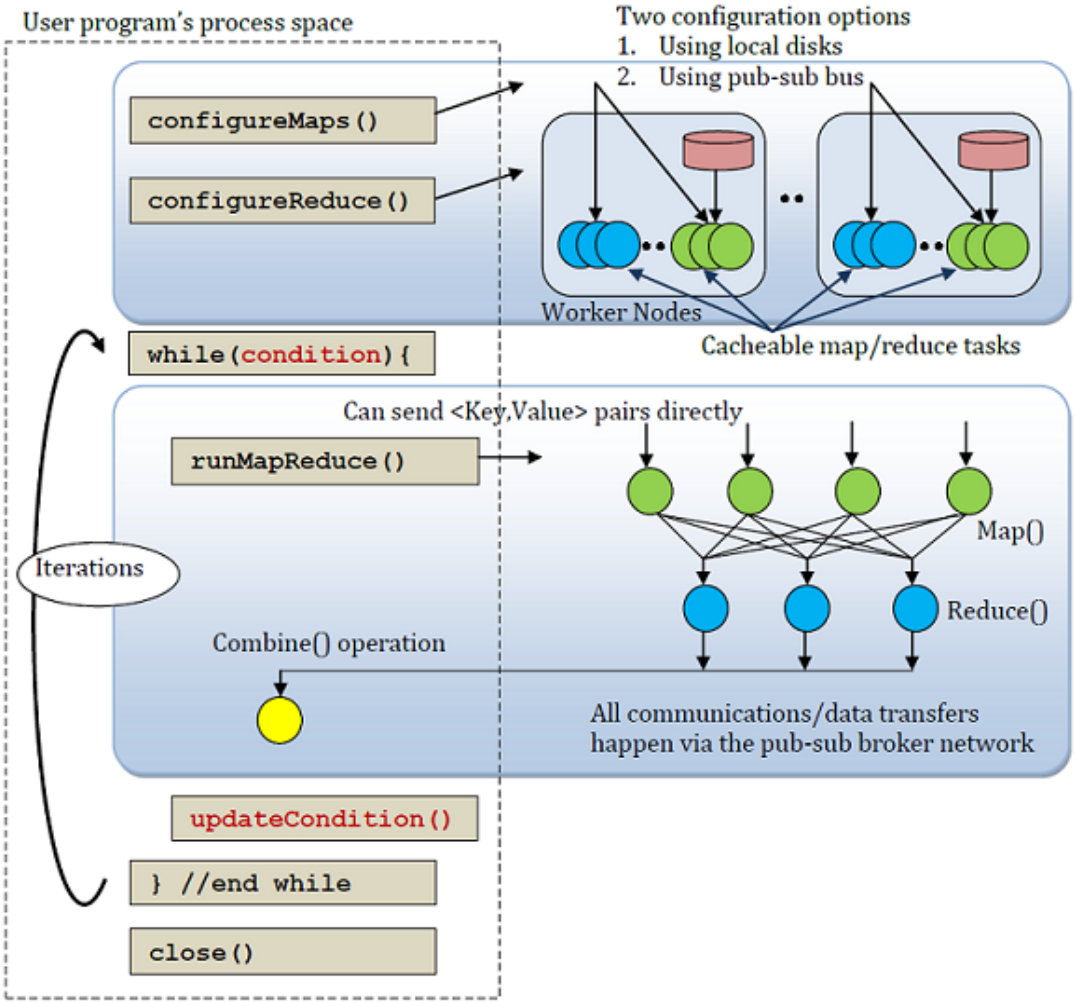
Twister Broker network

L'infrastruttura di comunicazione usata si basa sul paradigma publish/subscribe per gestire 4 tipi di comunicazione:

1. Inviare e ricevere eventi di controllo
2. Inviare dati dal client ai Twister daemons
3. Trasferimento dei dati intermedi tra i mapper e i reducer
4. Rispedire l'output dei reduce task al master node per l'operazione di combine

Attualmente sono supportate NaradaBrokering e ActiveMQ come infrastructure di message passing

Twister Programming model



Handling Input e output

Mette a disposizione 2 meccanismi per accedere ai dati di input:

- Leggere i dati direttamente dai dischi locali dei nodi
  - Permette di iniziare la computazione con un grande dataset
  - Si assume che i dati letti sono divisi in file
  - Potrebbe essere necessario dividere dei grandi file in file più piccoli
- Ricevere i dati direttamente dalla broker network
  - Metodo non efficiente per trasferire grandi quantità di dati input
  - Utile per inviare dati variabili ai vari map task

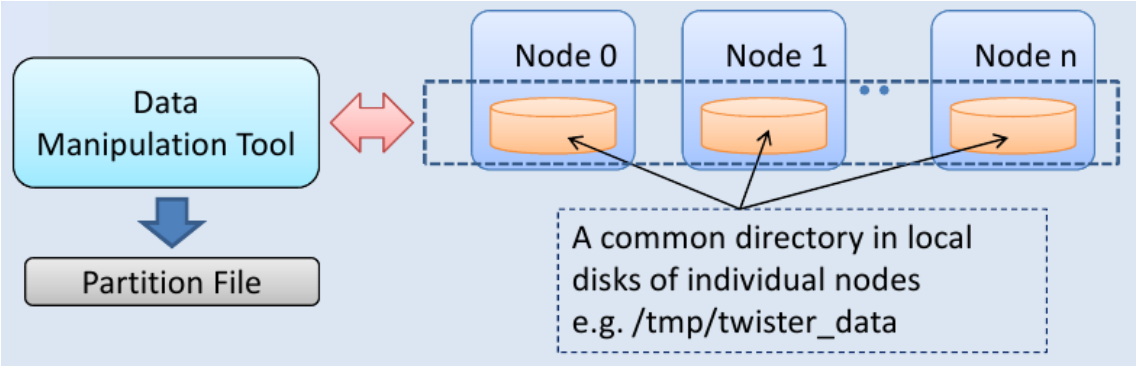
Partition File

File No	Node IP	Daemon No	File partition path
4	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-23.bin
5	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-0.bin
6	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-27.bin
7	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-20.bin
8	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-23.bin
9	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-25.bin
10	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-18.bin
11	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-15.bin

- I dati riguardo la distribuzione dei file di input vengono memorizzati dal **partiton file**
- Il **partiton file** è formato da una lista di tuple
- Questo meccanismo non assicura le stesse servizi di un HDSF

Il meccanismo implementato in Twister cerca di ispirarsi al concetto di **move computation to data**

Data Manipulation tool



- Sono previste le funzionalità base per gestire i dati tra i dischi locali
- I dati sono partizionati in file (In Hadoop sono partizionati in blocchi di grandezza fissa)
- Prevede la duplicazione dei file

Handling Intermediate Data

Assunzione

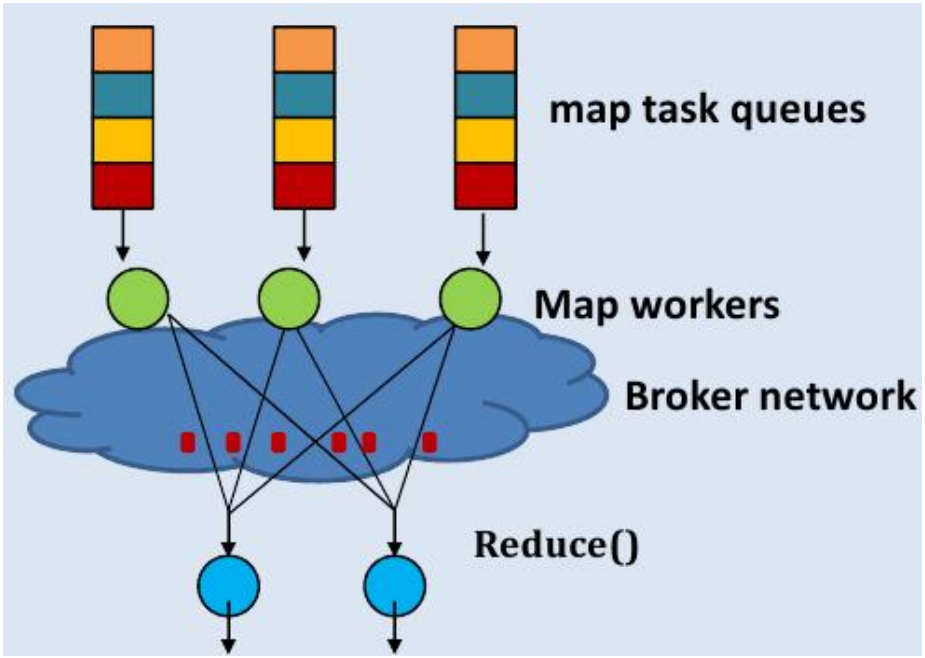
Gli intermediate data entrano nella memoria distribuita

- Gestisce gli intermediate data nella memoria distribuita dei workers
- I risultati di un map task viene inviato direttamente attraverso la broker network verso il reducers appropriato
- Per supportare applicazioni che utilizzano grandi quantità di intermediate data si potrebbe estendere Twister per fare si che vengano salvati sui dischi locali

Pub/Sub Messaging

- L'uso infrastruttura di tipo Pub/sub aumenta la scalabilità e l'efficienza del runtime
- I demoni possono comunicare con broker differenti
- Specialmente utile se si una mapReduceBcast()

Bechmark eseguiti su 624 Twister daemon e 5 Broker incrementano le prestazioni del broadcast di 4 volte



Scheduling Tasks

- Supporto per Long MapReduce task
- Twister utilizza uno scheduling statico
  - Supporta il task reuse
  - Potrebbe portare ad un utilizzo non ottimizzato delle risorse
  - Si può minimizzare questo effetto randomizzando l'assegnamento dei dati in input
- I cachable MapReduce tasks sono utili se la loro posizione resta fissa

Fault Tollerance

- Recupero dai fault solamente a livello dell' iterazione
- Non supporta il recupero di un singolo task
- Assunzioni:
  - La Broker network è affidabile
  - Il programma principale e i Twister drivers non possono avere fault
- Ogni fallimento Hardware/Daemon viene gestito secondo questi passi
  - Termina i tasks correnti
  - Cerca tra le risorse a disposizione un nuovo pool di Workers node
  - Configura MapReduce con i dati statici (Riassegna il task sulla base della località dei dati)
  - Riesegue l'iterazione fallita

Twister API

```
1.configureMaps(PartitionFile partitionFile)
2.configureMaps(Value[] values)
3.configureReduce(Value[] values)
4.runMapReduce()
5.runMapReduce(KeyValue[] keyValues)
6.runMapReduceBCast(Value value)
7.map(MapOutputCollector collector, Key key, Value val)
8.reduce(ReduceOutputCollector collector, Key key,List<Value>
  values)
9.combine(Map<Key, Value> keyValues)
```

Pairwise Distance Calculation

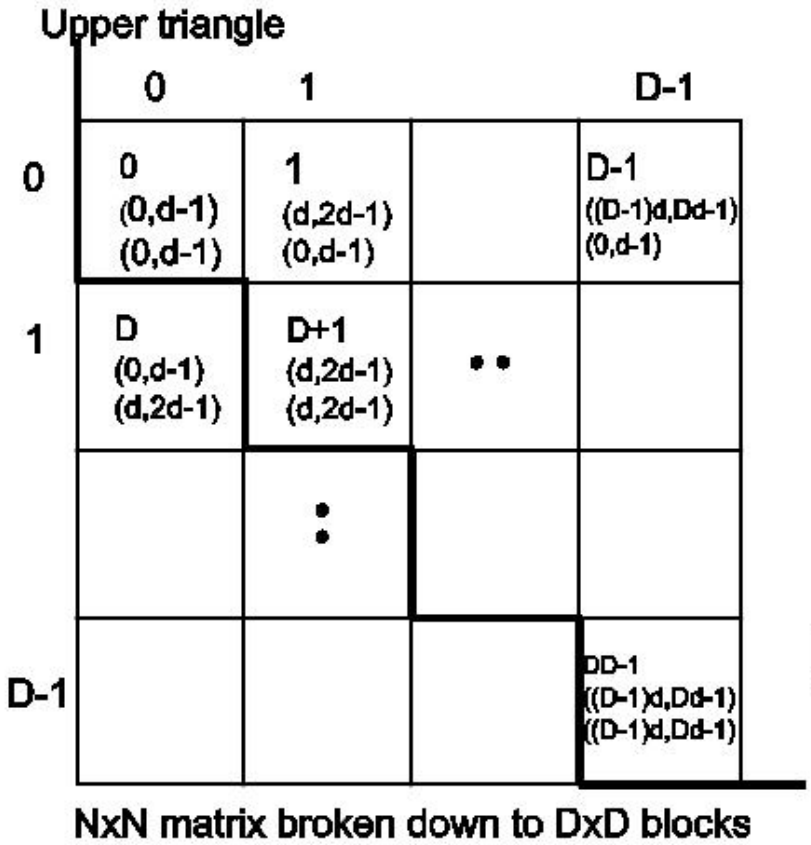
Calcolare la differenza che c'è tra tutti gli elementi di un insieme e tutti quelli di un altro

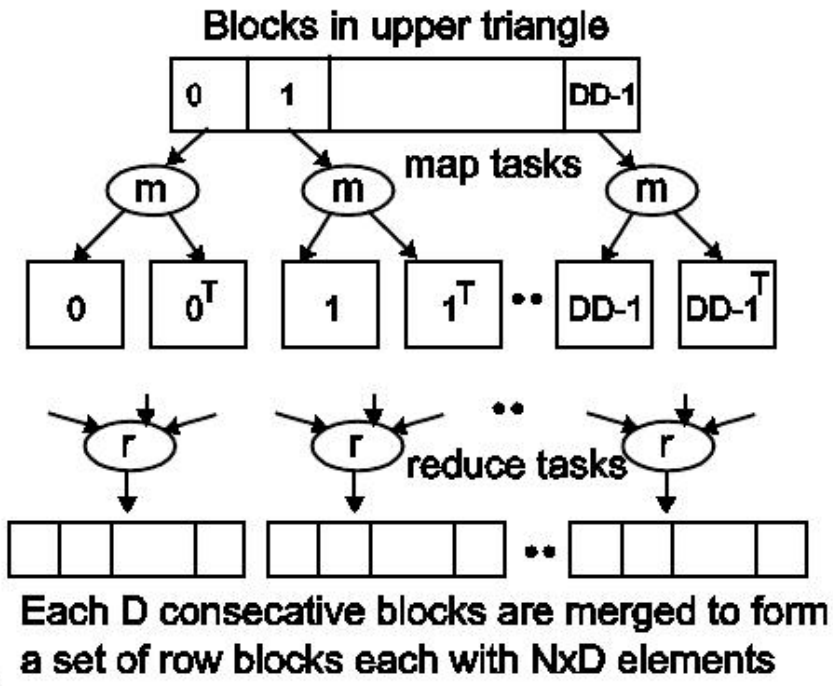
Pairwise Distance Calculation

Compare all elements of set A to all elements of set B via function F,yelding matrix M, such that $M[i, j] = F(A[i], B[j])$

- L'applicazione implementata è *Smith Waterman Gotho*
- Questa applicazione e stata implementata per dimostrare la capacita di Twister di supportare tipiche applicazioni di MapReduce

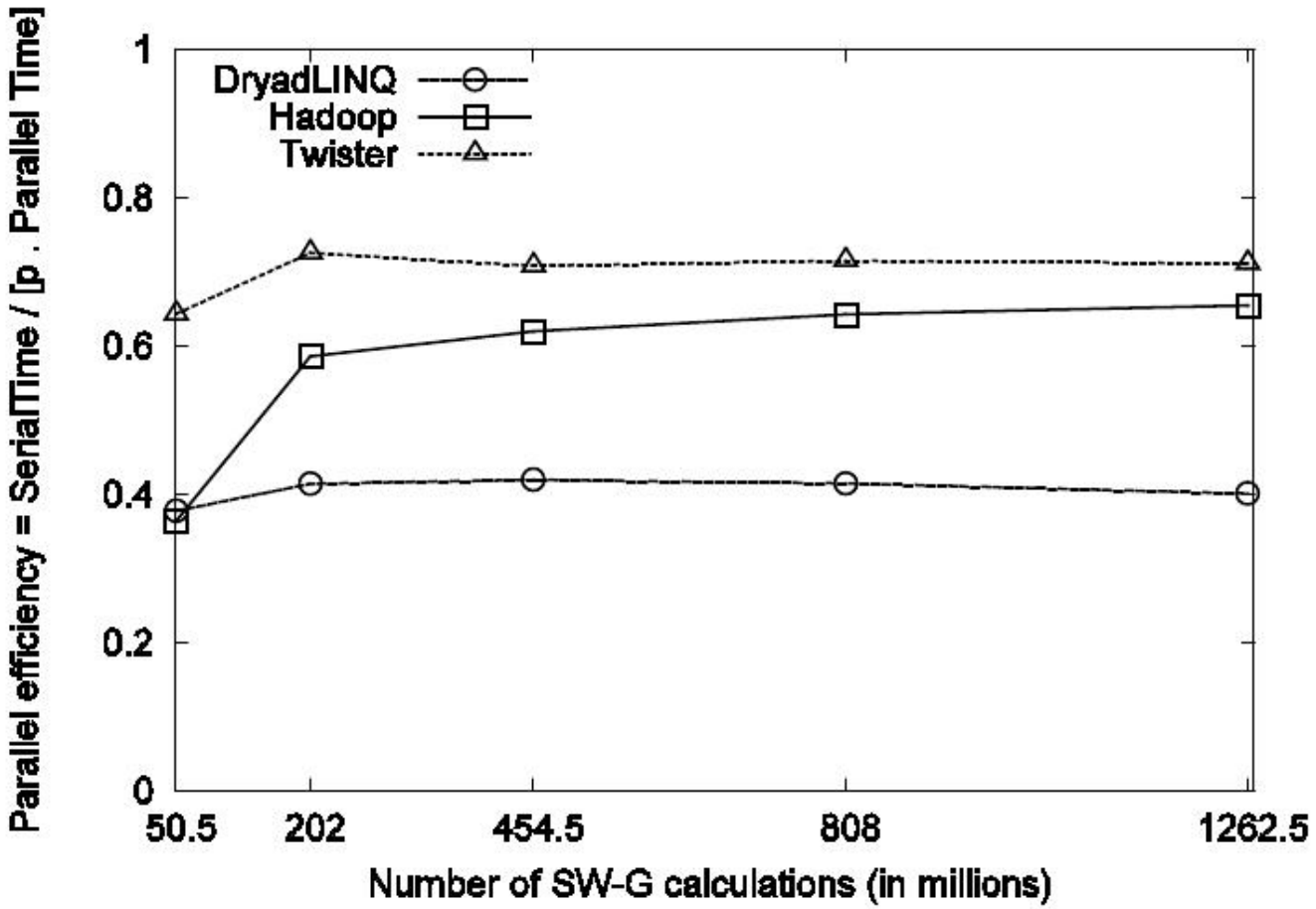
Implementazione per Twister





Implementazione per Twister

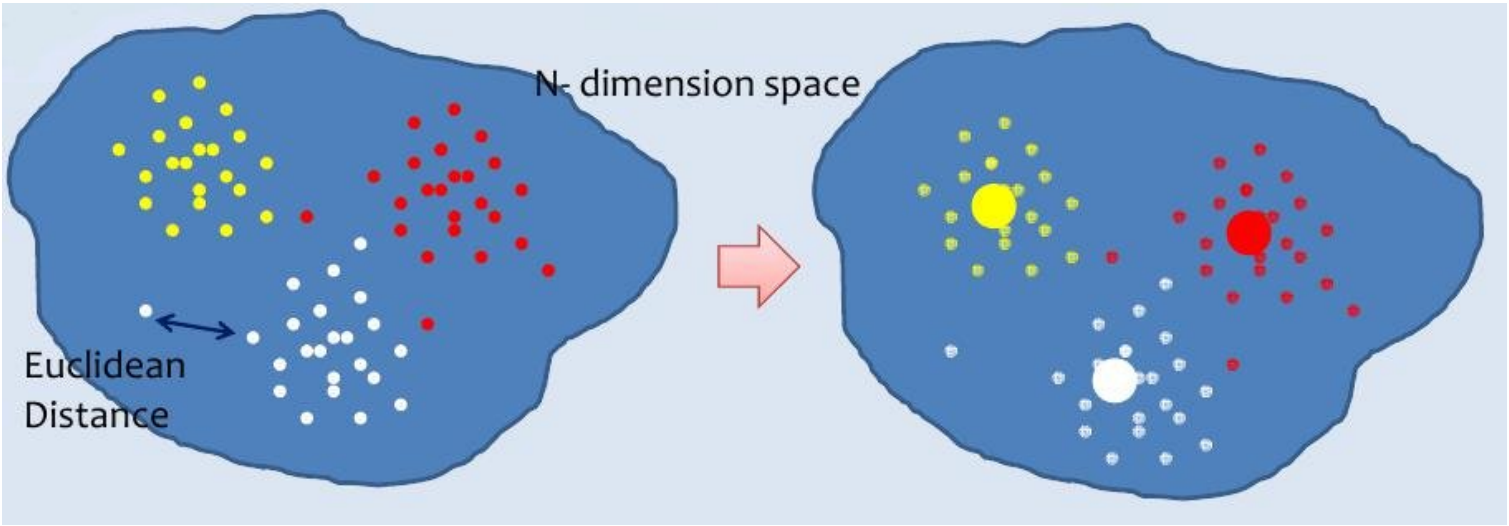
Parallel Efficency $\eta = \frac{T(1)}{(p * T(p))}$



Risultati dei Test

- Tutti i sistemi raggiungono la massima efficienza e la mantengono con l'incremento dei dati
- Per efficienza assoluta Twister supera gli altri runtimes grazie al suo più veloce sistema di comunicazione e le ottimizzazioni di scheduling

K-Means Clustering

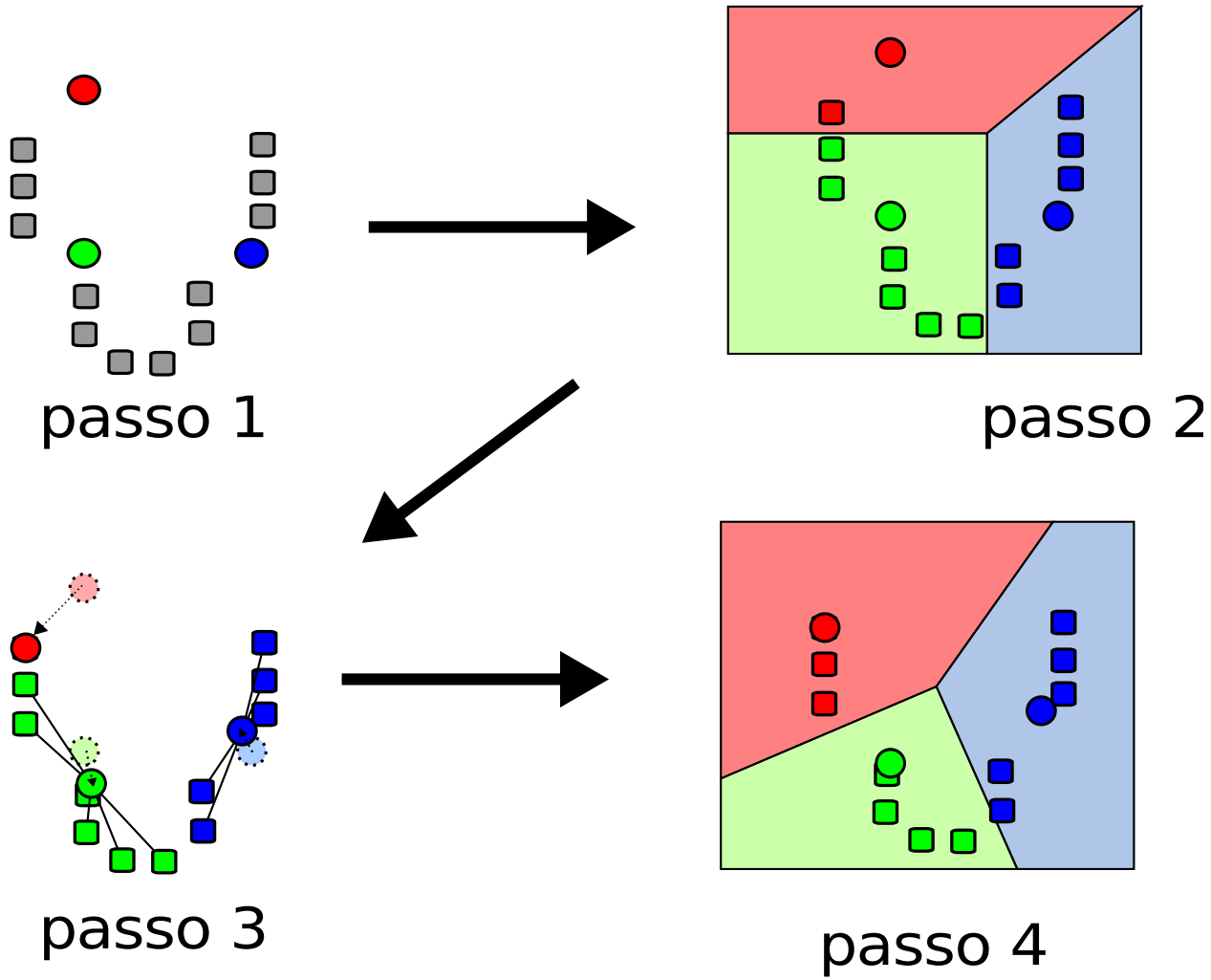


- Distribuzione di punti in uno spazio n-dimensionale
- Vengono individuati K centri
- Si calcolano le distanze di tutti i punti dal centro
- Iteriamo il processo per raffinare la posizione dei centri

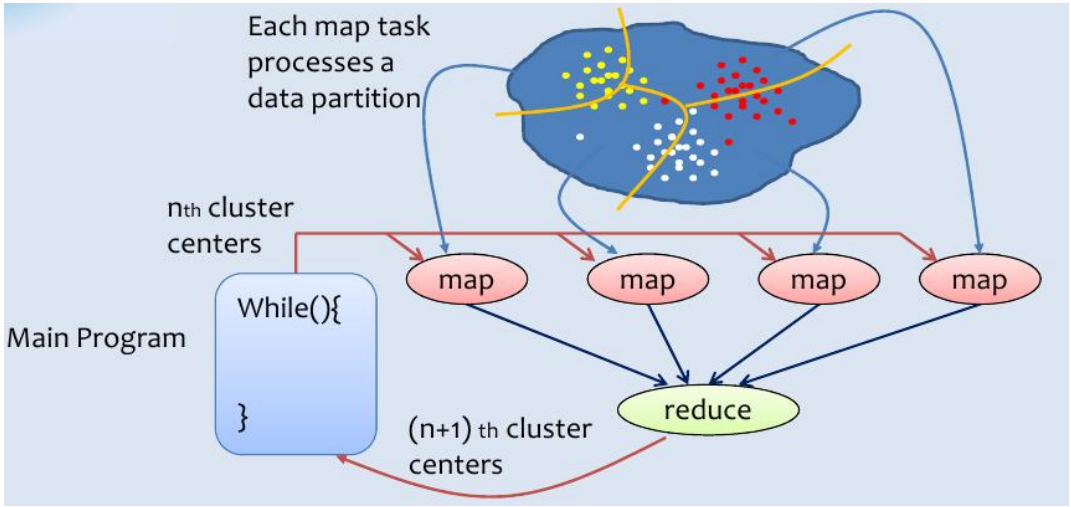
K-Means Algorithms

- Distribuzione di punti in uno spazio n-dimensionale
- Vengono individuati K centri
- Si calcolano le distanze di tutti i punti dai centri
- Iteriamo il processo per raffinare la posizione dei centri

Esempio di K-Means

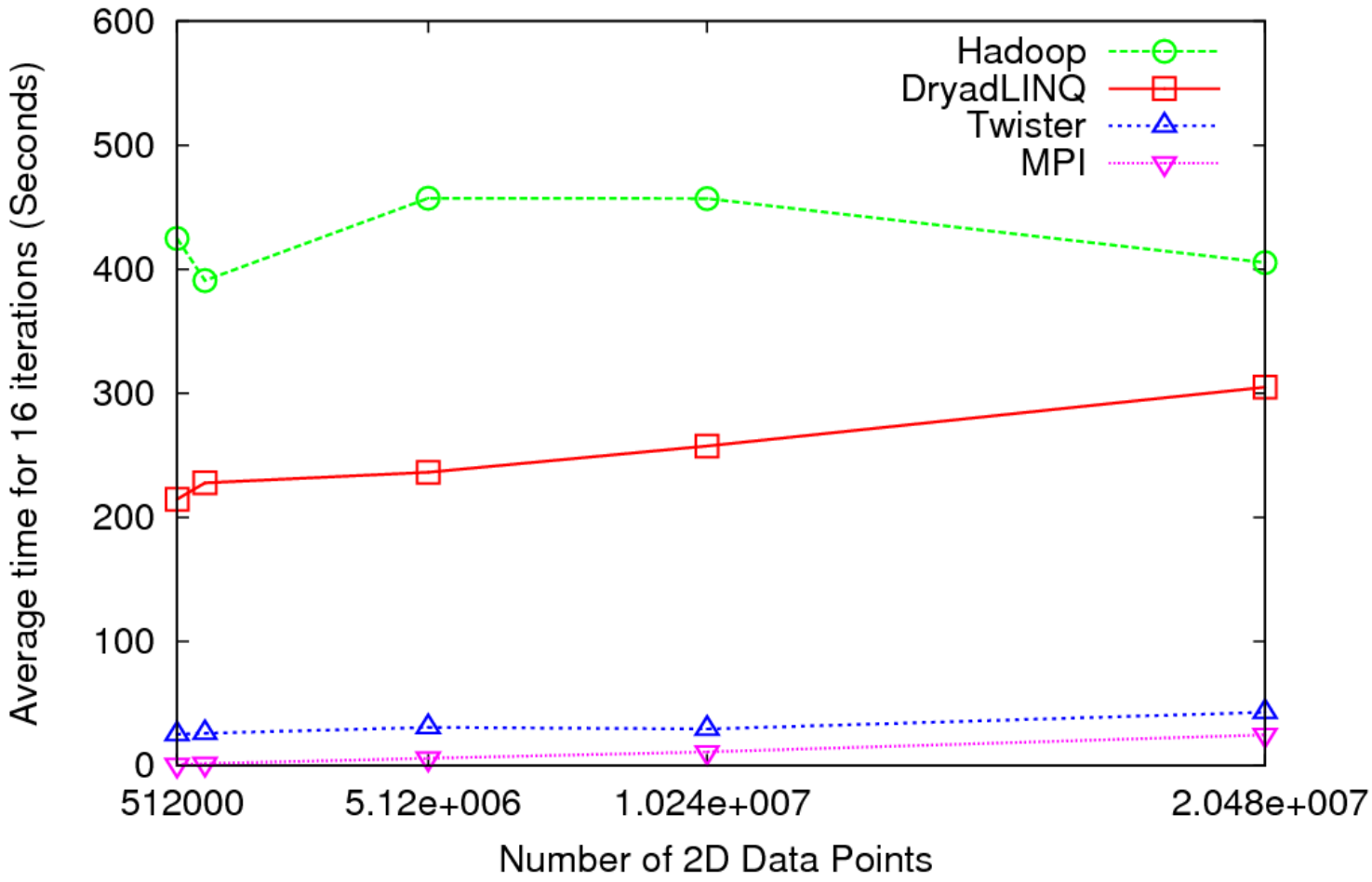


K-Means MapReduce



- I Map tasks calcolano la distanza tra ogni punto della loro partizione e tutti i centri
- Assegnano i punti a ogni centro e sommano tutti i valori parziali per ogni centro
- In output producono la somma per ogni centro e il numero di punti assegnati alla partizione
- I Reduce task calcolano tutte le corrispondenti somme parziali

K-means performance



Page Rank

Page Rank

Alghoritms that compute numerical value to each web pages,which reflect the probability that the random surfer will access this page
$x_i = \sum_{j \in B_i} (\frac{1}{N_j}) x_j$

- x_i importanza della pagina i
- $j \in B_i$ pagina j che è linkata alla pagina i
- N_j Numero di link uscienti dalla pagina j
- x_j Importanza della pagina j

Esempi di pagerank

dopo 1 iterazione

- Page A = 1
- Page B = 1
- Page C= 1

- Page A = 1.85
- Page B = 0.57
- Page C = 0.57

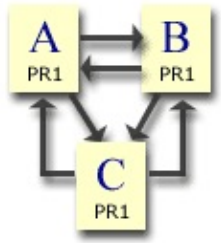
- Page A = 1.425
- Page B = 1
- Page C = 0.575

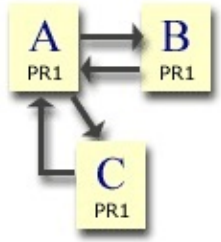
dopo 100 iterazioni

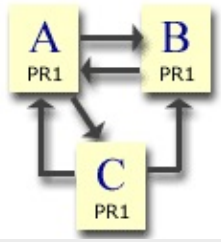
- Page A = 1
- Page B = 1
- Page C = 1

- Page A = 1.459
- Page B = 0.7702
- Page C = 0.7702

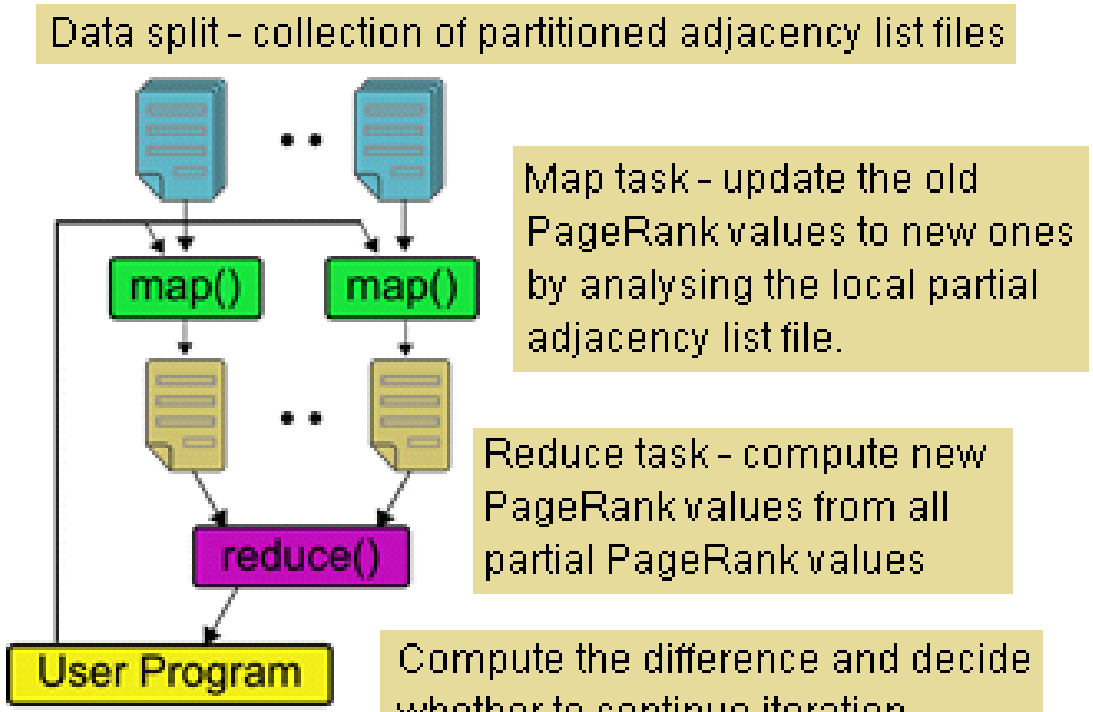
- Page A = 1.298
- Page B = 0.9999
- Page C = 0.7017







Page Rank for MapReduce



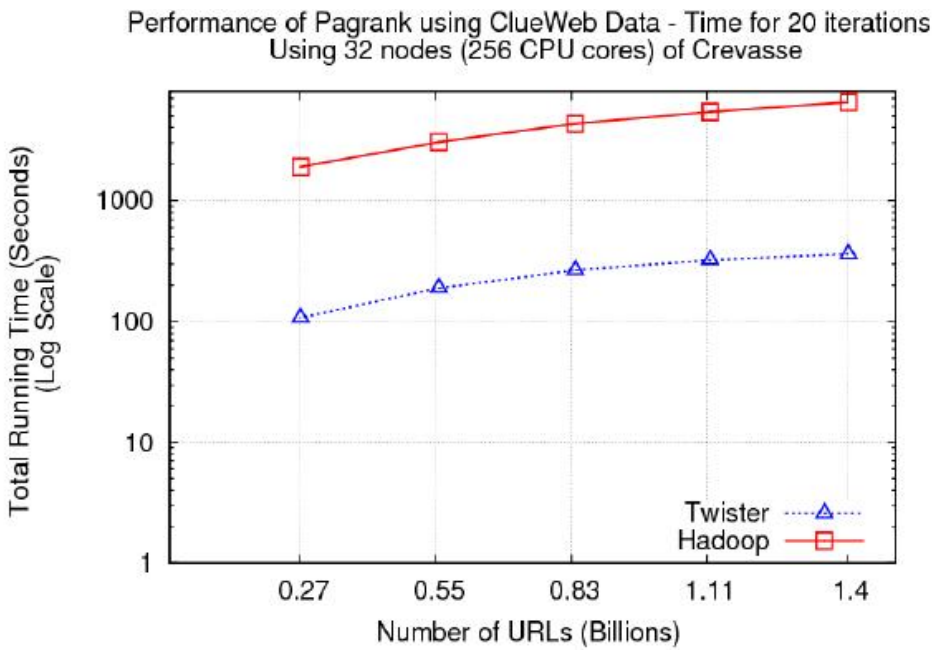
MapReduce for PageRank Algorithm

Page Rank for Twister

Per utilizzare le capacità di Twister si applicano 4 ottimizzazioni

- Configurare la matrice di adiacenza come come static data
- Utilizzare i valori di pagerank come variabile data
- Usare il broadcast per mandare i variabile data ai mapper
- Incrementare la granularità dei map tasks computando insiemi di URL

Twister Page Rank performance



I test sono stati effettuati con Clueweb come input e mostrano che è mantenuto l' 80% di efficienza utilizzando 256 core rispetto a utilizzarne 128

Conclusioni

- Twister estende MapReduce per algoritmi iterativi
- Sono stati implementati molti algoritmi.
- Paragonabile alle altre implementazioni in applicazioni di MapReduce classiche
- Estende il range di problemi risolvibili efficientemente con MapReduce

Sviluppi Futuri

- Ricercare una più efficiente infrastruttura di comunicazione per snellire le comunicazioni
- Incorporare in Twister un file system distribuito
- Estendendere ulteriormente il modello di programmazione per supportare più classi di applicazioni

Domande