Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in

Ingegneria Informatica

Database NoSQL, studio e implementazione di Riak

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Matteo Del Pioluogo

Anno Accademico 2013/2014

Dedicato a mio Nonno Giovanni
alle mie gioie Beatrice, Jacopo,
Camilla e Agata.
Ringrazio i miei Familiari,
il Prof. Di Nunzio
e tutti i miei cari amici.

Indice

1 Introduzione **1**

2 Riak **3**
2.1 Teorema CAP . . . 3
2.2 Organizzazione dei dati . . . 4
2.2.1 Chiavi . . . 5
2.2.2 Oggetti . . . 5
2.2.3 Buckets . . . 5

3 Cluster in Riak **7**
3.1 L'anello . . . 7
3.1.1 Gestione delle dei valori da parte dei vnodes . . . 8
3.1.2 Gossip Protocol . . . 8
3.2 Eventuale consistenza . . . 9
3.2.1 Scenari di errore . . . 9
3.2.2 Lettura quando un nodo primario è fuori uso . . . 10
3.2.3 Lettura quando tutti i nodi primari sono fuori uso . . . 10
3.2.4 Scrittura e lettura quando un nodo primario fallisce e successivamente ritorna . . . 11

4 Implementazione **13**
4.1 Il mio Twitter . . . 13
4.2 Il Cluster . . . 13
4.2.1 Rappresentazione schema Entità-Relazione . . . 14
4.2.2 Rappresentazione con Riak . . . 15
4.3 Python e Riak . . . 15
4.4 Popolare il Database . . . 16
4.5 Le richieste di dati . . . 19
4.5.1 Richieste con il metodo get . . . 19
4.5.2 Richieste con Map-Reduce . . . 20

5 Conclusioni **23**

Elenco delle figure

3.1 Anello, esempio di partizioni 8
3.2 Replicazione Anello Riak 9

4.1 Il mio Twitter, schema relazionale 15
4.2 Rappresentazione grafica della nostra implementazione di Twitter con Riak 16
4.3 Grafico inserimento 18
4.4 Schema Funzionamento Map Reduce 21
4.5 Grafico Prestazioni della Map Reduce 22

Elenco delle tabelle

4.1 Inserimento con un nodo fisico 17
4.2 Inserimento con due nodi fisici 17
4.3 Inserimento con tre nodi fisici 18
4.4 Inserimento con quattro nodi fisici 18
4.5 Inserimento con quattro nodi fisici, uno dei quali in Wi-Fi 19
4.6 Prestazioni con il metodo get, in un database una popolazione di 1000 utenti 20
4.7 Prestazioni con il metodo get, in un database una popolazione di 1000 utenti 21

Capitolo 1

Introduzione

Il termine NoSQL (Not Only SQL), un'ampia famiglia di nuovi database che non seguono i principi dei database relazionali (o RDBMS), ovvero non memorizzano i dati in classici schemi tabelle-relazioni e non viene usato SQL come linguaggio di interrogazione e manipolazione. I database NoSQL sono spesso confrontati con diversi criteri, quali la scalabilità, le prestazioni e la consistenza:

- (Scalabilità): indica la capacita di un sistema di accrescere o decrescere le proprie prestazione a seconda delle necessità.
- (Prestazione): Solitamente viene stimata tramite un applicazione di benchmark (alcuni NoSQL hanno già un istruzione interna che svolge questa funzione) la quale verifica i vantaggi su certi tipi di operazioni.
- (Consistenza): una base di dati si dice consistente, quando una transizione viene eseguita dallinizio alla fine senza linterferenza di altre transizioni e dovrebbe portare la base di dati da uno stato consistente ad un altro.

I database NO-SQL, non sono una novità, infatti il termine venne forgiato dall'italiano Carlo Strozzi nel 1998, ma sono una valida alternativa ai RDBMS, infatti hanno alcune peculiarità che posso essere molto utili:

- NO-SQL abbraccia totalmente la filosofia (open - source), i database relazionali invece spesso non lo sono.
- I dati sono altamente (portabili) su sistemi differenti.
- Non definisce uno schema rigido (schemaless) e non occorre tipare i campi, per cui non esistono limiti o restrizioni ai dati memorizzati nei database NO-SQL.
- **V**elocità di esecuzione, interrogazione di grosse quantità di dati e possibilità di distribuirli su più sistemi (replicazione dei dati), con un meccanismo totalmente trasparente all'utilizzatore;
- I DBMS NO-SQL si focalizzano su una (scalabilità) orizzontale e non verticale come quelli relazionali.

La prima parte del progetto sarà incentrata nello studio di Riak[1] un NO-SQL. Infatti è importante conoscere come Riak gestisce i dati e quali sono le opzioni più interessanti che possono essere modificate per ottenere maggiori vantaggi. Nella seconda parte costruiremo un piccolo cluster installandoci sopra Riak, dopodiché progetteremo una piccola applicazione, la quale simula un famoso social network. L'applicazione dovrà avere degli utenti registrati, ogni utente dovrà poter scrivere dei messaggi e conoscere i messaggi degli altri utenti. Nell'ultima parte del progetto analizzeremo come con Riak si possono manipolare i dati e quali prestazioni si ottengono con il nostro semplice cluster.

[1] http://basho.com/riak/

Capitolo 2

Riak

Riak è un database open source con licenza Apache 2.0, nato nel 2009 influenzato dal Teorema CAP formulato da Eric Brewer e dal Amazons Dynamo Paper[1].

2.1 Teorema CAP

Essendo un sistema distribuito anche Riak è vincolato dal teorema CAP. Il teorema CAP afferma che un sistema distribuito non può garantire contemporaneamente tutte e tre le seguenti caratteristiche:

- **Coerenza**: Tutti i nodi hanno gli stessi dati nello stesso momento.
- **Disponibilità**: La garanzia di una risposta anche in caso di fallimento.
- **Tolleranza**: Il sistema continua a funzionare nonostante guasti o errori.

Riak ha le seguenti caratteristiche:

- **Disponibilità**: Riak replica e recupera i dati in modo intelligente, affinché sia disponibile per operazioni di lettura e scrittura, anche in condizioni di guasto.
- **Tolleranza ai guasti** perdere l'accesso a molti nodi a causa di partizione della rete o di guasto hardware e non perdere mai i dati.
- **Operazioni Semplici**: Aggiungere o rimuovere macchine da un cluster Riak è molto semplice, la complessità non cambia anche se il cluster è di grandi dimensioni.
- **Scalabilità**: Riak distribuisce automaticamente i dati in tutto il cluster e produce un aumento di prestazioni quasi lineare come si aggiunge la capacità.

[1] http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

Da queste caratteristiche si può dire che Riak, soddisfa contemporaneamente, secondo il teorema CAP, la disponibilità e la tolleranza mentre non soddisfa la coerenza dei dati. Per soddisfare la consistenza, dei dati dunque usa un modello più debole chiamato **Eventual Consistency**.

Un buon motivo per usare Riak è quando i dati che stiamo trattando non stanno tutti in un solo dispositivo fisico. La disponibilità di Riak è un ottima caratteristica quando i tempi di attesa sono inaccettabili. Nessuno può promettere che Riak abbia un tempo di funzionamento continuo, ma Riak è progettato per resistere a partizioni della rete e a guasti hardware, che invece interferirebbero significativamente con la maggior parte degli altri database. Un'altra caratteristica è la sua prevedibile latenza, infatti le sue operazioni fondamentali (scrittura, lettura, eliminazione) non coinvolgono unioni di dati complessi. In questo modo risolve tempestivamente ogni richiesta, per questo motivo spesso è scelto come backend di memorizzazione dati per altri software di gestione dei dati.

Basho (società che sviluppa Riak), afferma che i cluster devono essere almeno di 5 nodi quindi Riak potrebbe essere eccessivo per i piccoli database. Ovviamente però se si prevede di avere una crescita molto rapida è unottima scelta per giocare di anticipo. Riak utilizza un modello di dati molto semplice (chiave/valore), questo significa che per essere performante i nostri dati devono essere denormalizzati. Spesso è molto semplice avere dati di questo tipo, in caso contrario Riak non è la soluzione migliore al problema che volevamo risolvere. Riak permette di individuare i valori che servono tramite alcuni criteri specifici, mentre se la nostra applicazione ha bisogno di molte interrogazioni e non bastano solo dei valori chiave allora sicuramente Riak non è una buona soluzione. Spesso il carico di lavoro non è molto prevedibile o facilmente quantificabile, motivo per cui Basho ha messo a disposizione un utile comando (basho_bench[2]) per fornire un analisi delle prestazioni della nostra base di dati.

2.2 Organizzazione dei dati

In una base di dati relazionale, i dati sono organizzati in tabelle e ogni dato è rappresentato come una relazione. All'interno di tali tabelle ogni dato ha una sua riga ed è organizzato dividendo i suoi vari attributi tramite le colonne. È possibile la modifica o il recupero di intere tabelle, righe singole o un gruppo di colonne all' interno di un insieme di righe. Al contrario Riak ha un modello per l'organizzazione dei dati molto più semplice, in cui l'oggetto è sia l'elemento più grande che il più piccolo con cui avremo a che fare. Infatti durante una qualsiasi manipolazione dei dati l'oggetto deve essere sempre completamente recuperato e aggiornato, non esistono recuperi o modifiche parziali.

[2]https://github.com/basho/basho_bench/

2.2.1 Chiavi

Le chiavi in Riak sono semplicemente dei valori binari (o stringhe) utilizzati per identificare gli oggetti. Per il Client che si interfaccia a Riak, ciascun Bucket sembra rappresentare uno spazio delle chiavi separato. Infatti è importante sapere che Riak considera la coppia Chiave-Bucket come un'unica entità per operazioni di interrogazione.

2.2.2 Oggetti

Gli oggetti sono l'unica unità di memorizzazione dei dati in Riak. In Riak gli oggetti sono essenzialmente strutture individuate da un Bucket e da una Chiave. L'oggetto contiene una lista di coppie contenente i metadati.

2.2.3 Buckets

I buckets sono uno spazio virtuale in cui archiviare chiavi che fanno riferimento a dati simili. I buckets possono essere pensati come tabelle se li si compara ai database relazionali oppure a delle cartelle se comparati ad un file system. I buckets come configurazione predefinita lasciano la massima libertà di scelta, ma è possibile ovviamente configurare i vari buckets con attributi specifici, quando viene modificato un attributo l'intero anello riceve una notifica del cambiamento tramite un protocollo chiamato **gossip protocol** di cui parleremo più avanti.

- Un attributo molto potente è n_val, che permette di indicare il numero di copie che ogni oggetto deve avere dentro il cluster.
- Per specificare se un oggetto può essere una copia di uno già esistente si usa allow_mult.
- Con gli attributi r, pr, w, dw, pw e rw si possono indicare il numero di letture/scritture necessario per essere sicuri che una operazione si avvenuta con successo.
- L'attributo Precommit può contenere una lista di funzioni (Scritte in Erlang e Javascript) da eseguire prima di scrivere un oggetto.
- L'attributo Postcommit può contenere una lista di funzioni (Javascript, Erlang) da eseguire dopo aver scritto un oggetto.

Capitolo 3

Cluster in Riak

Riak è stato sviluppato per essere utilizzato in un cluster. Il cluster è formato da un insieme di nodi che nel mondo fisico rappresentano hosts. Ogni nodo ha un suo insieme di nodi virtuali (vnodes), ognuno dei quali è responsabile per l'archiviazione di porzioni diverse dello spazio delle chiavi. I nodi non sono cloni l'uno dell'altro, né tutti partecipano a soddisfare la stessa richiesta. Il modo in cui i dati vengono replicati, quando, con quale modo vengono uniti, quale modello di fallimento utilizzare, è configurabile in fase di esecuzione. Due importanti valori da conoscere in un cluster Riak sono il valore W e R.

Le API di Riak permettono di settare il valore W che deve essere soddisfatto ad ogni Update. W rappresenta il numero di nodi che devono restituire un successo per poter considerare una richiesta di Update completata. Questo permette a Riak di funzionare nonostante alcuni nodi siano irraggiungibili o vi sia del lag (una latenza alta).

Come la scrittura anche la lettura ha il suo valore da settare con le API di Riak, questo valore è chiamato R. Indica il numero di nodi che devono restituire un risposta positiva prima di poter considerare la lettura completa.

3.1 L'anello

L'interfaccia utente di Riak ci permette di agire direttamente su chiavi e buckets, Riak internamente computa in un hash da 160-bit le coppie bucket/chiave e le mappa in un 'anello' ordinato. Questo anello è diviso in partizioni ed ogni vnodes è responsabile di una partizione. I nodi di un cluster Riak cercano sempre avere un numero all'incirca uguale di vnodes. Nel caso generale, ciò significa che ogni nodo nel cluster è responsabile di 1 / (numero di nodi) dell'anello, o (numero di partizioni) / (numero di nodi) vnodes. Per esempio, se due nodi definiscono un cluster con 16 partizioni, ogni nodo avrà avviato 8 vnodes. Ad intervalli regolari ogni nodo cerca di rivendicare le sue partizioni lungo lanello in modo tale da mantenere una distribuzione uniforme tra i nodi membri e che nessun nodo sia responsabile di più di una replica di una chiave.

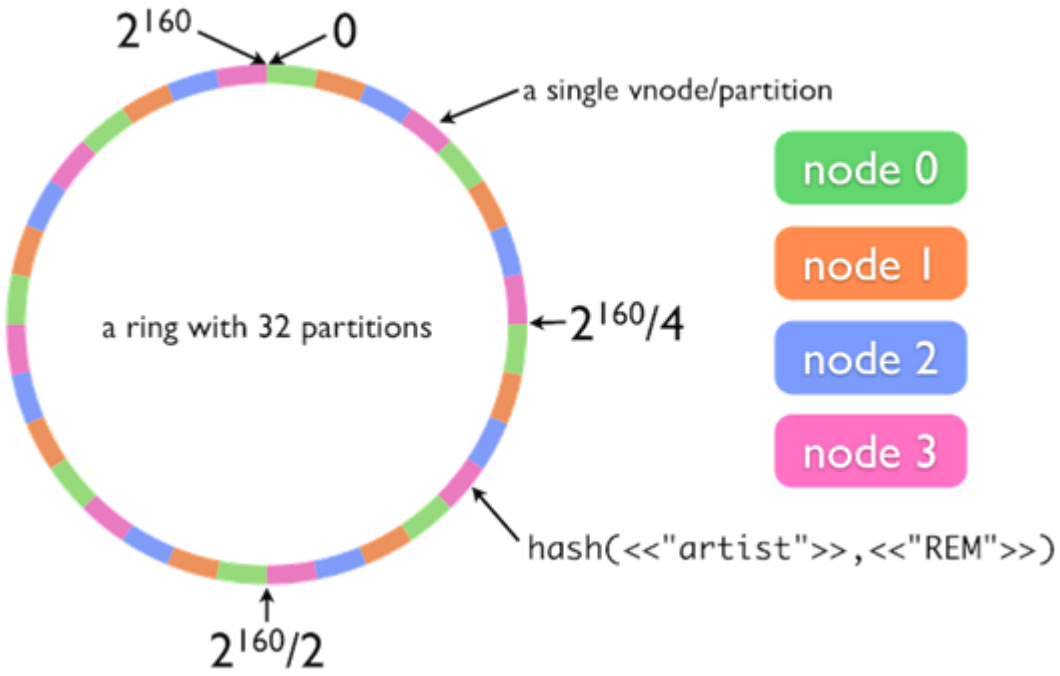

Figura 3.1: Anello, esempio di partizioni

3.1.1 Gestione delle dei valori da parte dei vnodes

Quando un valore sta per essere memorizzato nel cluster, ogni nodo può partecipare come coordinatore della richiesta pendente. Il nodo di coordinamento legge lo stato dell'anello per determinare quale vnode possiede la partizione in cui la chiave del valore appartiene, quindi invia la richiesta put a quel vnode, successivamente anche ai vnodes responsabili per i prossimi N-1 partizioni sul ring, dove N è un parametro configurabile nel bucket che descrive il numero di copie del valore da memorizzare. La richiesta di put può anche specificare che almeno W (vnodes che saranno $\leq$ N) rispondano con successo che l'operazione è avvenuta, gli altri N-W vnodes mancanti potranno replicare il loro successo anche che si è ricevuta la prima risposta affermativa da parte degli W vnodes. Richieste di fetch o get funzionano con un meccanismo molto simile. Ovviamente le richieste di get hanno al posto di W il parametro R.

3.1.2 Gossip Protocol

Lo stato dell'anello è conosciuto da tutti i nodi del cluster grazie ad un protocollo gossip. Ogni volta che un nodo cambia il suo spazio di influenza sul ring, annuncia il suo cambiamento attraverso questo protocollo. Inoltre periodicamente, con questo protocollo, ri-annuncia ciò che sa circa l'anello, nel caso ad alcuni nodi manchino aggiornamenti precedenti.

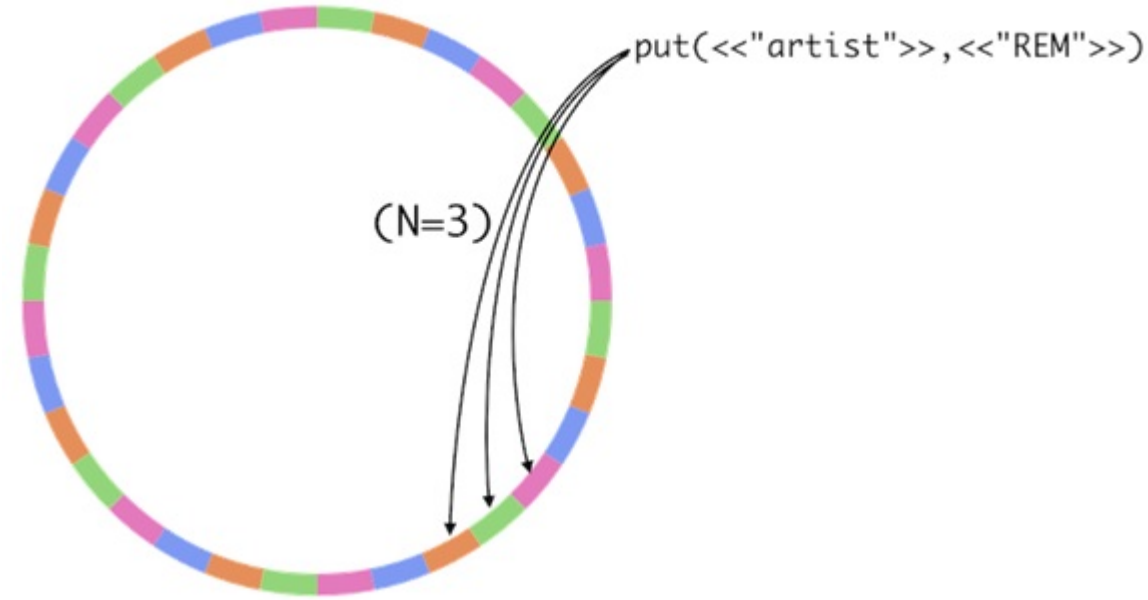

Figura 3.2: Replicazione Anello Riak

3.2 Eventuale consistenza

In un ambiente distribuito e fault-tolerant (tolleranza ai guasti) come Riak, l'imprevisto è previsto. Ciò significa che i nodi possono lasciare e unirsi al cluster in qualsiasi momento, sia che si tratti per caso (errore del nodo, partizione della rete, ecc) o di proposito, come la rimozione di un nodo dal cluster Riak. Anche con uno o più nodi irraggiungibili, il cluster è ancora capace di accettare scritture e servire letture.

3.2.1 Scenari di errore

Per comprendere meglio cosa accade durante un guasto analizziamo alcuni esempi di guasti tipici. Nel caso elementare di guasto, almeno un nodo è fuori uso, lasciando due repliche intatte allinterno del cluster. I Clients possono comunque aspettarsi che le loro richieste vengano esaudite dagli altri 2 nodi. Il tutto grazie ad una specifica dell'applicazione che implementa una Graceful Degradation (capacità di un sistema di mantenere una certa qualità del servizio nonostante guasti) in modo automatico oppure che può essere utilizzata durante il runtime, o basta semplicemente riprovare fino la parte dei dati che si cercava non viene trovata quando la prima richiesta non viene completamente esaudita. Ricordiamo che ogni chiave appartiene a N nodi virtuali. Chiamiamo primari i vnodes che sono direttamente interessati alla chiave. Interessanti, in caso di guasti, sono i nodi virtuali secondari che sono eseguiti su nodi con partizioni primarie vicine nello spazio delle chiavi come controfigure di quelli primari quando non sono disponibili (denominato fallback).

Quando Riak deve esaudire una richiesta deve fare le seguenti operazioni elementari:

- Determinare i vnodes responsabili per la chiave richiesta.
- Inviare una richiesta a tutti i vnodes determinati nella fase precedente.
- Attendere fino a quando le richieste vengo confermate in numero sufficiente in modo da soddisfare il quorum.
- Restituire il valore relativo alla richiesta posta dal Client.

Ora per esempio, ipotizziamo di avere un cluster composto da 5 nodi sani e con un replicazione dei dati pari a 3.

3.2.2 Lettura quando un nodo primario è fuori uso

- I dati sono scritti in una chiave con W=3.
- Un nodo diventa irraggiungibile, questo accade al nodo primario per questa chiave.
- I dati sono letti con una data chiave e R=3.
- Quindi Riak ritorna not_found alla prima richiesta.
- Riak legge i dati da un nodo secondario dove sono stati nel frattempo replicati. La lettura di riparazione avviene sempre indipendente dal valore di R, anche con una R di 2, questo per assicurarsi che i nodi che si occupano di una certo dato siano consistenti.
- La successive lettura ritorna un valore corretto di lettura per un R=3, 2 valori vengono da un nodo primario e 1 da un secondario.

Notare che se R fosse stato 2 allora si avrebbe subito avuto una risposta affermativa.

3.2.3 Lettura quando tutti i nodi primari sono fuori uso

- I dati sono scritti in una chiave con W=3.
- Tutti i nodi primari vanno fuori uso.
- I dati sono scritti con R=3.

Questo incidente darà sempre un errore di not_found, non è possibile fare un lettura di riparazione, perché essendo che sono caduti tutti i nodi primari non è stato possibile fare neanche una replica.

3.2.4 Scrittura e lettura quando un nodo primario fallisce e successivamente ritorna

- Un primario è irraggiungibile.
- I dati sono scritti con W=3.
- Un secondario prende la responsabilità di scrivere.
- Il primario torna in raggiungibile.
- In un tempo predefinito di 60 secondi, avverrà il trasferimento di dati che aggiorneranno il nodo primario appena tornato raggiungibile.
- Dopo che l'handoff (il trasferimento per aggiornare il primario) sarà avvenuto, il nodo primario sarà di nuovo pronto per servire altre richieste.

Notare che se R fosse stato 2 allora si avrebbe subito avuto una risposta affermativa.

Capitolo 4

Implementazione

L'esplosione dei social network avvenuta negli ultimi anni, ha aperto la strada a i NoSQL, questo per varie ragioni come la scalabilità. Per questo motivo ho scelto di verificare le prestazioni di un piccolo cluster Riak e implementare una semplice versione del famoso social network Twitter.

4.1 Il mio Twitter

Per prima cosa, pensiamo come dovrebbe essere una semplice implementazione di Twitter, e quindi le seguenti voci da implementare:

- **Utente**: la persona fisica che si è registrata al servizio e che con dei post scritti nella sua pagina personale, può raccontare qualcosa a tutti i visitatori della sua pagina.
- **Tweet**: ciò che la persona scrive, che quindi dovrà avere un riferimento alla persona che l'ha scritto. Tali contenuti, sono definiti tweet per la loro lunghezza che non può superare i 140 caratteri ricordano il cinguettio di un uccellino, un idea analoga è quella di Post.
- **Followers**: Utenti che hanno deciso di seguire uno stesso utente, cioè hanno nella loro pagina la possibilità di visualizzare cosa lutente scrive nei suoi Tweet. Ogni utente può essere follone di quante persone vuole, e può seguire quante persone vuole.
- **Following**: Utenti che un determinato utente ha deciso di seguire.

4.2 Il Cluster

Riak è utilizzato per problemi distribuiti, ma un cluster commerciale non ha un prezzo abbordabile, quindi la nostra simulazione utilizzeremo, dei dispositivi fisici chiamati Raspberry Pi. Il Raspberry PI è un calcolatore, implementato su una sola scheda elettronica, altre caratteristiche principali da tenere in considerazione sono:

- Processore: ARM1176JZF-S da 700 MHz.
- Memoria Ram: uno 256 MB gli altri due 512 MB.
- Memoria fisica: ogni Raspberry ha una scheda SD da 16 GB per memorizzare dati.
- Scheda di Rete da 10 Mb, che potrebbe essere un collo di bottiglia.
- Il sistema operativo: Raspbian, un gratuito sintema operativo basato su Debian e ottimizzato per funzionare con lhardware del Raspberry Pi.
- La versione di Riak installata è la 1.4.2.
- I loro nomi nell'anello saranno: Riak@192.168.0.101, Riak@192.168.0.102, Riak@192.168.0.103 (l'ultimo è quello da 256 MB di ram)

Inoltre per approfondire ancora di più l'aspetto del database distribuito, faremo una ulteriore simulazione con 4 macchine fisiche. La 4 macchina è un laptop Asus A3000 con le seguenti caratteristiche:

- Processore: Intel pentium m 725.
- Memoria Ram : 512MB.
- Memoria fisica: HD da 80 GB.
- Sistemema operatvo : Ubuntu.
- Nome allinterno dellanello: Riak@192.168.0.106

Questo per monitorare i possibili cambiamenti introducendo una macchina molto diversa rispetto alle altre nellanello.

4.2.1 Rappresentazione schema Entità-Relazione

Per spiegare quali sono i dati che il nostro database gestirà si può pensare a quale sarebbe lo schema ER se usassimo un RDBMS.

La figura 6.1 rappresenta come avremo dovuto creare la nostra base di dati in maniera relazionale. Avremmo due sole entità Tweet e Utente, due relazioni scrive e segue. Ogni Utente può aver scritto zero o N Tweet, ma ogni Tweet deve essere stato scritto da uno e unico utente. Mentre ogni utente può seguire zero o N utenti, inoltre ogni utente può essere seguito da zero o N utenti.

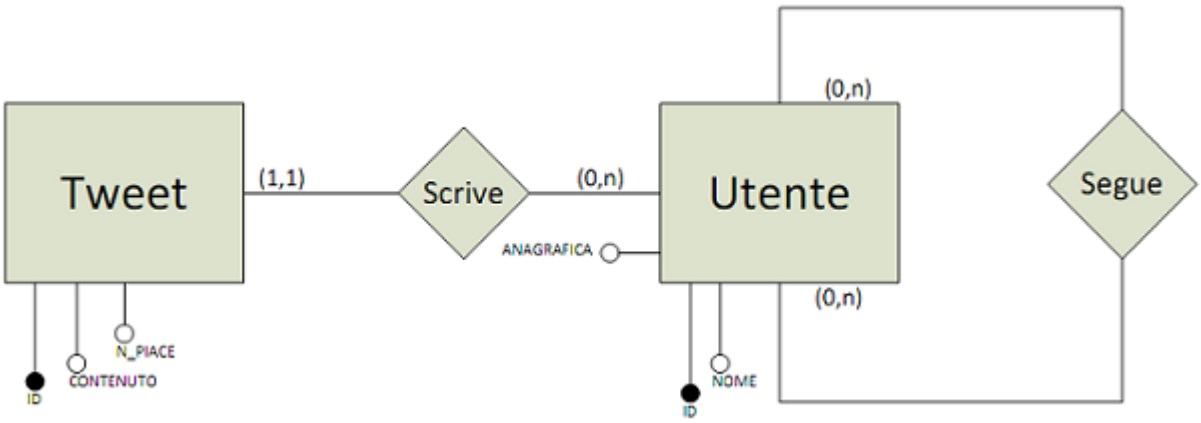

Figura 4.1: Il mio Twitter, schema relazionale

4.2.2 Rappresentazione con Riak

Utilizzando un NoSQL abbiamo molto più scelta di espressione, cioè non esiste un unica soluzione per rappresentare il modo con il quale gestiremo i nostri dati. Nel nostro caso l'idea è creare un bucket per ogni aspetto principale della nostra applicazione, cioè Utente, Follower, Following e Tweet. Ogni bucket conterrà degli oggetti la cui chiave è la stessa se quell'oggetto ha a che fare con lo stesso utente. Cosi ogni buckets avrà al suo interno oggetti che contengono valori o vettori. Un altra soluzione poteva essere di fare un unico bucket, con oggetti molto complessi, cioè ogni oggetto contiene tutte le informazioni di un utente, ma questo rispecchia un NO-SQL orientato ai documenti, mentre Riak è chiave valore. Un esmpio, Alice è un utente la cui chiave per accedere all'oggetto contente tutta la sua anagrafica nel bucket Utente è la stringa Utente1. Se Alice scrive un tweet, questo viene memorizzato nel bucket Tweet in un oggetto la cui chiave è proprio Utente1. Lutente Bob (la cui chiave è Utente2) decide di diventare un follower di Alice, quindi l'applicazione dovrà salvare in (Following, Utente2) la chiave di Alice, mentre in (Follower, Utente1) la chiave di Bob.

4.3 Python e Riak

Per la manipolazione dei dati useremo Python 2.7 e la libreria[1] di Python Riak fornita da Basho. Per prima cosa dobbiamo collegarci all'anello:

```
ip='192.168.0.101'
myClient=riak.RiakClient(host=ip,pb_port=8087,protocol='pbc')
```

Ora creiamo I 4 bucket.

```
UserBucket = myClient.bucket('User')
TweetBucket = myClient.bucket('Tweet')
FollowerBucket = myClient.bucket('Follower')
FollowingBucket = myClient.bucket('Following')
```

[1]https://github.com/basho/riak-python-client

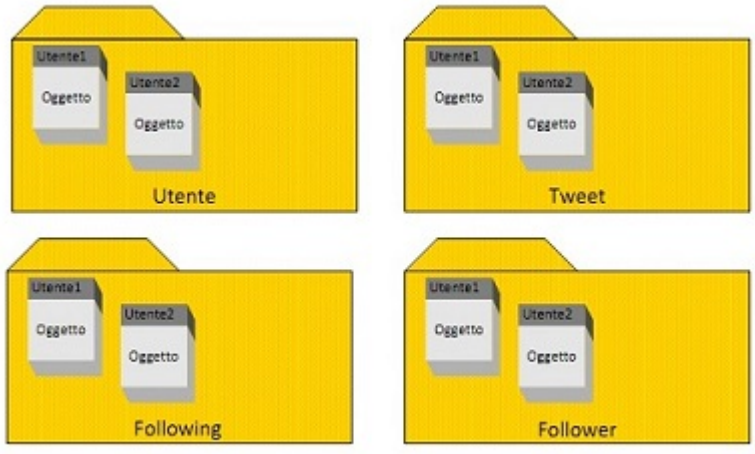

Figura 4.2: Rappresentazione grafica della nostra implementazione di Twitter con Riak

Ora scelgo come struttura per memorizzare i dati nei buckets Utenti e Tweet una mappa

```
User={'id':"Utente1",'name':"Alice",'surname':"Rossi",'birth':"16/08/1956"}
Tweet={'id':"Utente1",'value':"Hello World"}
```

Mentre i follower e i following li memorizzeremo in una lista

```
Follower=[Utente2,Utente3]
Following=[Utente2,Utente4,Utente5]
```

Per memorizzare i dati nel bucket si usano sempre gli stessi comandi nonostante le strutture dati diverse

```
newUser=UserBucket.new("Utente1", data=User)
newUser.store()
newfollowing = FollowingBucket.new("Utente1",data=Following)
newfollowing.store()
```

Per compiere una ricerca invece basta conoscere la chiave e il bucket. In questo caso verra restituito loggetto contente tutti i dati anagrafici di Alice.

```
fetchedUser = UserBucket.get( Utente1 )
```

Infine per eliminare qualcosa, ad esempio il commento di alice

```
fetchedTweet = TweetBucket.get(Utente1)
fetchedTweet.delete()
```

4.4 Popolare il Database

Come prima cosa abbiamo creato un programma in Python per simulare in maniera automatica il popolamento del nostro database. La prima simulazione sarà con Riak attivo in un solo nodo fisico e da un secondo terminale verrà eseguito

il programma per popolare la base di dati, dopo alcune simulazioni, incrementeremo il numero di nodi per vedere eventuali discostamenti nelle tempistiche di inserimento. Il programma simulerà in maniera sequenziale N utenti, i quali uno alla volta inseriranno i loro dati, un Tweet di lunghezza variabile fino a 70 caratteri e ognuno di loro sarà Follower degli altri quindi sarà anche Following degli altri. Ogni volta che aumenteremo il numero degli utenti la base di dati sarà ovviamente svuotata prima di procedere con la nuova simulazione. Per attenuare eventuali errori di stima del tempo, i test verranno effettuati più volte con lo stesso numero di utenti così da ottenere un tempo[2] medio.

Numero di Utenti	**Inserimento Utenti**	**Inserimento Post**	**Inserimento Follower**	**Inserimento Following**	**Totale**
10	0,6359	0,4779	0,5332	0,5583	2,2053
50	1,7839	1,6128	1,3033	1,3129	6,0129
100	2,8199	2,6627	3,2312	2,7556	11,4694
500	14,1175	15,1188	76,7523	70,2787	176,2673
1000	26,1569	34,101	121,5642	120,7521	302,5742
5000	131,1341	169,1198	686,3461	679,2399	1665,8399

Tabella 4.1: Inserimento con un nodo fisico

Numero di Utenti	**Inserimento Utenti**	**Inserimento Post**	**Inserimento Follower**	**Inserimento Following**	**Totale**
10	0,7012	0,5315	0,6231	0,5269	2,3827
50	2,8021	2,1371	1,8573	2,4972	9,2937
100	5,6039	4,3704	2,9993	3,2271	16,2007
500	13,7258	14,2759	15,6123	23,9843	67,5983
1000	23,5004	31,2187	72,6903	65,6841	193,0935
5000	140,5878	145,6914	399,38	401,26	1086,9192

Tabella 4.2: Inserimento con due nodi fisici

Confrontando i dati forniti, si osserva, che come si poteva sospettare all'aumentare delle macchine fisiche in gioco, i tempi di risposta sono migliori.
Per pochi utenti la differenza e quasi nulla, mentre al crescere degli utenti il divario di prestazione è molto elevato, avere due o tre RaspberryPi non sembra cambiare molto le prestazioni, probabilmente servirebbe un numero molto più elevato di dispositivi per aumentare in modo significativo la prestazione. Il maggior carico di lavoro si ha nellinserimento dei follower e dei following, perché diventano dei vettori molto lungi se il numero di utenti è elevato e se ogni utente decide di

[2] i dati raccolti sono il tempo in secondi

Numero di Utenti	Inserimento Utenti	Inserimento Post	Inserimento Follower	Inserimento Following	Totale
10	0,6438	0,5974	0,5672	0,5071	2,3155
50	1,2766	1,4638	2,1545	2,2013	7,0962
100	2,6929	2,9303	3,8963	2,5181	12,0376
500	12,9847	13,9635	17,531	16,1162	60,5954
1000	22,3402	29,849	61,8889	67,9177	181,9958
5000	105,1742	124,6781	398,7889	399,7889	1028,4301

Tabella 4.3: Inserimento con tre nodi fisici

Numero di Utenti	Inserimento Utenti	Inserimento Post	Inserimento Follower	Inserimento Following	Totale
10	0,1331	0,1494	0,1217	0,1203	0,5245
50	0,5745	0,6875	0,5871	0,6854	2,5345
100	1,3515	2,2459	1,2801	1,3874	6,2649
500	12,6887	11,024	18,46	16,46	58,6327
1000	19,0472	20,48	25,9452	29,5342	95,0066
5000	89,8576	86,6875	262,2101	256,754	695,5092

Tabella 4.4: Inserimento con quattro nodi fisici

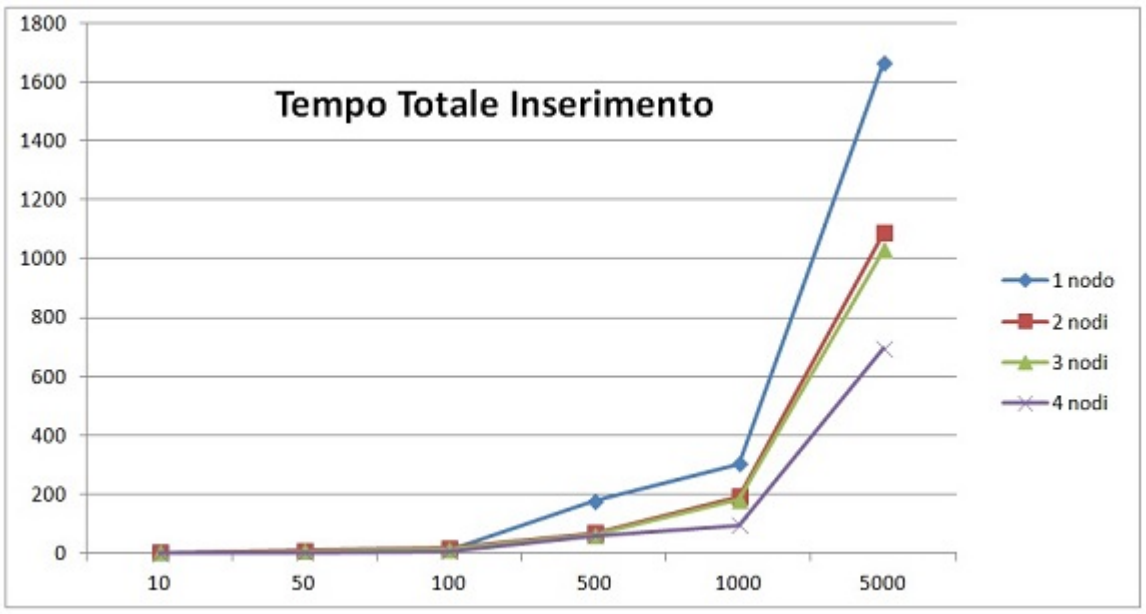

Figura 4.3: Grafico inserimento

seguire un numero altrettanto elevato di utenti. Infine laggiunta di un quarto nodo con capacità computazionali più elevate dei raspberry provoca certamente un aumento di prestazioni. Ora però è interessante vedere come rispondono questi vari modelli di anello, nel caso in cui N utenti vogliano caricare il proprio profilo,

cioè lo storico dei suoi commenti e visualizzare i commenti dei suoi following. Prima di analizzare i prossimi aspetti interessanti c'è un ultimo inserimento che abbiamo fatto per testare affondo Riak, cioè abbiamo provato a connettere uno dei quattro dispositivi fisici tramite WI-Fi, Basho sconsiglia questa pratica e i dati raccolti ne sono la prova.

Numero di Utenti	Inserimento Utenti	Inserimento Post	Inserimento Follower	Inserimento Following	Totale
10	0,3174	0,3067	0,2680	0,2994	1,1915
50	1,7031	1,4315	1,4495	1,4412	6,02525
100	2,5572	2,54	2,6150	2,7893	10,5015
500	14,3648	13,3287	15,3573	16,0177	59,06845
1000	25,2312	28,1367	36,6027	39,8081	129,7786
5000	126,0828	136,0572	599,8717	601,2891	1463,3137

Tabella 4.5: Inserimento con quattro nodi fisici, uno dei quali in Wi-Fi

Da segnalare inoltre che in alcuni dei test, ci sono stati degli errori di timeout, cioè durante l'inserimento in questa configurazione alcuni dati sono andati persi, il server non ha mai dato una risposta.

4.5 Le richieste di dati

Quando un utente entrerà nel nostro Riak-Twitter, gli apparirà una pagina contenente i suoi ultimi tweet, e alcuni dei tweet scritti dagli utenti che lui ha scelto di seguire. Quindi Riak dovrà restituire all'utente nel minor tempo possibile, la sua anagrafica, alcuni dei sui post e alcuni tweet degli altri utenti. Rieseguiremo i test cambiando, ovviamente ogni volta il numero di nodi in gioco.

4.5.1 Richieste con il metodo get

Come prima simulazione abbiamo provato ad eseguire nel modo più semplice possibile la creazione della pagina personale di N utenti, ponendo che la popolazione della nostra base di dati sia di 1000 utenti. In maniera sequenziale N utenti chiederanno alla nostra base di dati di caricare la loro anagrafica e 20 commenti in modo casuale tra i loro Following. Per prima cosa ogni utente otterrà la sua Anagrafica:

```
fetchedUser = UserBucket.get(idfetch)
```

Poi ogni utente dovrà ottenere la usa lista di Following

```
fetchedFollowing = FollowingBucket.get(idfetch)
```

Infine verranno scelti in maniera causale 20 utenti tra i following e verranno restituiti i loro commenti

```
fetchedTweet = TweetBucket.get(idfetch)
```

Questo tutta via è il peggior modo per estrarre dati, infatti non si sfrutta in alcun modo il fatto di avere un sistema distribuito su più nodi. Quando viene fatta una richiesta utilizzando il comando di get di Python, la richiesta viene inoltrata al nodo impostato inizialmente per l'avvio della comunicazione. Il nodo gestirà la richiesta cercando i dati richiesti in tutto l'anello e quando avrà finito invierà la risposta al host che ha formulato l'interrogazione. Quindi avere più nodi, non migliora le prestazioni anzi le peggiora. Quindi la simulazione è stata fatta solo con uno e due nodi, visto che gli altri casi erano analoghi.

Numero di Utenti collegati	**1 Raspberry-Pi**	**2 Rapberry-Pi**
1	0,0712	0,0654
10	1,5939	1,9256
50	36,7172	43,9117
100	156,8812	178,4139
200	641,1932	654,7896
500	4004,8599	4489,555

Tabella 4.6: Prestazioni con il metodo get, in un database una popolazione di 1000 utenti

4.5.2 Richieste con Map-Reduce

Il modo corretto per eseguire delle richieste all'anello è usare il meccanismo Map-Reduce. In questo modo le richieste fatte da un Host all'anello di Riak arriva al nodo indicato inizialmente tramite indirizzo ip, che prende il nome di Coordinatore. Il Coordinatore appena riceve la richiesta, la divide in parti più semplici. Poi il nodo spedirà queste richieste più semplici, ad alcuni nodi dell'anello attendendo da loro la risposta. Quando ha ottenuto le risposte dai nodi interrogati, il Coordinatore unirà le varie risposte e manderà il risultato finale all'host che ha formulato inizialmente la richiesta. Questo meccanismo permette di velocizzare molto i tempi di attesa delle richieste, perché sfrutta più nodi fisici per ottenere risposta alle richieste.

Per prima cosa dobbiamo impostare come eseguire la query, quindi passandogli myClient e poi nello stesso framento di codice passiamo anche quale utente prelevare dal Bucket User:

```
query=myClient.add("User",str(id))
```

Poi facciamo in modo che la nostra query prenda dal Bucket Tweet, tutti i commenti dell'utente che si sta collegando:

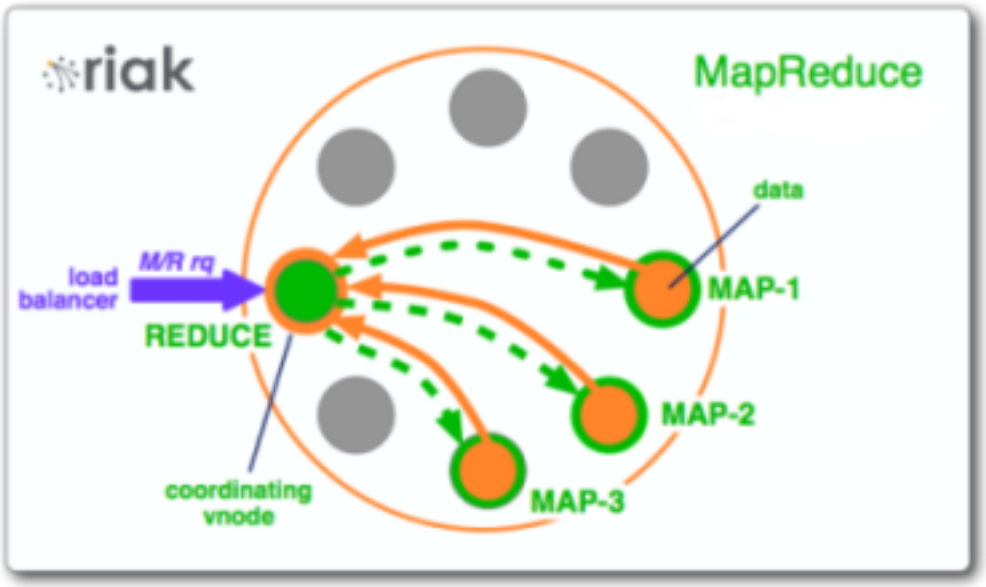

Figura 4.4: Schema Funzionamento Map Reduce

```
query.add("Tweet",str(id))
```

Inoltre per semplificare la simulazione poniamo di conoscere già in anticipo la lista di utenti alla quale l'utete vuole collegarsi, quindi per avere i loro commenti basterà:

```
query.add("Tweet",listIdFollowing)
```

Infine per eseguire la query basta usare:

```
results = query.run()
```

Numero di Utenti collegati	1 Nodo Fisico	2 Nodi Fisici	3 Nodi Fisici	4 Nodi Fisici
1	0,5241	0,4712	0,4566	0,4075
10	5,0233	4,6892	4,4026	4,1223
50	24,5763	23,6687	21,3548	19,0602
100	49,1988	46,0655	43,4183	38,6972
200	98,7598	90,8963	86,7885	76,3317
500	246,8709	229,9811	216,2297	190,8018

Tabella 4.7: Prestazioni con il metodo get, in un database una popolazione di 1000 utenti

Dal Grafico notiamo un notevole miglioramento, rispetto alla precedente simulazione senza Map-Reduce. All'aumentare dei nodi fisici le prestazioni migliorano, questo perchè Riak ha una buona scalabilità.

Nella nostra applicazione vi è la possibilità che ogni utente può mettere una priorità al proprio tweet, cioè ogni utente può inserire un valore da 1 a 10 il quale

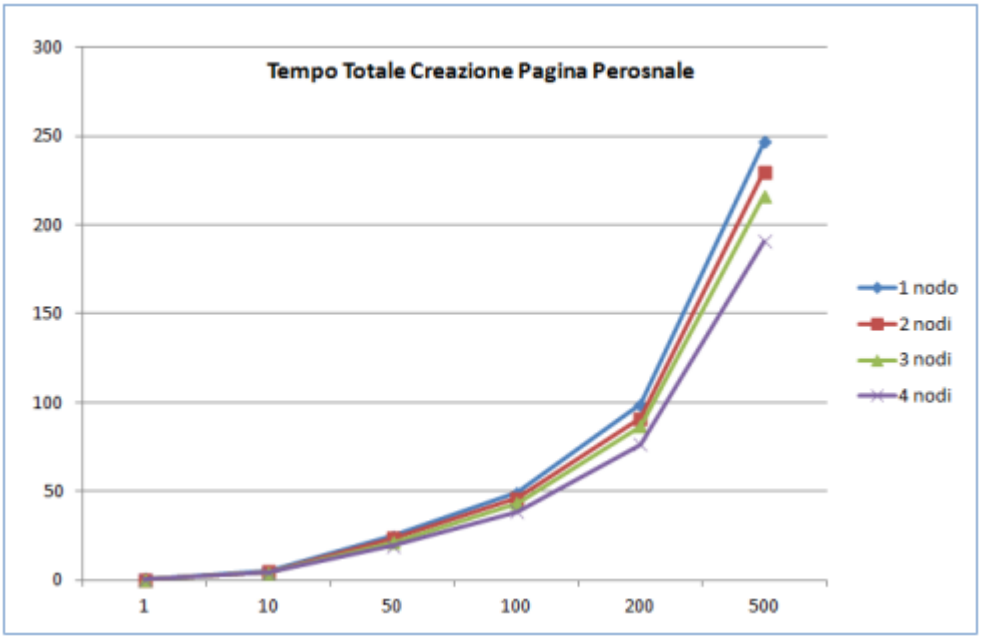

Figura 4.5: Grafico Prestazioni della Map Reduce

indica quanto l'utente ritiene importante quello che ha scritto (1 è la priorità massima). In questo modo ogni utente visualizzerà, i tweet dei suoi following con priorità più alta. l'oggetto del bucket Tweet diventerà così:

```
Tweet={'id':"Utente1",'value':"Hello World",''}
```

Mentre per visualizzare la propria pagina l'utente dovrà utilizzare una map-reduce più complessa. Per prima cosa passeremo impostiamo quali sono i suoi following:

```
query=myClinet.add("Tweet",listIdFollowing)
```

Ora con l'utilizzo di javascript creiamo una funzione, la quale nella fase di map seleziona da un singolo following solo il contenuto del suo commento e la sua priorità.

```
query.map("function(v) { var data = JSON.parse(v.values[0].data);
return [[data.like, data.value]]; }")
```

Ora utilizzeremo la fase di Reduce che ci permette prima di ordinare per priorità e infine di ottenere solo i primi venti risultati.

```
query.reduce("function(v) { return v.sort(); }")
query.query.reduce_limit(20)
```

Questa è una richiesta abbastanza complessa, ma non è di difficile comprensione per un conoscitore dei linguaggi orientati agli oggetti, per questo motivo la Map-Reduce è molto apprezzata dai programmatori.

Capitolo 5

Conclusioni

L'argomento principale della tesi ha riguardato lo studio di una base di dati non relazionale e anche una sua semplice implementazione di un famoso social network. Questo tipo di base di dati è sempre più diffusa, perchè permette una buona scalabilità e quindi buone prestazioni nella manipolazione di grosse quantità di dati. Una base di dati come Riak è fondamentale per un social network e anche per molti negozi on-line (come Ideeli e Copious) sopratutto per la tolleranza ai guasti. La tolleranza ai guasti, in Riak, è garantita dalla replicazione dei dati, i quali sono distribuiti su molti nodi fisici distribuiti ad anello. Se un nodo fisico subisce un guasto, Riak continua a funzionare, garantendo la continuazione dei suoi servizi. La possibilità di installare Riak in piccoli dispositivi, come i Raspberry-Pi, permette di abbassare i costi per l'acquisto dei nodi fisici su cui installare Riak. La presenza di un sistema distribuito, inoltre permette di migliorare le prestazioni quando si effettua una interrogazione, grazie alla funzione di map-reduce. Grazie alla map-reduce un solo nodo non deve farsi carico dell'intera interrogazione e trovare tutti i dati che servono lungo l'anello, ma tutti i nodi fisici possono contribuire a risolvere l'interrogazione.

Nel futuro Riak, sarà sempre più usato, anche perchè i social network stanno diventando sempre più di uso quotidiano. Anche i negozi on-line stanno aumentando e anche il numero dei loro utilizzatori. Anche nel settore delle applicazioni per smartphone alcune note aziende (Rovio) hanno deciso di utilizzarlo per i loro servizi. Con il servizio Riak CS (Cloud Storage), un software open source per l'archiviazione di dati basato su Riak, si puó affittare uno spazio di archiviazione dati per uso personale o come base per servizi e applicazioni, magari di una moderna start-up con idee legate al mondo del web.

Bibliografia

[1] Ramez Elmasri, Shamkant B. Navathe : Sistemi di Basi di Dati. USA, (2011), (Pearson, Italia, 2011)

[2] http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/, Ilya Katsov

[3] http://www.linqitalia.com/articoli/nosql/introduzione-database-nosql.aspx, Fabrizio Iezzoni

[4] http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Philosophy%20of%20NoSQL, Carlo Strozzi

[5] http://docs.basho.com/riak/latest/theory/why-riak/

[6] http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[7] http://www.raspbian.org/

[8] Eric Redmond, Jim R. Wilson: Seven Databases in Seven Weeks. Pragmatic Bookshelf (May 18, 2012)

[9] http://www.zeerko.com/guida-twitter/cose-twitter/

[10] http://basho.com/building-a-riak-cluster-on-raspberry-pi/,Eric Redmond

[11] http://basho.github.io/riak-python-client/, Documentazione Client Python-Riak