Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

Software per l'Analisi di Big Data su più livelli e sua Applicazione a Tracce di Google

Tesi di Laurea in Sistemi Complessi

Relatore:
Chiar.mo Prof.
Ozalp Babaoglu

Correlatore:
Chiar.mo Prof.
Moreno Marzolla

Presentata da:
Alkida Balliu

Sessione 3
2012/2013

Alla mia famiglia, amici e colleghi...

Introduzione

"Nel giro di due giorni produciamo la stessa quantità di dati generati dagli albori della civiltà sino al 2003", sostiene nel 2010 Eric Schmidt. Secondo IBM, entro il 2020 il mondo avrà generato una massa di dati dell'ordine dei 40 zettabyte (10^{21} Byte).
Il mondo odierno è sommerso dai Big Data, provenienti dalle più svariate fonti. Seppur ad una prima occhiata questa mole di dati sembri caotica e confusa, in essa sono contenute informazioni per svolgere studi di diversi tipi.

Un'analisi accurata dei Big Data è molto importante per estrarre informazioni utili, in particolar modo se il sistema studiato è di difficile comprensione, come il caso dei sistemi complessi. I data center sono un esempio calzante di un sistema complesso che affronta la gestione dei Big Data. Essi si scontrano spesso con situazioni imprevedibili che portano il sistema in uno stato indesiderato. Pertanto, un'attenta analisi dei dati prodotti dai data center, sarebbe di rilevante aiuto in uno studio di previsione degli stati critici del sistema.

Il processo che consiste nell'estrapolare informazioni utili dai dati viene chiamato *Data Analytics*. Vi sono diversi software volti ad effettuare analisi di dati, come ad esempio *WEKA* e *Oracle Data Mining*, che per estrarre informazioni dai dati utilizzano tecniche di *Data Mining*. Alcuni di essi però (come ad esempio WEKA, che lavora su RAM) non sono adatti per lo studio di un grande volume di dati, quali i Big Data.
Oltre a tali software, vi sono numerosi linguaggi ed ambienti di sviluppo, come ad esempio R, contenenti funzionalità volte a studi statistici, ampia-

mente utilizzati nelle analisi dei dati. Essi però non sono di facile padronanza e richiedono che l'utente abbia delle capacità di programmazione.
Un framework largamente utilizzato quando si ha a che fare con la gestione dei Big Data è *Hadoop/MapReduce*. Esso permette di processare in maniera distribuita una grossa quantità di dati, rendendo possibile la loro veloce manipolazione.

L'obiettivo di questo lavoro è quello di creare del software che metta a disposizione strumenti per effettuare una completa analisi dei Big Data, in maniera automatica. Pertanto è stato scelto di implementarvi il linguaggio e le funzionalità di R, nonché la possibilità di utilizzare Hadoop/MapReduce, mettendo così a disposizione dei contenitori dati scalabili.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'arte

1.1 *Big Data*

1.1.1 Introduzione ai Big Data

Con l'avanzare della tecnologia, si sente sempre di più parlare di *Big Data*. Le definizioni proposte riguardo questo concetto sono molteplici, ma generalmente con il termine Big Data si intende una grande mole di dati dai quali risulta difficile estrapolare informazioni utili.
I Big Data si distinguono per le seguenti caratteristiche [1]:

- *Volume*: si generano dati dell'ordine dei Terabytes (TB) al giorno;
- *Varietà*: i dati sono di tutti i tipi, di diversi formati;
- *Velocità*: IBM stima che più del 90% dei dati esistenti ai giorni d'oggi sono stati creati soltanto negli ultimi 2 anni.

Eric Schmidt, presidente del consiglio d'amministrazione di Google, così affermava nel 2010: "ogni due giorni creiamo una quantità di dati pari al numero totale dei dati generati dagli albori della civiltà sino al 2003"[2]. Per avere un'idea generale sui numeri che descrivono i Big Data, si possono osservare i seguenti dati facenti riferimento all'anno 2010 [3]:

- il numero di messaggi inviati tramite Twitter (tweet) arriva a 340 milioni al giorno;
- Google riceve più di 2 milioni di *query* di ricerca al minuto;
- l'Organizzazione Europea per la Ricerca Nucleare (CERN) genera 40 Terabyte al secondo;
- i sensori di un *jet* producono 10 Terabyte di dati ogni 30 minuti.

1.1.2 Gestione e analisi dei Big Data nei sistemi complessi: i data center

I sistemi sono entità costruite da componenti (agenti) ben definiti che comunicano fra loro in maniera dinamica. In un semplice sistema gli agenti sono relativamente pochi, le loro interazioni sono regolari ed è facile predire i loro comportamenti e, di conseguenza, quello dell'intero sistema. Quando invece gli agenti sono tanti e la loro interazione rende il comportamento globale imprevedibile e difficile da descrivere, questo sistema è detto *Sistema Complesso*. Le comunicazioni fra gli agenti in questo tipo di sistemi è tale che anche un piccolo cambiamento influisce nel comportamento dell'intero sistema.

Un'altra maniera di vedere un sistema complesso lo riscontriamo nei sistemi cosiddetti *caotici*, i quali dipendono sensibilmente dalle condizioni iniziali. Questi sistemi sono affetti dal famoso *effetto farfalla*, menzionato anche da Alan Turing nel suo saggio *Macchine calcolatrici ed intelligenza*, nella sezione *Universalità dei calcolatori numerici* nel 1950: "il sistema dell'universo come un tutto è tale che errori molto piccoli nelle condizioni iniziali possono avere effetti disastrosi in un momento successivo"[4].
I sistemi complessi sono presenti non solo in ambito informatico, ma li troviamo anche in biologia, economia, fisica, natura, etc...

Un ulteriore esempio di sistema complesso è quello dei *data center* (figura 1.1): un data center è un sistema centralizzato di computer collegati fra loro

Figura 1.1: Data center di Google

con il fine di collezionare, maneggiare e divulgare i dati. Viene chiamato *cluster* di un data center un insieme di macchine connesse fra loro e che si comportano come un unico sistema. Si definisce invece *cella* l'insieme di macchine, comunemente facenti parte dello stesso cluster, che condividono lo stesso sistema di gestione che si occupa di allocare il lavoro alle macchine.

In quanto sistemi complessi, spesso i data center si sono scontrati con situazioni non predicibili. Molti colossi come Facebook, Twitter o Amazon si sono trovati nella situazione in cui non sono riusciti a garantire la continuità dei loro servizi per diverse ore consecutive. Spesso le motivazioni del *downtime* (tempo in cui i servizi non sono disponibili) non sono divulgate, ma qualche volta è successo che i grandi colossi decidessero di divulgarne le cause: il 24 dicembre del 2012 una parte degli utenti di Amazon ha subito 12 ore di downtime per via di una modifica apparentemente innocua effettuata da un operatore [5]. Dopo questo episodio, Amazon ha risolto il problema ed è sicura di non riscontrarlo più, ma in generale non sa a priori quali modifiche in futuro potrebbero creare altri tipi di problemi.

I data center quindi costituiscono un punto di incontro fra i Big Data e i sistemi complessi. Giorno dopo giorno essi si trovano a scontrarsi con

nuove sfide, in quanto devono gestire un grande volume ed un'ampia varietà di dati.
Fare un'analisi accurata dei Big Data è molto importante, in quanto può rivelarsi di aiuto per poter identificare in anticipo stati indesiderati in sistemi complessi, come ad esempio i data center. Tornando all'esempio di Amazon, una mirata analisi predittiva dei dati, poteva rivelarsi d'aiuto e forse in grado di indicare quella specifica modifica come "pericolosa" per l'intero sistema.

Pertanto l'obbiettivo di questo lavoro è quello di realizzare un software efficace di analisi dei Big Data, con il fine di individuare stati sicuri, pre-critici o critici nei data center. Le tracce dei dati sui quali effettuare analisi e validare il software sono provenienti dai cluster di Google, rilasciate nel 2011 dallo stesso ente.

1.2 Tracce di Google

Le tracce rilasciate da Google sono provenienti da circa 12500 macchine diverse e riguardano un periodo di 29 giorni[6][7]. L'esatta configurazione delle macchine non è stata resa nota, pertanto le misure riguardanti la CPU e la memoria sono state normalizzate rispetto alla macchina con capacità maggiore. Un riassunto delle caratteristiche delle tracce viene mostrato in tabella 1.1.

Caratteristiche delle tracce	Valori
Intervallo di tempo delle tracce	29 giorni
Job in esecuzione	650K
Numero totale di utenti	925
Task inviati	25M
Eventi dello scheduler	143M
Utilizzo delle risorse	1233M
Dimensioni compresse	39GB

Tabella 1.1: Riassunto delle tracce rilasciate da Google

L'intero insieme di dati è stato suddiviso in sei tabelle diverse:

- *Eventi delle macchine*: tre tipi di eventi
  - Una macchina viene messa a disposizione di un cluster;
  - Una macchina è stata rimossa dal cluster;
  - Le risorse disponibili in una macchina sono state modificate.
- *Attributi delle macchine*: gli attributi delle macchine sono descritti da coppie $< chiave, valore >$ tramite le quali vengono descritte le proprietà di una determinata macchina (come ad esempio la versione del kernel);
- *Eventi dei job*: descrive il ciclo di vita di un job;
- *Eventi dei task*: delinea il ciclo di vita di un task;
- *Vincoli dei task*: vi sono diversi vincoli che un task può avere, rappresentati da stringhe offuscate che indicano l'attributo della macchina e del suo valore, ed un operatore di confronto;
- *Utilizzo di risorse da parte dei task*.

I job e task attraversano diversi stati durante il loro ciclo di vita. Le transizioni da uno stato all'altro vengono descritte dagli eventi (rappresentati da valori che vanno da 0 a 8) dei vari job e task. Una versione semplificata del ciclo di vita di questi ultimi è descritta in figura 1.2.

I dati sono rappresentati da tabelle in formato CSV; le righe sono disposte in ordine di marca temporale e quest'ultima si trova sempre nella prima colonna.

1.3 Data Analytics

Seppur nell'uso comune vengono spesso interscambiati, vi è una grande differenza fra i termini *informazione* e *dato*: i dati rappresentano eventi,

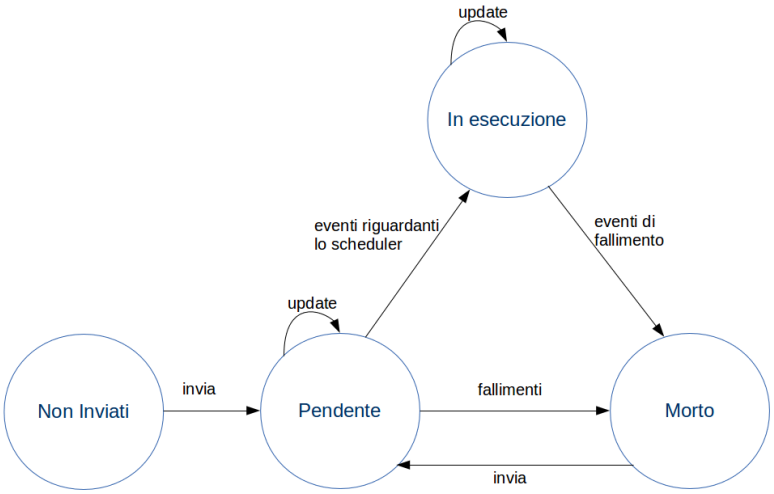

Figura 1.2: Stati di transizione per i job e i task

fatti ai quali non è stata data alcuna interpretazione; invece l'informazione è quello che si ricava analizzando i dati dando a loro un significato.

Questo processo durante il quale si cerca di estrapolare informazioni dai dati, viene chiamato *Data Analytics*.

1.3.1 Data Mining

Il termine *Data Mining* deriva dalla lingua inglese e vuole significare *estrarre informazioni utili dai dati*. Il Data Mining quindi si occupa di scoprire informazioni notevoli da un grande volume di dati, solitamente immagazzinati in un database, cercando di svelare dei pattern interessanti. L'architettura di un tipico sistema di Data Mining è rappresentata nella Figura 1.3, dove si possono osservare i seguenti componenti:

- *Tipi di fonti*: i dati possono essere collezionati da diversi tipi di *repository*; in questa fase è possibile effettuare una prima analisi sui dati mediante tecniche di integrazione e di pulizia;

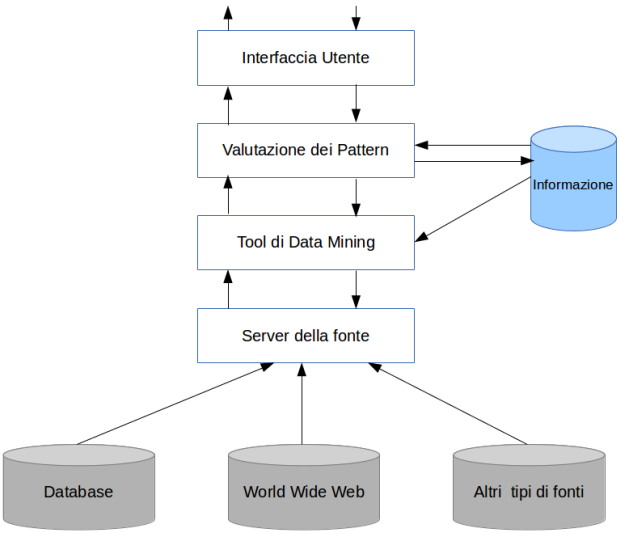

Figura 1.3: Architettura di un tipico sistema di Data Mining.

- *Server della fonte*: in questa fase avviene il prelievo dei dati rilevanti; essi dipendono dal tipo di operazione di Data Mining che si vuole eseguire;

- *Tool di Data Mining*: consiste in un insieme di funzionalità volte ad effettuare Data Mining;

- *Valutazione dei Pattern*: questa entità si occupa di individuare se i dati presentano dei pattern interessanti;

- *Informazione*: questo componente contiene informazioni sui dati (come ad esempio una classificazione gerarchica) e comunica con le entità *Tool di Data Mining* e *Valutazione dei pattern* per riuscire in una buona analisi dei dati;

- *Interfaccia Utente*: utile per una facile comunicazione fra l'utente e il sistema di Data Mining.

I compiti di un sistema di Data Mining si possono raccogliere in due macro gruppi: descrittivi e di previsione. I primi si occupano di analizzare le proprietà e caratteristiche principali dei dati, mentre invece i secondi, datogli un insieme di dati, vi applicano una mirata analisi per effettuare previsioni. In generale possiamo distinguere 6 funzionalità del Data Mining:

- *Descrizione*: sono analisi che cercano di descrivere i dati tramite riassunti e confronti in modo da caratterizzarli;
- *Associazioni e Correlazioni*: in questa analisi si cercano pattern frequenti nei dati;
- *Classificazione e Previsione*: per riuscire in questo tipo di studio, si fa utilizzo delle *reti neurali*, analisi di *regressione* e i cosiddetti *alberi di scelte*;
- *Analisi di Classificazione (Clustering)*: si individuano elementi che sono accomunati da qualche caratteristica formando così un insieme che li contiene;
- *Analisi di Anomalie*: si svelano dati che risultano anomali rispetto all'insieme intero di dati;
- *Analisi di Evoluzione*: modella e descrive regolarità nei dati che cambiano nel tempo.

Le tecniche di Data Mining vengono adottate in diversi campi: statistica, *machine learning*, scienze della formazione, economia, medicina e altre discipline. Fra alcuni dei software di Data Mining più famosi possiamo menzionare *WEKA* [8], *Oracle Data Mining (ODM)* [9]; *IBM Intelligent Miner* [10], etc...

Nonostante sia molto utilizzato, nel nostro lavoro non si è fatto uso di Data Mining, in quanto questo tipo di analisi non entrava nel nostro obiettivo primario, ed alcuni tool, come ad esempio *WEKA*, non sono adatti per lavorare con i Big Data.

1.3.2 R

Introduzione ad R

R è un linguaggio ed un ambiente di sviluppo rilasciato sotto licenza *GPL*, dove sono presenti numerose funzionalità utili agli studi statistici [11, 12]. Esso è un linguaggio *case sensitive*; la visione che si ha quando si utilizza R è quella di un *prompt* solitamente rappresentato con il segno '>', tuttavia modificabile tramite un semplice comando. Più funzioni possono essere poste su diverse righe oppure separate da punto e virgola (';'). I commenti sono preceduti dal simbolo '#'.
Per informazioni e approfondimenti sulle funzionalità di un determinato comando vi è la possibilità di invocare la funzione *help*.

La rapida espansione di R ha portato ad un ampio numero di pacchetti esistenti oggigiorno, il che mostra come questo linguaggio sia facilmente estendibile. Fra le altre, in R sono incluse:

- funzionalità per una facile manipolazione dei dati;
- un insieme di operatori per maneggiare agevolmente vettori e matrici;
- un'ampia collezione di strumenti per l'analisi dei dati;
- visualizzazioni grafiche di ogni tipo;
- un semplice linguaggio di programmazione che include cicli, condizioni, funzioni ricorsive definite dagli utenti e utilità di input/output.

A differenza di altri linguaggi adatti a fare statistica, le analisi in R sono fatte per passi: come nei linguaggi classici, R permette di salvare il risultato intermedio in variabili, in modo da poterle utilizzare in futuri passaggi.

Statistica con R

Analisi statistiche si possono facilmente realizzare utilizzando R [13]. Pertanto, in questa sezione vedremo alcuni esempi delle principali analisi statistiche. Più in dettaglio si esamineranno i seguenti aspetti:

- Regressione lineare
  - Funzione *lm()*.
- Regressione non lineare
  - Funzione *nls()*;
- Funzioni di ripartizione note
  - Normale;
  - Poisson;
  - Esponenziale;
  - Gamma;
  - Funzione *fitdist()* del pacchetto *fitdistrplus*.

La ***regressione lineare*** viene utilizzata per mettere in relazione una variabile dipendente con un insieme di variabili indipendenti. Nel caso di regressione lineare semplice, la funzione in grado di interpretare questi tipi di dati ha la forma $y = a+bx$ dove y è la variabile dipendente e x rappresenta i vari valori in input. Fra tutte le funzioni, viene scelta quella che minimizza l'errore[1]: $\sum_{i=1}^{n}(y_i - (a + bx_i))^2$ deve essere minima.
Per calcolare tale funzione con R si fa uso del comando *lm()* (Linear Model), alla quale vengono forniti in input una formula dipendente dai nomi delle colonne da porre in relazione, e la variabile contenente i dati stessi (per più informazioni si digiti "?lm"). Ad esempio, se volessimo porre in relazione le colonne *peso* e *altezza* presenti in una tabella chiamata *mieiDati* mediante la funzione lineare $y = a + bx$ che meglio spiega il loro andamento, in R scriveremmo:

```
> risultato_lineare <- lm(altezza~peso, mieiDati)
```

[1]Errore standard: somma dei quadrati delle differenze tra i valori osservati e quelli teorici

La funzione *lm()* è inoltre in grado di approssimare un insieme di dati con altri tipi di regressione lineare, come ad esempio quello polinomiale, esponenziale, logaritmico, etc...

```
# Approssimazione con un polinomio di secondo grado
 > lm(y ~ I(x^2) + I(x), data=mydata)
```

Per approssimare un insieme di dati mediante una qualsiasi funzione si adotta la ***regressione non lineare***, che utilizza metodi iterativi per minimizzare la funzione di errore (non è garantito che venga trovata la soluzione ottima). In R vi è un comando chiamato *nls()* (NonLinear Least Squares), il quale prende in input una funzione qualsiasi (diversamente dal caso di *lm()* che richiedeva una formula), un insieme di dati ed una stima dei valori iniziali.
Osserviamo come sia anche qui possibile effettuare della regressione mediante un polinomio, ma a differenza del caso precedente, la funzione *nls()* non è necessario che conosca l'espressione della formula, che potrebbe essere arbitrariamente complessa, contenendo ad esempio dei cicli; verrà utilizzato un metodo iterativo per valutare la formula nei punti d'interesse:

```
# Viene creato un campione di esempio
 > x <- c(0,1,2,3,4,5)
 > y <- c(3.1,-1.1,-3.1,-3,-1,3)
 > mydata <- as.data.frame(cbind(x, y))


# Funzione da utilizzare per approssimare i dati
  > f <- function(x,a,b,c){ a*x^2+b*x+c }


# Si specifica una stima dei parametri iniziali
  > stima <- list(a = 1, b = 1, c = 1)
  > nls(y ~ f(x,a,b,c), data = mydata, start = stima)
```

R offre un'ampia scelta di comandi di ***funzioni di ripartizione note***. Tali funzioni sono importanti, specialmente se esibite graficamente, in quan-

to sono di rilevante aiuto in uno studio dove è necessaria una simulazione di campionamenti che esibiscono una certa distribuzione nota. Osserviamo alcune di esse:

```
# Distribuzione normale (o gaussiana):
# 100 campioni, media = 20, deviazione standard pari a 4

  > dist_norm <- rnorm(n=100, m=20, sd=4)

# Distribuzione di Poisson:
# esempio con 50 campioni e valore del LAMBDA uguale a 25
  > dist_pois <- rpois(n=50, lambda=25)

# Distribuzione esponenziale: esempio con 30 campioni e
# tasso di decadimento pari ad 1
  > dist_exp <- rexp(n=30, rate=1)

# Distribuzione gamma: esempio con 100 campioni,
# LAMBDA = 0.2 (parametro di scala),
# alpha = 2.7 (parametro di forma)
  > dist_gamma <- rgamma(n=100, rate=0.2, scale=2.7)
```

Nel caso in cui si abbia un insieme di valori di cui si vuole approssimarne la densità, è possibile procedere in due modi differenti. Il primo consiste nel calcolare la distribuzione dei dati ed applicare su di essa il comando precedentemente descritto *nls()*, specificando la funzione di distribuzione con cui approssimare i dati ed una buona stima di parametri iniziali. Il secondo approccio consiste nell'utilizzare il comando *fitdist* del pacchetto *fitdistrplus*, che automatizza gran parte di questo lavoro.
Tale funzione tenta di approssimare i dati mediante distribuzioni note, trovando automaticamente i giusti parametri (quelli che minimizzano l'errore standard) per meglio modellare l'andamento dei dati. Infatti, per analizzare

se un campione di dati, che chiamiamo *mydata*, mostra un andamento di tipo gaussiano, è possibile scrivere:

```
# Si carica la libreria fitdistrplus
  > library (fitdistrplus)

# Si generano dati di esempio distribuiti secondo una gaussiana
# con media 8
  > mydata <- rnorm(100,8)

# Si applica il comando fitdist; se l'errore standard sarà
# basso allora deduciamo che i dati hanno una
# distribuzione normale.
  > fitdist(mydata, "norm")
```

Quello che risulterebbe dalla visualizzazione grafica del risultato, si può notare in figura 1.4

In maniera simile possiamo interrogare i dati per analizzare se esibiscono una distribuzione di Poisson, lognormale, gamma, esponenziale o altre, scrivendo al posto di "*norm*" rispettivamente *pois*, *lnorm*, *gamma*, *exp*.

Per avere un'idea generale sulle distribuzioni note interrogabili tramite R, si consulti la tabella 1.2.

Distribuzione	Nome in R	Argomenti aggiuntivi
beta	*beta*	*shape1, shape2, ncp*
binomiale	*binom*	*size, prob*
geometrica	*geom*	*prob*
log-normale	*lnorm*	*meanlog, sdlog*
logistica	*logis*	*location, scale*
t-student	*t*	*df, ncp*
Weibull	*weibull*	*shape, scale*

Tabella 1.2: Distribuzioni note in R

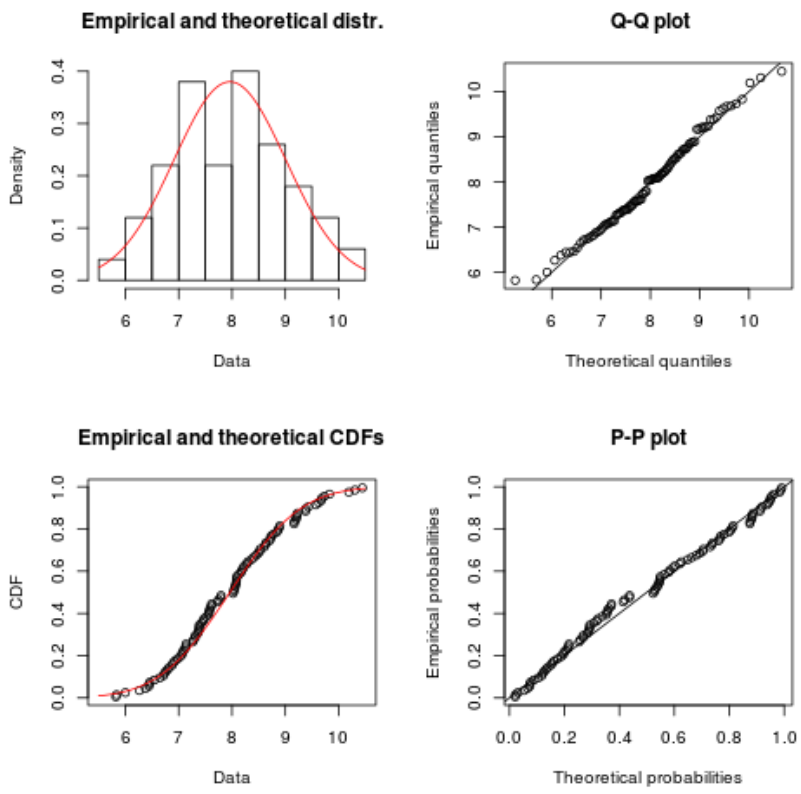

Figura 1.4: Esempio dell'applicazione della funzione *fitdist()*

Rappresentazione grafica con R

La rappresentazione grafica è molto importante per effettuare un'analisi adeguata dei dati in quanto è uno strumento con il quale si può ottenere un *feedback* visivo del proprio lavoro. R permette di raffigurare i dati tramite grafici di diversi tipi:

- *Plot*: per disegnare i dati come punti in un piano cartesiano si utilizza la funzione *plot()*;
- *Più funzioni nello stesso grafico*: se eseguito dopo un *plot()*, il comando *lines* permette di visualizzare nello stesso grafico la funzione specificata (figura 1.5);

```
# Generiamo due insiemi di punti distribuiti
# su due diverse parabole
 > x <- sort(runif(100,0,5))
 > y <- function(x){ x^2 -5 * x + 3 }

 > x2 <- sort(runif(100,0,5))
 > y2 <- function(x){ -x^2 + 4 * x - 2 }

# Si esegue "plot()" sul primo insieme di dati
 > plot(x, y(x), type="o", col="blue")

# Si aggiunge allo stesso grafico la funzione
# rappresentante il secondo insieme di dati
 > lines(x2, y2(x2), type="o", lty=2, col="red")
```

- *Istogrammi*: per la creazione di istogrammi, in R si utilizza la funzione *hist()* che prende in input i dati desiderati;
- *Funzione di densità*: il comando *density()* genera i valori della funzione di densità, visualizzabile tramite il solito comando *plot()*;

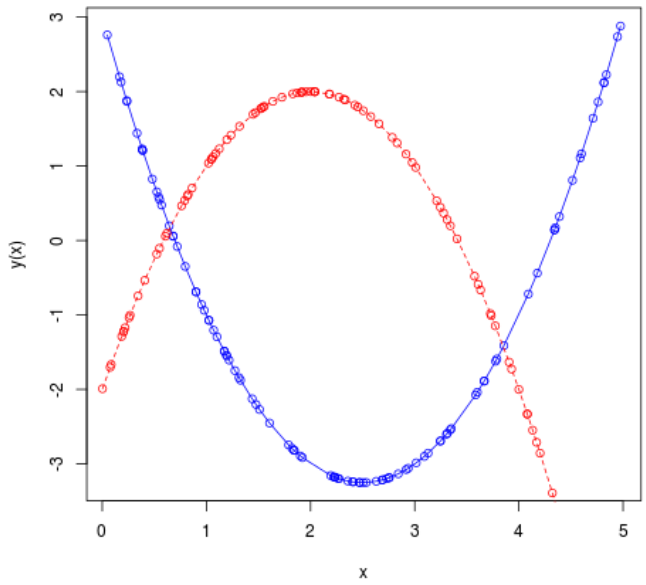

Figura 1.5: Due plot nello stesso grafico

```
# Si visualizza la funzione di densità dell'insieme
# "dataset"
 > plot(density(dataset), col="blue")
```

- *Funzione di ripartizione cumulativa empirica*: si può calcolare mediante il comando *ecdf()* (figura 1.6);

```
# Visualizzazione grafica della funzione di
# ripartizione cumulativa empirica dell'insieme
# "dataset"
 > plot(ecdf(dataset))
```

- *Diagramma circolare*: i cosiddetti grafici a torta sono realizzabile tramite il comando *pie()*;

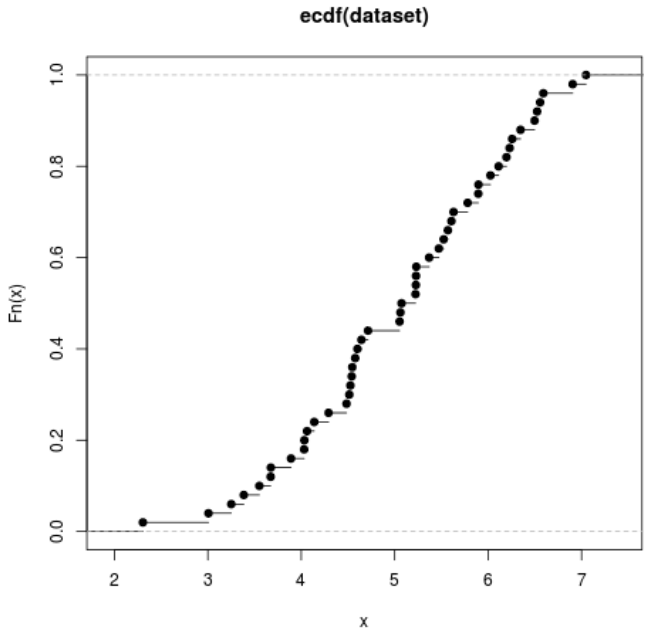

Figura 1.6: Funzione di ripartizione dell'insieme dataset

Nonostante R permetta di eseguire una vasta serie di operazioni statistiche, averne piena padronanza non è immediato e richiede abilità di programmazione.

1.3.3 Apache Hadoop MapReduce

Estrarre informazioni utili dai Big Data, utilizzando software di analisi su una singola macchina, potrebbe risultare un'impresa impossibile, in quanto una semplice operazione su un insieme così ampio di dati potrebbe richiedere interi giorni.
Il software *Apache Hadoop* [14] è un *framework* scritto in linguaggio *Java*, che permette di processare grandi quantità di dati in modo distribuito, dividendo il lavoro fra numerose macchine facenti parte di un cluster. Esso include i seguenti moduli:

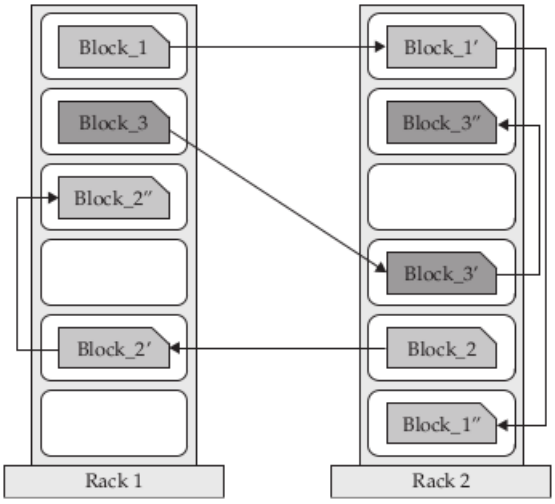

Figura 1.7: Esempio di come sono scritti i blocchi nell'HDFS. Si noti come ogni blocco viene scritto tre volte (opzione di default). Per motivi di ridondanza, ogni blocco è scritto in almeno un *server rack* diverso.

Hadoop Common

Tale modulo include le *utility* generali che supportano gli altri moduli.

Hadoop Distributed File System (HDFS)

HDFS è un *file system* distribuito altamente tollerante ai guasti sul quale si appoggia Hadoop. I file vengono suddivisi in *blocchi*. Essi sono replicati per rendere il sistema tollerante ai guasti (figura 1.7).

In questo modo i blocchi vengono distribuiti in maniera tale che le funzioni di *map* e *reduce* processino contemporaneamente un piccolo insieme di dati per macchina. Le dimensioni dei blocchi e il numero di repliche sono configurabili per ogni file (la dimensione di *default* di un blocco in Apache Hadoop è di 64MB). Pertanto le principali mete di HDFS sono:

- Alta tolleranza ai guasti e veloce ripresa;
- Dato che le applicazioni che vengono eseguite su questo *file system* devono accedere ai dati, si cerca di dare enfasi alla velocità di accesso

al disco, mettendo in secondo piano i problemi legati alla latenza;

- Poter gestire una vasta quantità di dati ed essere *multipiattaforma*.

In tabella 1.3 si possono consultare alcuni esempi di comandi *shell* di HDFS.

Comando	Descrizione
cat	Copia il file sullo standard output (*stdout*)
chmod	Cambia i permessi di lettura/scrittura per un dato file (o un insieme di file)
chown	Cambia il proprietario di un file (o un insieme di file)
copyFromLocal	Copia un file dal file system locale all'HDFS
copyToLocal	Copia un file da HDFS al file system locale
cp	Copia i file di HDFS da una cartella all'altra
ls	Mostra una lista dei file appartenenti ad una cartella
mkdir	Crea una cartella in HDFS
mv	Sposta i file da una cartella all'altra

Tabella 1.3: Esempi di comandi shell di HDFS

Hadoop YARN

Consiste in un *framework* che si occupa di schedulare i job e gestire le risorse dei cluster.

Hadoop MapReduce

È un sistema basato su YARN che si occupa di processare parallelamente i Big Data. Questo *software framework* opera su coppie $< chiave, valore >$ ed è stato progettato per facilitare la scrittura delle applicazioni. Esse si occuperanno di specificare principalmente due funzioni: *Mapper* e *Reducer*.

La prima sarà responsabile di calcolare tutte le chiavi e relativi valori di un frammento; mentre la seconda si occuperà di calcolare il risultato aggregato di ogni insieme di valori con la stessa chiave.

Per meglio spiegare il procedimento prendiamo in esempio una funzione, chiamata *WordCount*, alla quale viene dato in input un *file* di testo e fornisce in output il numero di volte che compare ogni parola. Per fare ciò con *MapReduce*, ogni riga viene data in pasto ad una funzione di tipo *Mapper*, che genererà una coppia $< chiave, valore >$ per ogni parola della riga (nel caso specifico, la coppia sarà della forma $< parola, 1 >$). Successivamente, le funzioni di tipo *Reducer* si occuperanno di sommare i valori della stessa chiave procurando il risultato atteso. In questa maniera, eseguendo la funzione in modo distribuito si riesce ad ottenere l'esito in poco tempo.
Una delle modalità possibili per implementare questa operazione è la seguente:

```
map <- function(k,lines) {
  words.list <- strsplit(lines, '\\s')
  words <- unlist(words.list)
  return(keyval(words, 1))
}
reduce <- function(word, counts) {
  keyval(word, sum(counts))
}
wordcount <- function (input, output=NULL) {
  mapreduce(input=input, output=output,
            input.format="text",map=map,reduce=reduce)
}
```

Hadoop è molto utilizzato quando si devono gestire dati delle dimensioni dei Big Data e, seppur necessario, da solo non è sufficiente per fare un'analisi statistica dei dati, in quanto il suo linguaggio risulta essere scomodo nel caso in cui si vogliano eseguire operazioni molto diverse dall'aggregazione.

Capitolo 2

Big Data Analyzer (*BiDAl*)

In questo lavoro è stato implementato un programma (BiDAL) che permette di effettuare un'accurata analisi di dati. Tale software è stato pensato per poter gestire anche dati della grandezza dei Big Data.
Per realizzare al meglio questo obbiettivo, è stato considerato di essenziale importanza utilizzare R e Hadoop/MapReduce nel nostro software.

2.1 Introduzione a BiDAl

Big Data Analyzer (BiDAl) è un programma scritto in *Java* per effettuare analisi sui Big Data. Esso permette di esaminare l'input in modo automatico, indipendentemente dal tipo di contenitore dei dati (*storage*).
Nel programma sono presenti 3 tipi di storage differenti:

- Bash: utilizza il file system normale per salvare i file e comandi bash per manipolarli; tale storage è stato implementato con l'intento di mostrare l'estendibilità del software piuttosto che per la sua effettiva utilità.
- SQLite: adopera SQLite come file system, *RSQLite* come linguaggio per manipolare tabelle, R per fare analisi sui dati.
- Hadoop: fa uso del file system HDFS e come linguaggio adopera RHadoop.

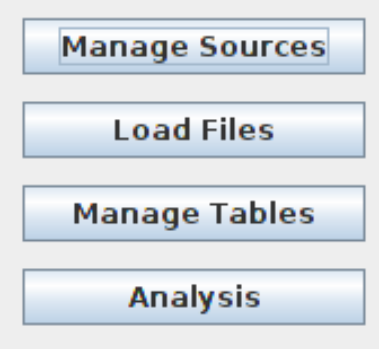

Figura 2.1: Finestra principale del software

Graficamente, la prima visuale che BiDAl offre consiste in una finestra con 4 collegamenti ad altrettante interfacce (figura 2.1).

Manage Sources:
Questo ambiente permette di aprire un file (o una cartella) rappresentante un contenitore di dati. Inoltre, viene offerta la possibilità di creare un file vuoto (o una cartella vuota) contenente un determinato tipo di storage.
Una volta caricati i contenitori dati d'interesse, si prosegue con la seconda interfaccia.

Load Files:
Questo collegamento porta ad un ambiente in cui è possibile selezionare file e caricarli in tabelle facenti parte dello storage desiderato.
Successivamente, dopo aver caricato nel programma i dati richiesti, si prosegue con il terzo collegamento.

Manage Tables:
Tale interfaccia è dedicata alla manipolazione delle tabelle in input, a seconda dell'analisi che si desidera effettuare. Pertanto questa sezione consiste nel creare nuove tabelle eseguendo comandi che permettono di:

- Selezionare, da una certa tabella, determinate righe/colonne;
- Raggruppare gli elementi per una certa proprietà (media, somma minimo, massimo, etc. . .);

- Creare una nuova tabella specificando una condizione che i dati devono soddisfare.

Tutto questo si può realizzare comodamente tramite query *SQL* generate automaticamente dal programma oppure inserite manualmente dall'utente.
I comandi SQL gestiti automaticamente dal programma sono: *select*, *from*, *where* e *groupby*. Quest'ultimo permette di raggruppare gli elementi calcolando, per ogni gruppo: numero di elementi, media, minimo, massimo, somma, varianza e deviazione standard.

Nonostante SQL sia un linguaggio differente da quelli utilizzati dai diversi contenitori di dati, il software permette di utilizzarlo senza preoccuparsene di ciò, in quanto BiDAl è in grado di mappare automaticamente i comandi SQL nel linguaggio dello storage utilizzato. Così, ad esempio, se ci troviamo a lavorare su un contenitore dati di tipo Hadoop, il programma andrà a mappare il comando SQL in RHadoop. Similmente, se ci troviamo a lavorare con tabelle contenute in SQLite (oppure Bash), il software automaticamente mapperà il linguaggio SQL in quello RSQLite (oppure Bash). In questa maniera, l'utente può passare da uno storage all'altro senza preoccuparsi delle differenze che vi sono fra loro.
In seguito alle operazioni che andranno a creare le tabelle contenenti i dati da analizzare, si passa al quarto ed ultimo ambiente.

Analysis:
Quest'ultima interfaccia è quella dove avviene la vera e propria analisi dei dati. A questo scopo, vi sono implementati una serie di comandi divisi per categorie. Ogni comando ha un proprio storage di lavoro, ma si vedrà successivamente come questo non comporta un limite all'utente. Vi sono quindi comandi di tipo Bash, R e MapReduce. Dopo ogni comando, il risultato intermedio viene salvato nello storage di lavoro, nonché visualizzato nella *GUI* per offrire un approccio visivo.

Spesso però, ci si può trovare nella situazione in cui, ad esempio, dopo aver eseguito una operazione di tipo MapReduce su un insieme di dati in input, sul risultato ottenuto si vuole effettuare un'analisi con R, passando così

da uno storage all'altro. Il programma permette di fare automaticamente questo tipo di operazioni senza che l'utente si preoccupi del cambiamento. Infatti il software, quando si accorge che viene utilizzato un comando non appartenente al contenitore dati corrente, si occupa di esportare automaticamente la tabella d'interesse dallo storage attuale e importarla in quello giusto per potervi applicare il comando desiderato. Riprendendo l'esempio di prima, se dopo aver eseguito il comando di tipo MapReduce l'utente vuole eseguirne uno di tipo R, il programma esporta la tabella intermedia dallo storage Hadoop e la importa in quello SQLite. L'utente non si accorge di queste esportazioni/importazioni, ma continua la sua analisi senza preoccuparsi del cambiamento da uno storage all'altro.
BiDAl offre la possibilità di eseguire una serie di comandi e, successivamente, salvare la sequenza in una nuova variabile, per poterla riutilizzare in futuro. In caso l'utente sia esperto nell'utilizzo dei linguaggi utilizzati per gestire gli storage, viene data la possibilità di scrivere manualmente qualsiasi comando. Inoltre, i comandi in BiDAl sono salvati in file di testo e caricati all'avvio del programma. Questo approccio rende il software facilmente estendibile e fornisce l'opportunità all'utente di arricchirlo con comandi a lui d'interesse.

2.1.1 SQLite

SQLite è una libreria che permette di gestire piccoli database, interrogabili tramite il linguaggio SQL. Il vantaggio che rende i database adatti al nostro caso, consiste proprio nel fatto che essi sono utilizzabili direttamente a livello applicazione. È stato pertanto scelto di adoperare SQLite, in quanto permette di salvare i dati in un unico file, senza il bisogno di mettere in piedi un server. Il software, comunque, si può facilmente estendere inserendo altri tipi di database.

Le query SQL, utilizzate ampiamente per manipolare tabelle, vengono eseguite mediante R, per mezzo dell'apposita libreria RSQLite. Si adoperano invece le funzioni di R per analizzare i dati che risiedono nel contenitore dati SQLite. Per rendere possibili operazioni mediante R su una struttura dati

ad essa estranea, vengono eseguiti i seguenti passi: inizialmente si esegue una query adottando RSQLite per esportare la tabella d'interesse in RAM; successivamente, viene utilizzato R su questa tabella, ormai estrapolata dalla struttura dove originariamente risiedeva.
Tutto questo viene eseguito in automatico senza che l'utente se ne accorga.

2.1.2 R con *BiDAl*

In BiDAl sono implementati diversi comandi R per effettuare molteplici operazioni sui dati. Pertanto sono disponibili funzioni per compiere differenti analisi:

- Estrarre una colonna da una tabella;
- Filtrare i valori in base ad una condizione;
- Approssimare i dati mediante distribuzioni note (mediante funzione fitdist());
- Creare nuove tabelle risultanti da comandi eseguiti precedentemente;
- Calcolare funzioni aggregate sui dati;
- Mostrare graficamente i risultati.

È opportuno ribadire che i comandi risiedono su file esterni, ergo risulta semplice estendere il programma inserendo comandi d'interesse per svariati utilizzi.

2.1.3 Hadoop/MapReduce con *BiDAl*

Per poter riuscire nella gestione dei Big Data, è stato necessario aggiungere al programma il supporto ad Hadoop. Generalmente, le funzioni Mapper e Reducer sono implementate in Java: in questo modo vengono generati file che andranno compilati per poi successivamente essere eseguiti. Questo

approccio però, oltre a complicare l'implementazione, forzerebbe l'utente ad utilizzare diverse sintassi.

Per ovviare a ciò, si è scelto di implementare le funzioni di Mapper e Reducer tramite la libreria *RHadoop*, cosicché l'utente potesse utilizzare R sia sui file contenuti in SQLite che su quelli in Hadoop. Per riuscire in una operazione di MapReduce, inizialmente i file desiderati devono essere trasferiti dallo storage al file system HDFS per poi implementare le funzioni Mapper e Reducer d'interesse. La fase riguardante il passaggio dei file all'HDFS viene eseguita in automatico dal programma; invece, per quanto riguarda le funzioni di Mapper e Reducer, il programma permette di estenderle facilmente per modificarle in base a ciò che l'utente desidera fare.

2.2 Esempi di Utilizzo: analisi delle tracce di Google

Osserviamo di seguito degli esempi di utilizzo del programma.

2.2.1 Eventi di downtime delle macchine

Nelle tracce rilasciate da Google vi è un file che descrive gli eventi delle macchine del loro data center. Questa tabella viene chiamata machine_events e contiene 6 colonne:

- Timestamp;
- Id della macchina;
- Tipo di evento;
- Id della piattaforma;
- Capacità di CPU;
- Capacità di memoria.

Gli eventi sono di 3 tipi:

- Inserimento (0): la macchina si unisce al cluster;
- Rimozione (1): la macchina viene rimossa dal cluster;
- Aggiornamento (2): la macchina viene aggiornata e le sue risorse disponibili vengono modificate.

Le ragioni per cui le macchine vengono rimosse dal cluster non sono state divulgate, tuttavia le tracce indicano che ogni giorno, vi sono circa 2.5% delle macchine che vengono rimosse per poi essere nuovamente aggiunte. Osserviamo quindi un'analisi che studia il tempo di downtime che subiscono le macchine, raffigurando i risultati tramite una funzione di ripartizione cumulativa (CDF):

1. Inizialmente, mediante l'interfaccia *Manage Sources*, viene creato un contenitore dati di tipo SQLite che chiamiamo *Google Storage*, come si può notare in Figura 2.2. In esso viene caricato il file *machine_events.csv*, a noi d'interesse, che nominiamo *Machine_Events* (Figura 2.3). Questo processo viene fatto nell'ambiente *Load Files*.

2. Come secondo passo, la tabella d'origine viene ridotta ad una più piccola contenente soltanto i dati desiderati. Infatti, da *Machine_Events*, si selezionano le prime tre colonne e tra esse, le righe che come tipo di evento hanno 0 (inserimento) oppure 1 (rimozione). Inoltre, il comando generato automaticamente viene modificato manualmente per convertire il tempo da microsecondi a secondi. In questa maniera, viene creata una nuova tabella, chiamata *Machine_Downtime*, contenente i dati necessari per l'analisi (Figura 2.4).

3. Successivamente, si utilizza l'interfaccia *Analysis* per analizzare e rappresentare il downtime delle macchine mediante una CDF. In Figura 2.5 viene mostrata una parte dell'interfaccia di BiDAl in cui si mostra l'anteprima della tabella selezionata, sulla quale vogliamo effettuare

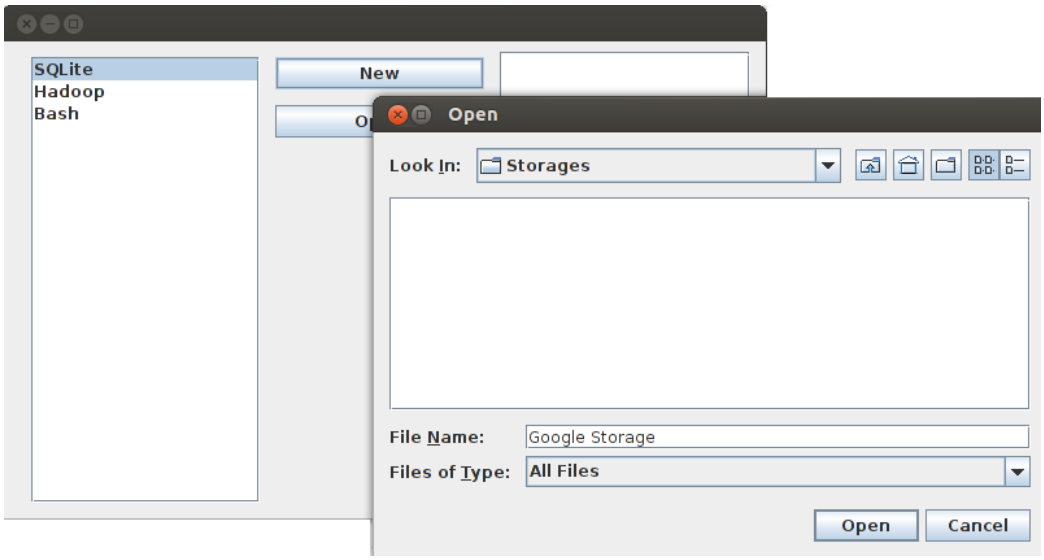

Figura 2.2: Creazione dello storage *Google Storage* di tipo SQLite.

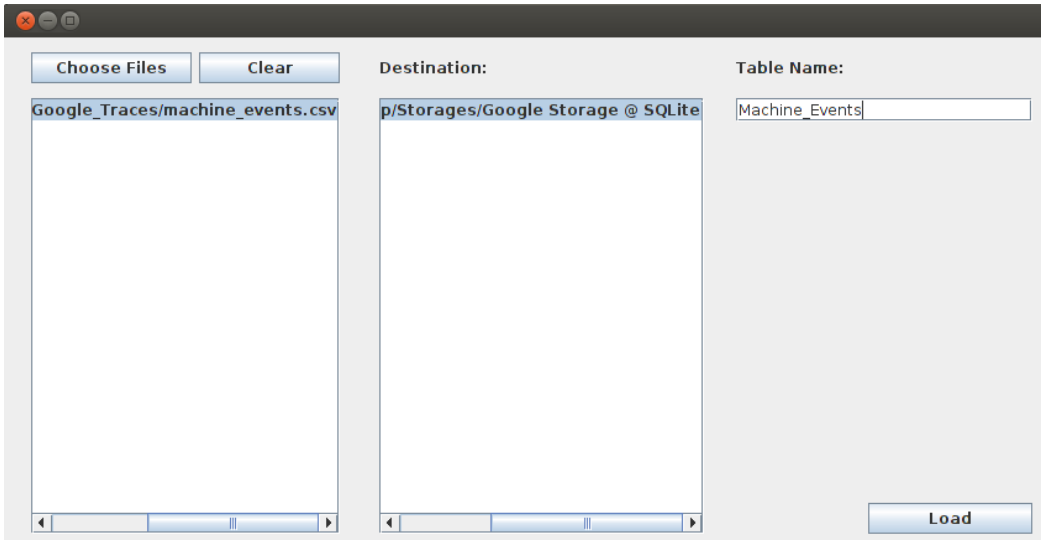

Figura 2.3: Caricamento del file machine_events.csv nella tabella chiamata Machine_Events del contenitore dati *Google Storage*.

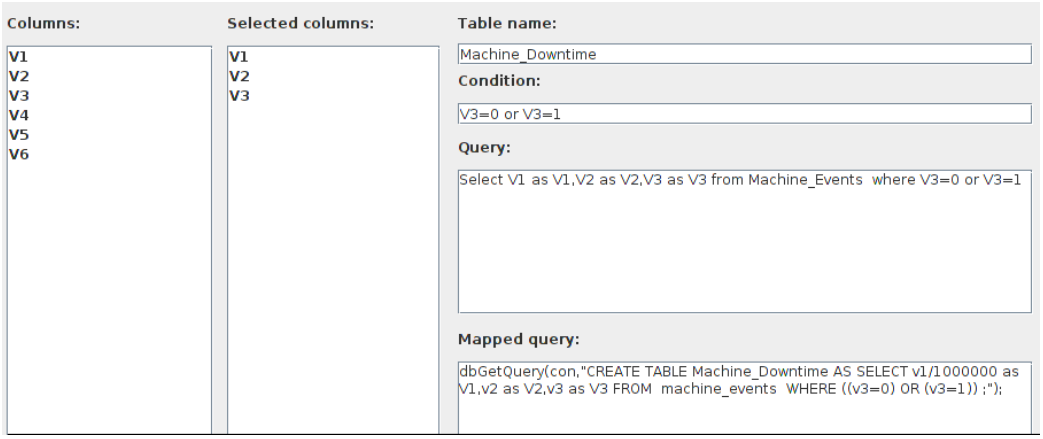

Figura 2.4: Si crea la tabella Machine_Downtime selezionando le colone 1, 2 e 3 dalla tabella Machine_Events, e si aggiunge la condizione desiderata nella quale vengono specificati i valori ammissibili della terza colonna. Il codice nelle aree *Query* e *Mapped Query* viene generato automaticamente.

l'analisi. Pertanto, viene applicato il comando R *aggregate* alla tabella *Machine_Downtime* per aggregare i dati secondo l'id delle macchine (Figura 2.6). Quindi, per ogni macchina, si considerano quelle che hanno 1 (rimozione) come valore dell'evento corrente e 0 (inserimento) come valore dell'evento successivo. In tal modo, viene calcolato il tempo di downtime dato dal tempo che intercorre dal verificarsi dell'evento di tipo 0 a quello di tipo 1.

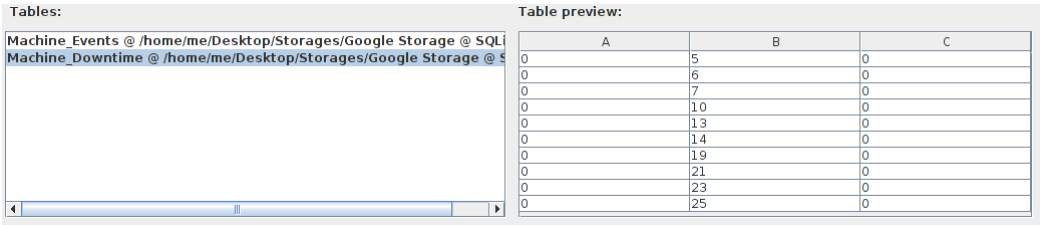

Figura 2.5: Viene selezionata la tabella Machine_Downtime sulla quale si vogliono effettuare delle analisi. Per una rappresentazione visiva, si offre l'anteprima delle prime 10 righe della tabella selezionata.

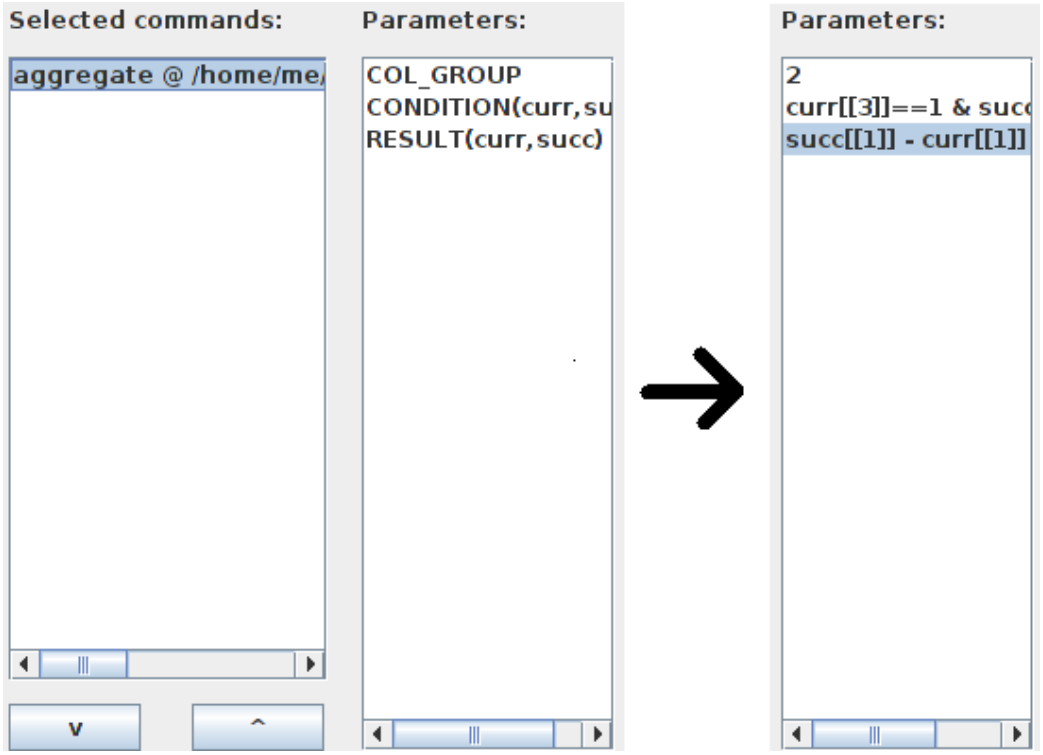

Figura 2.6: Si sceglie di effettuare il comando R *agregate* il quale prende in input dei parametri. Il programma suggerisce il tipo di parametro da inserire.

4. Infine viene utilizzato il comando *ecdf* di R sul risultato ottenuto, per ottenere dai dati la funzione di ripartizione cumulativa. Per una migliore visualizzazione grafica, si è utilizzato il comando *filter* per raffigurare risultati compresi in un massimo di 150 minuti di downtime. In Figura 2.7 si possono osservare i comandi effettuati, mentre il risultato è mostrato in Figura 2.8.

Come si può notare, più del 60% delle macchina subiscono un downtime inferiore a 25 minuti; mentre circa il 20% di esse viene sottoposto ad un downtime superiore a 77 minuti. I risultati ottenuti sono in accordo con altri presenti in letteratura [15].

2.2.2 Frequenza di aggiornamento delle macchine

Analizziamo con quanta frequenza vengono aggiornate le macchine del cluster di Google. Anche in questo caso, lavoriamo con la tabella machine_events, dove sono riportati gli eventi di aggiornamento delle macchine (eventi di tipo 2).

Per far notare come il programma supporta automaticamente il passaggio da un tipo di storage all'altro, si è scelto di effettuare questa analisi utilizzando comandi appartenenti a linguaggi differenti. Per questo motivo, viene creato un nuovo contenitore dati di tipo Bash (chiamato BashStorage). La procedura è la seguente:

1. Inizialmente viene caricato lo storage *Google Storage*, precedentemente realizzato. In seguito, viene creato il nuovo contenitore dati di tipo Bash, chiamato *BashStorage*.

2. Viene presa in esame la tabella machine_events, già presente in *Google Storage*, e da essa ne viene creata una nuova (chiamata *Machine_Events_Seconds*) uguale alla tabella d'origine ma con la differenza che la prima colonna (timestamp) viene convertita in secondi. Questa operazione si può osservare in Figura 2.9.

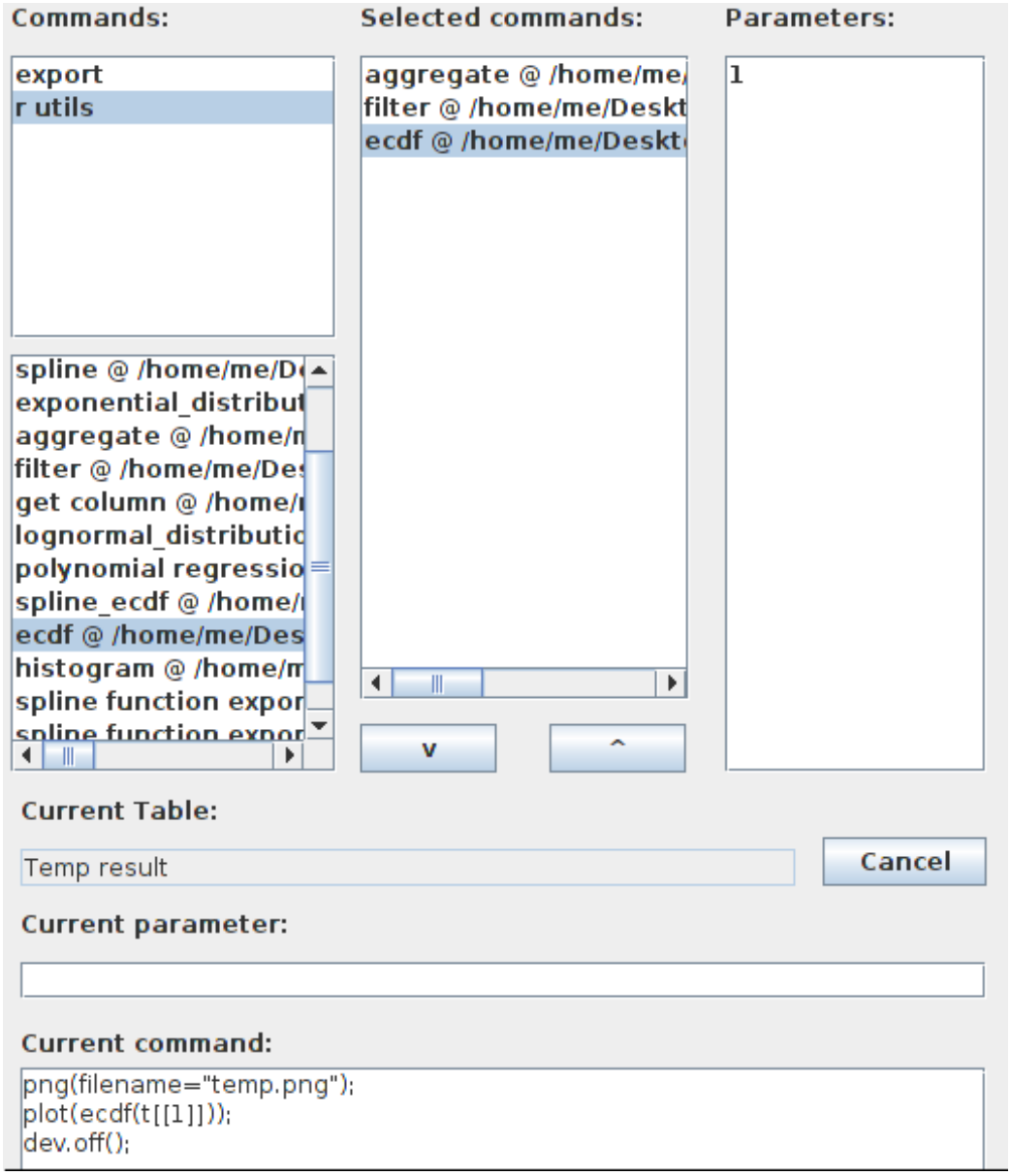

Figura 2.7: Sequenza di comandi R per calcolare il downtime delle macchine a partire dalla tabella Machine_Downtime. I comandi dell'area *Current comand* sono generati automaticamente dal programma.

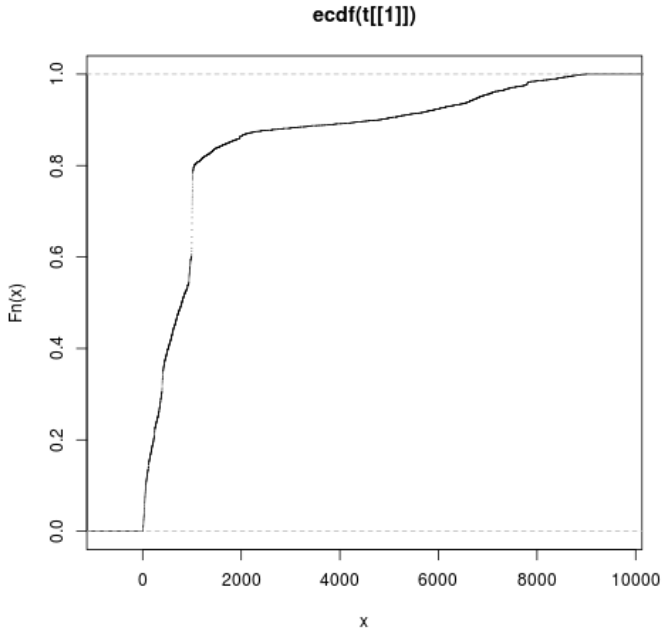

Figura 2.8: Il downtime delle macchine mostrato mediante CDF; sull'asse delle ascisse viene riportato il tempo in secondi, mentre in quello delle ordinate si ha la probabilità che una macchina subisca un determinato downtime.

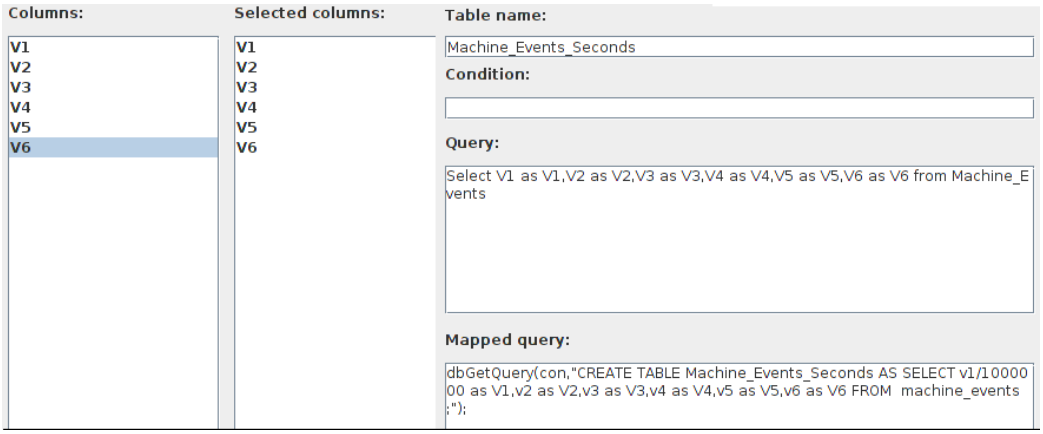

Figura 2.9: Creazione della tabella Machine_Events_Seconds a partire da quella Machine_Events già presente nel contenitore dati *Google Storage* di tipo SQLite. I comandi nelle aree *Query* e *Mapped query* sono generati automaticamente dal programma.

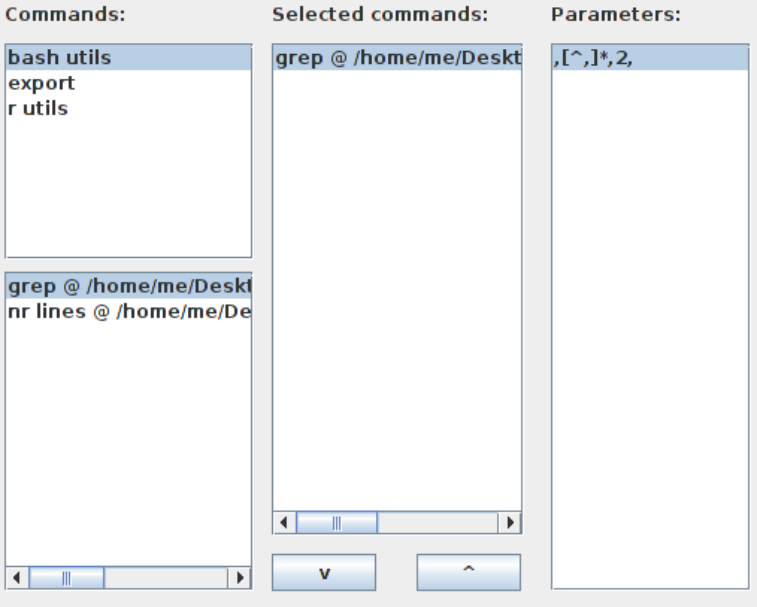

Figura 2.10: Si esegue il comando Bash *grep* su una tabella che risiede in uno storage di tipo SQLite. L'operazione avviene in maniera trasparente all'utente.

3. Sulla tabella *Machine_Events_Seconds*, che ormai risiede in un contenitore dati di tipo SQLite, viene eseguito un comando di tipo Bash per selezionare le righe che contengono un evento di tipo 2 (aggiornamento). In specifico, questa operazione si realizza mediante il comando *grep*, come si può notare in Figura 2.10.

4. Si utilizza la funzione R *get_column* tramite cui si seleziona la prima colonna contenente, a questo punto, il tempo in cui sono avvenuti gli aggiornamenti per ciascuna macchina.

5. Mediante il comando R *differences_between_rows* viene effettuata la differenza fra le righe della colonna precedentemente selezionata, ottenendo così i dati d'interesse (Figura 2.11). Infine, viene applicata sui dati la distribuzione esponenziale, la quale dimostra approssimare al meglio i dati. Il risultato si può notare in Figura 2.12.

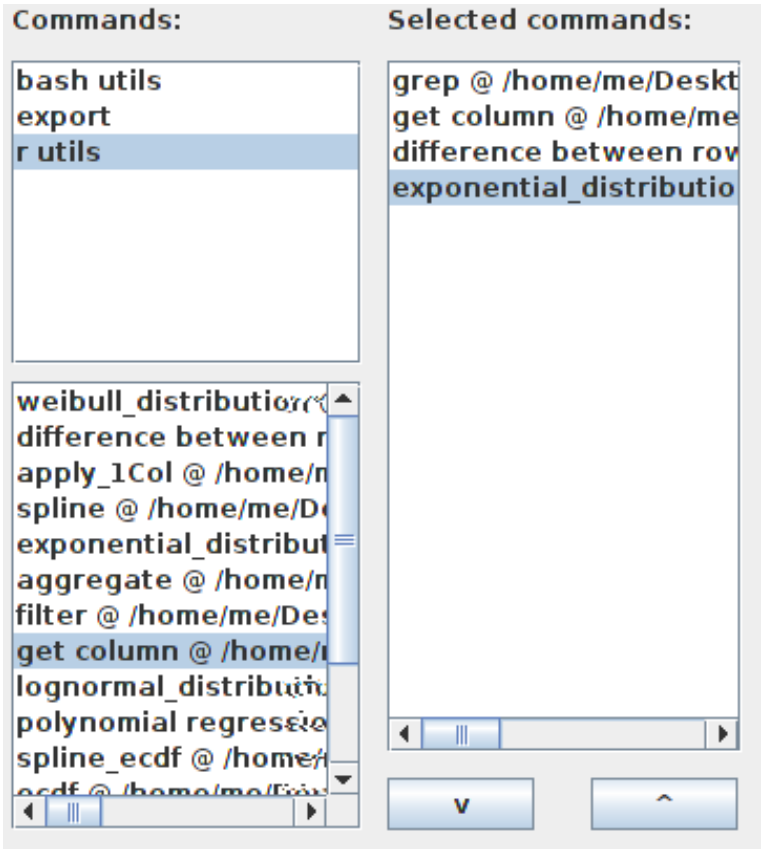

Figura 2.11: Sequenza di comandi R per calcolare la frequenza con cui le macchine vengono aggiornate.

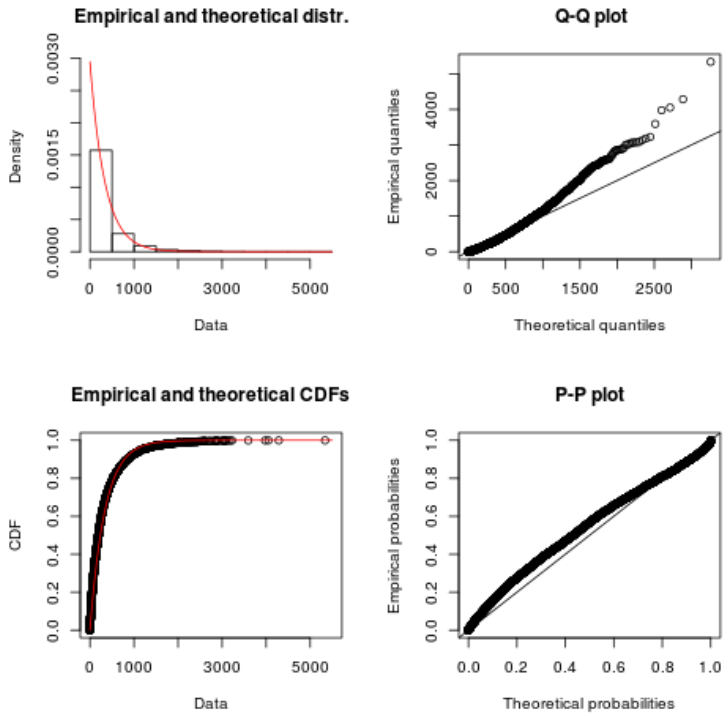

Figura 2.12: Applicazione della distribuzione esponenziale sui dati riguardanti la frequenza di aggiornamento delle macchine. Per una analisi più approfondita, questo comando mostra 4 grafici; il primo mette al confronto la distribuzione esponenziale con i dati empirici, il secondo è un grafico di tipo *Quantile-Quantile*, il terzo confronta la CDF dei dati empirici con quella dei dati teorici mentre il quarto è un grafico di tipo Probability-Probability per valutare quanto sono simili i dati empirici e quelli teorici.

Si noti che l'utente non si accorge del cambiamento da uno storage all'altro; tale operazione viene effettuata automaticamente dal programma.

2.2.3 Task per Job

In questo esempio viene mostrato l'utilizzo di Hadoop con BiDAl. Pertanto, si è scelto di analizzare, per ogni job, il numero di task in esso contenuti.

Per riuscire in questo tipo di analisi, si prende in considerazione la tabella *task_events.csv*, fornita dalle tracce rilasciate da Google. Tale tabella è composta da 13 colonne, tra le quali quelle contenenti l'id dei job e l'id dei task appartenenti ad un job, utili per questa analisi. Pertanto verranno considerate solo le due colonne contenenti tali informazioni non duplicate, creando così una nuova tabella chiamata *Job_Task_Id*.

1. Viene caricata la tabella *task_events.csv* (nominata *Task_Events*) nel contenitore dati *Google Storage* di tipo SQLite.

2. Si crea la tabella *Job_Task_Id* considerando le colonne 3 e 4 della tabella precedentemente caricata, modificando manualmente il comando generato mediante l'aggiunta di "*distinct*", per evitare i duplicati (potrebbero essere presenti più eventi per lo stesso task). In Figura 2.13, nell'area *Mapped Query* si può notare la presenza del comando "distinct", inserito manualmente nella giusta posizione.

3. Vengono applicate le funzioni Map e Reduce di Hadoop: la prima andrà a generare coppie $< job\_id, 1 >$; la seconda invece sarà responsabile di sommare i task per ogni job_id. Successivamente, il risultato viene fornito ad R per una rappresentazione grafica mediante la funzione di ripartizione cumulativa (ecdf). In Figura 2.14 si può osservare il risultato dell'analisi, indicante il fatto che in generale i job contengono pochi task.

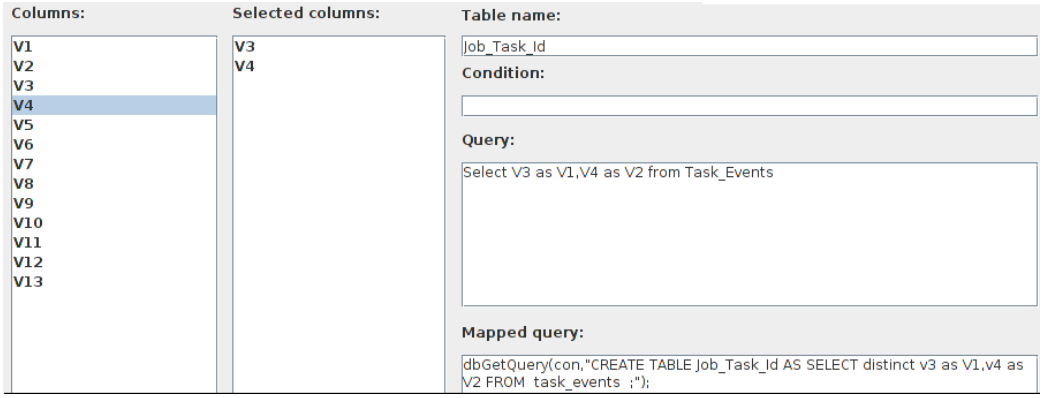

Figura 2.13: Creazione della tabella Job_Task_Id a partire da quella Task_Events.

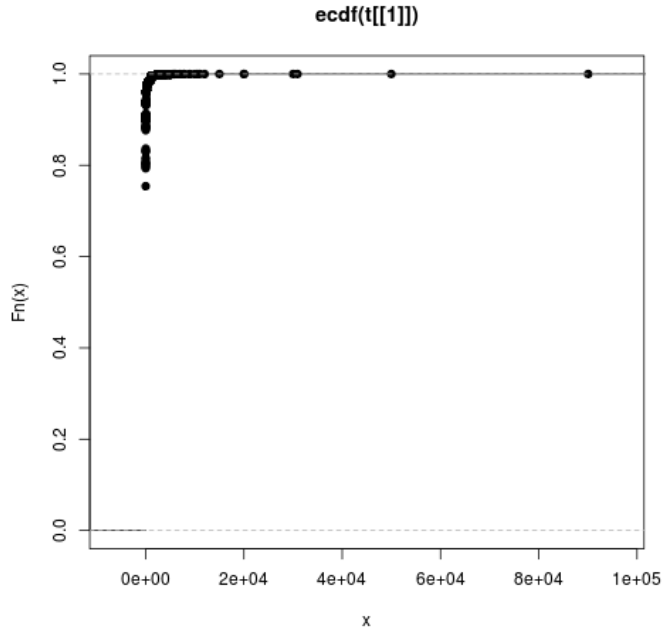

Figura 2.14: Il numero di task per job espresso mediante un CDF.

La creazione della tabella *Job_Task_Id*, come descritto prima, è stata eseguita tramite comandi SQL. È utile però sottolineare che è possibile realizzarla direttamente avvalendosi delle funzioni di tipo Mapper e Reducer, a partire dalla tabella *task_events.csv*: come Map viene adottata una funzione che genera come chiave la coppia $< job\_id, task\_id >$; come Reduce invece viene utilizzata una funzione che adopera come valore la chiave stessa.

2.3 Sintetizzazione delle tracce di Google

Un'analisi approfondita dei Big Data può essere di grande aiuto al fine di prevedere stati critici, pre-critici o sicuri in un data center, ma da solo non è sufficiente.
Infatti, BiDAl fa parte di un progetto più ampio che e per la sua realizzazione segue diversi passi:

- ***Effettuare analisi accurate di tracce esistenti***: utile per capire politiche adoperate da data center reali nonché per poter generare in futuro tracce verosimili contenenti informazioni utili;

- *Effettuare una simulazione di un data center e validarlo con tracce esistenti*: più precisamente, è stato realizzato un modello simulato di un cluster di Google, basandosi sulle tracce rilasciate dall'azienda stessa;

- *Realizzare un modello di guasti*: adatto per poter generare tracce contenenti informazioni utili a tale studio;

- *Applicare tecniche di machine learning*: adoperate per tentare di predire i guasti.

Gli esempi di utilizzo del software, osservati precedentemente, mostrano analisi effettuate sulle tracce rilasciate da Google. BiDAl però, è stato utilizzato anche per generare tracce sintetizzate a partire da quelle messe a

disposizione. Queste tracce sono state utilizzate dal modello simulato volto ad interpretare il data center di Google. Sono state quindi analizzate le seguenti distribuzioni:

- La quantità di CPU richiesta dai task;
- La quantità di memoria richiesta dai task;
- Priorità di ogni task;
- Durata dei task terminanti;
- Durata dei task uccisi;
- Il numero di task contenenti nei job;
- Il tempo di arrivo dei job;
- Tempo di arrivo dei guasti delle macchine;
- Durata dei guasti delle macchine;
- La quantità di CPU delle macchine;
- La quantità di RAM delle macchine.

Tali dati però, come anche riportato in letteratura [7], non seguono una qualche distribuzione nota.
Per poter riuscire a determinare l'andamento dei dati, inizialmente si è pensato di approssimarli mediante interpolazione utilizzando funzioni *spline*. Questo approccio non è risultato soddisfacente, in quanto la spline non riusciva ad interpretare in modo adeguato i dati per via della loro natura troppo oscillante.

Pertanto l'approccio adottato è il seguente: è stato scelto di rappresentare i dati mediante la loro funzione CDF per poi effettuare su di essi della interpolazione mediante un particolare utilizzo delle funzioni spline. Mentre le spline classiche mirano ad ottenere come risultato una curva armoniosa,

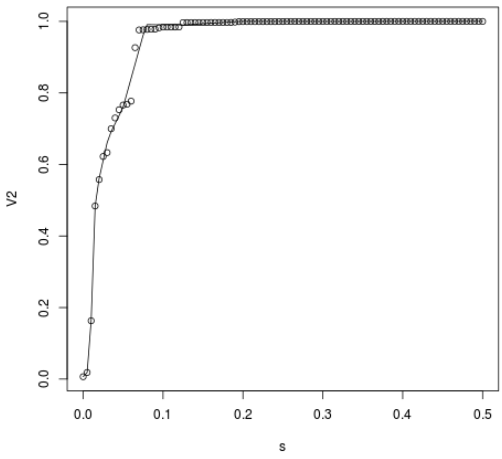

Figura 2.15: Interpolazione mediante spline della funzione CDF rappresentante la quantità di CPU richiesta dai task.

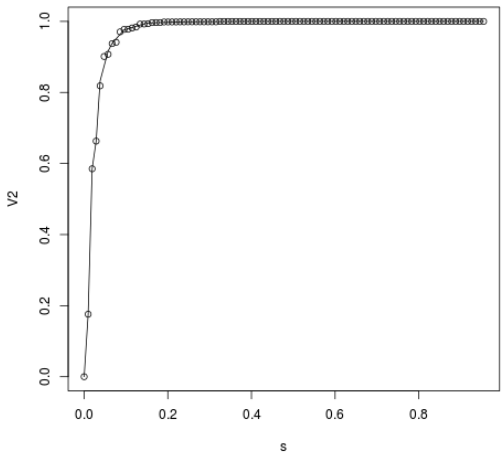

Figura 2.16: Interpolazione mediante spline della funzione CDF rappresentante la quantità di memoria RAM richiesta dai task.

in questo caso invece esse non sono altro che un insieme di segmenti collegati fra loro. Con tale meccanismo i dati si dividono in intervalli e per ogni intervallo viene generata una retta di regressione, cercando di minimizzare l'errore. La grandezza di tali intervalli viene disposta secondo una distribuzione logaritmica. Queste operazioni vengono effettuate in modo automatico dal programma.
Le distribuzioni analizzate sono state considerate come indipendenti l'una dall'altra pur essendo che alcune di esse sono strettamente correlate, come ad esempio la quantità di RAM e CPU domandata.
Nelle figure 2.15 e 2.16 si possono osservare i grafici riguardanti rispettivamente la quantità di CPU e di RAM richiesta dai task.

Per semplificare l'utilizzo delle tracce sintetizzate da parte del simulatore, è stato implementato un comando R che esporta e trasforma i risultati ottenuti mediante BiDAl in codice *C*, direttamente compilabile ed utilizzabile. Questo è stato reso possibile grazie al fatto che R è un linguaggio interpreta-

to ed è quindi possibile manipolare il codice in uso, come fosse una stringa. Successivamente si sono effettuate su tale stringa le opportune sostituzioni, per ovviare a lievi differenze che vi è fra la sintassi di R a quella di C.

Capitolo 3

Architettura di BiDAl

Il programma per l'analisi dei Big Data è stato implementato nel linguaggio di programmazione Java. Per la sua realizzazione è stato scelto di adottare il pattern architetturale *Model View Controller* (MVC), ampiamente utilizzato nell'ambito della programmazione ad oggetti. BiDAl esibisce una struttura facilmente estendibile a nuovi linguaggi e tipi di storage. Inoltre, i comandi disponibili per ogni contenitore dati sono comodamente estendibili in quanto non sono implementati direttamente nel codice ma sono situati su file esterni.

3.1 Architettura delle classi

BiDAl è logicamente diviso in tre macro componenti:

- *Interfaccia utente*: vi sono 5 classi che corrispondono ai 5 ambienti precedentemente descritti (View);
- *Gestione degli storage*: vi sono implementate varie classi volte alla gestione dei vari contenitori dati e dei comandi a loro associati (Model);
- *Interconnessione tra le componenti*: vi è una classe che si occupa di far comunicare l'interfaccia utente con i dati sottostanti (Controller);

Tale architettura generale viene mostrata in Figura 3.1 mediante un diagramma UML, dove vengono mostrate soltanto le classi principali. In seguito verranno approfonditi alcuni aspetti dell'architettura.

Per garantire l'accesso alla classe che interconnette le varie componenti, chiamata *Controller*, è stato utilizzato il design pattern *Singleton* (Figura 3.2). Questo permette di accedere facilmente ad un'istanza di una certa classe mediante il solo utilizzo del suo nome, indipendentemente da dove ci si trovi. il Singleton garantisce inoltre che l'istanza a cui si accede sia unica in tutto il codice.

Per gestire la creazione dei nuovi storage si è deciso di utilizzare una singola classe volta solamente ad istanziare i contenitori dati. La sua architettura, basata su un pattern noto come *Simple Factory*, è mostrata in Figura 3.3. Come si può notare, la classe *StorageFactory* fornisce il numero di storage disponibili ed un Id per ognuno di essi. In questa maniera, è possibile creare nuovi contenitori dati specificando allo StorageFactory l'Id desiderato.

Per osservare il funzionamento degli storage e dei comandi a loro associati si prende in esempio il caso del contenitore dati SQLite (raffigurato con la classe *SqliteStorage*) il quale come linguaggio utilizza R. Lo schema UML ad esso dedicato viene mostrata in Figura 3.4.

Come si può notare, SqliteStorage estende la classe *GenericStorage*, utilizzata per riferirsi ad un qualsiasi storage senza saperne il tipo concreto. Inoltre utilizza le classi *Rexecuter* per eseguire comandi R ed SqliteVisitor per mappare query SQL in comandi RSQLite. Vi sono inoltre presenti le classi *Command* e *AnalysisSource*: la prima espone i comandi disponibili mentre la seconda si occupa di mostrare le tabelle a disposizione.

3.2 Livello di astrazione

Come precedentemente discusso, per gestire le tabelle vi sono due possibili modalità: mediante query SQL oppure utilizzando direttamente i comandi disponibili allo storage.

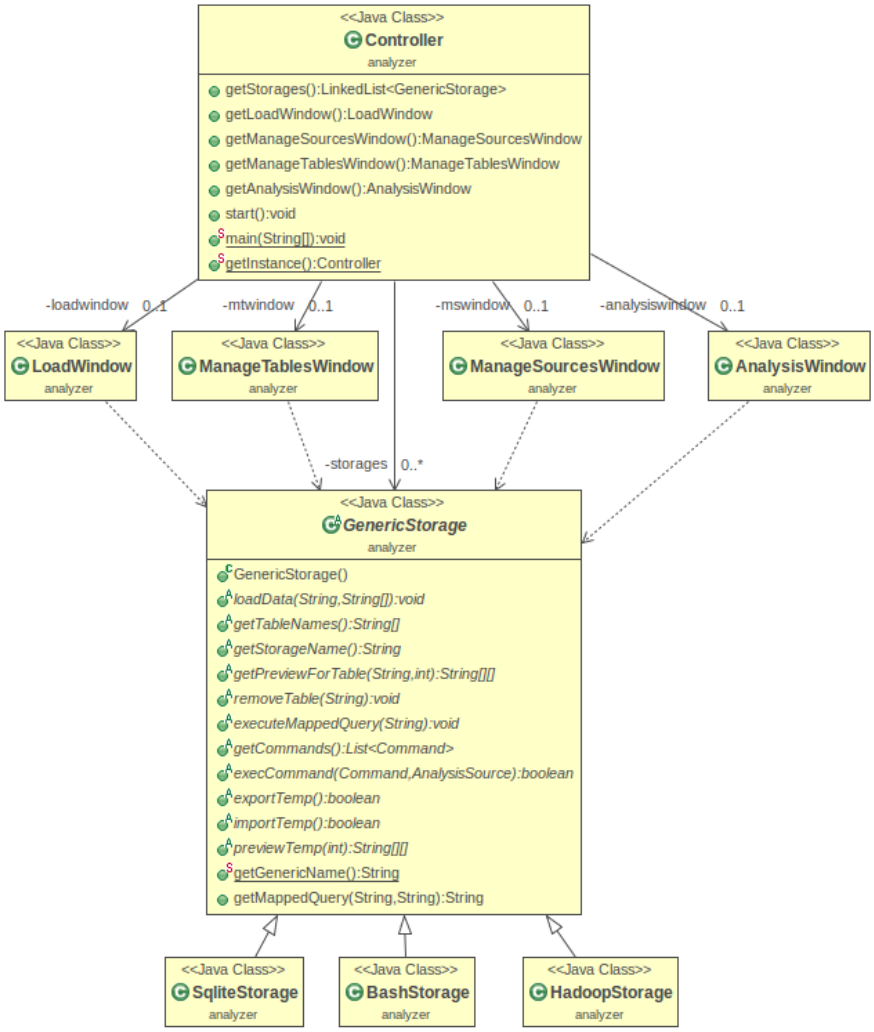

Figura 3.1: Architettura generale di BiDAl.

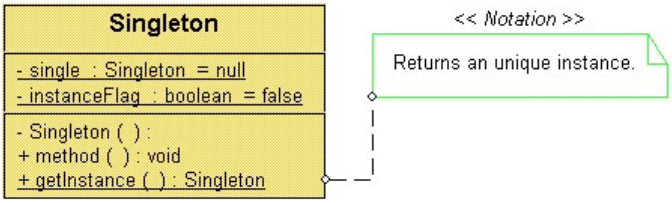

Figura 3.2: Esempio di una classe Singleton rappresentata mediante un diagramma UML.

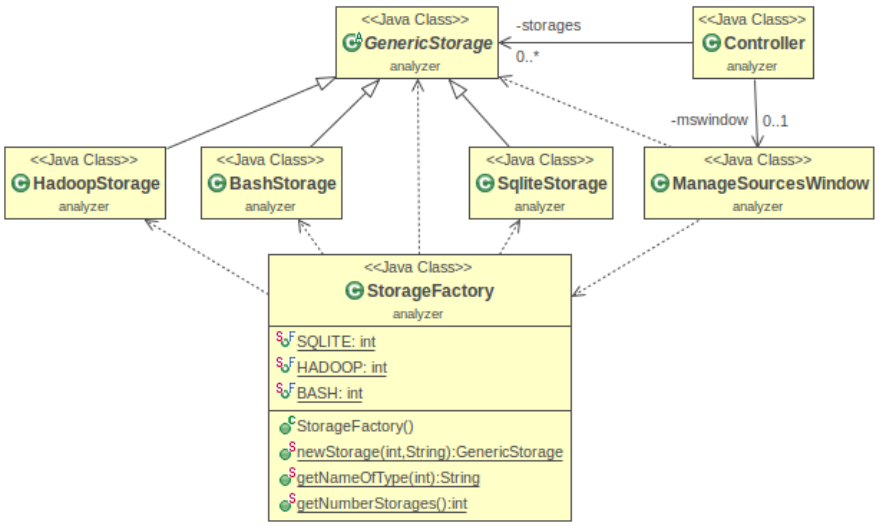

Figura 3.3: Schema UML che mostra in dettaglio l'architettura utilizzata per la gestione degli storage.

Il primo approccio viene reso possibile in quanto le query SQL vengono automaticamente mappate nel linguaggio sottostante, specifico per il contenitore dati in utilizzo. Più in dettaglio, è stato utilizzato un parser SQL opensource, chiamato *Akiban*, facente parte dell'omonimo DBMS. Esso viene utilizzato per generare una struttura dati ad albero a partire dalla stringa SQL datagli in input. Per visitare l'albero è stato adottato il design pattern *Visitor* (Figura 3.5) la cui implementazione in BiDAl si occupa di convertire l'albero in un comando specifico per lo storage. Ad esempio, la clausola *WHERE* del linguaggio SQL viene mappata in R mediante il comando *subset*.

Per quanto riguarda il secondo approccio, il programma permette di utilizzare comandi dedicati ad uno storage su tabelle di uno differente. Questo viene effettuato in maniera trasparente all'utente, rendendo quindi disponibili a tutti gli storage i comandi dedicati ad un certo contenitore dati. Questo viene reso possibile importando ed esportando, ad ogni cambio di storage, il risultato temporaneo. Le operazioni di importazione ed esportazione vengono effettuate mediante file di tipo *csv*. Ovviamente questo approccio suggerisce di effettuare un cambiamento di tipo di storage su risultati intermedi (di di-

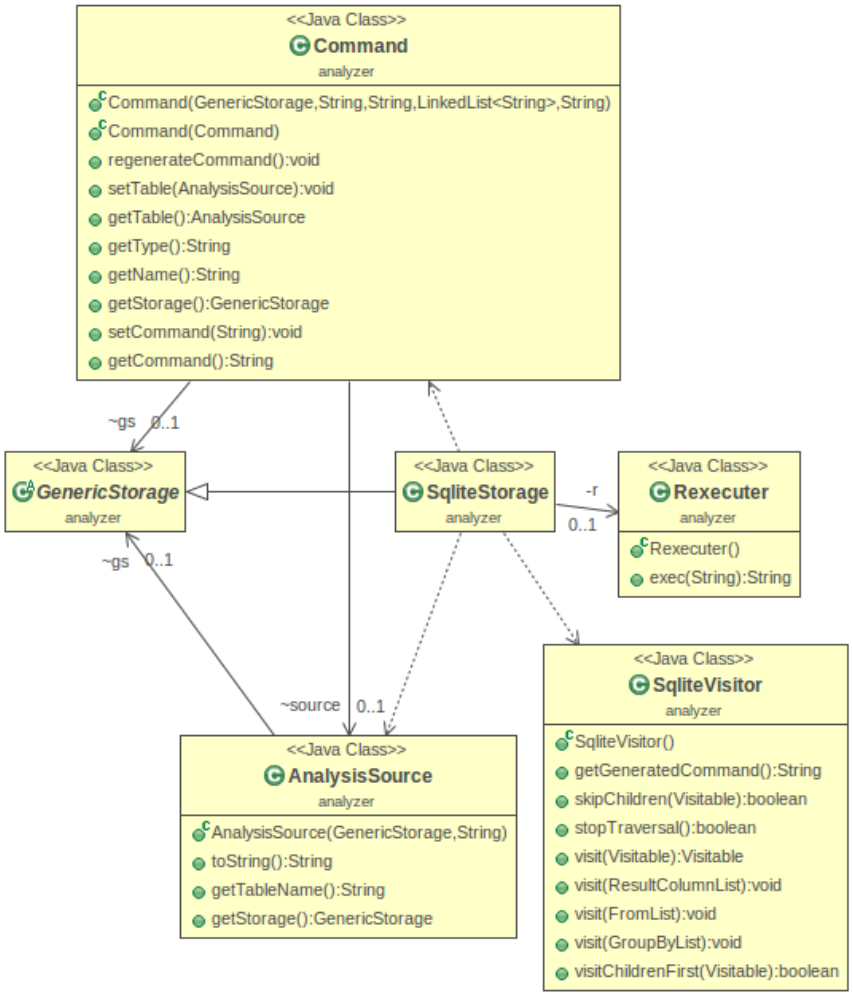

Figura 3.4: Esempio dell'architettura di uno storage di tipo SQLite.

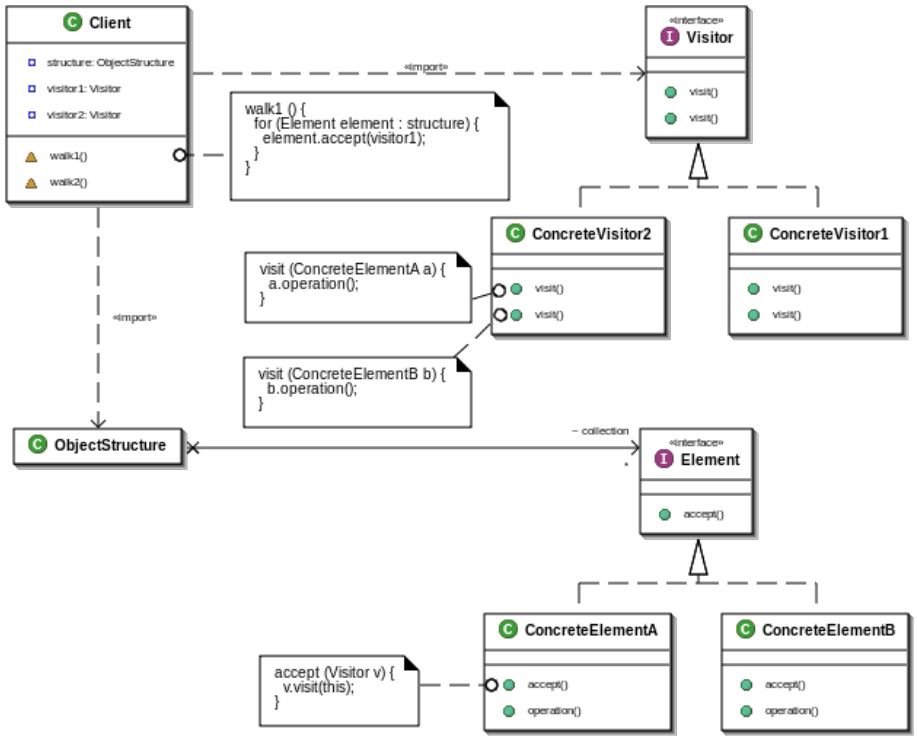

Figura 3.5: Schema UML dell'architettura utilizzata dal pattern Visitor.

mensioni ridotte) e non sulle tabelle originali, in quanto le grandi dimensioni potrebbero rallentare l'esecuzione (ad esempio è intelligente far gestire ad Hadoop tabelle che non possono essere contenute in RAM).
L'algoritmo con cui vengono eseguiti una sequenza di comandi è descritto dallo pseudocodice seguente:

```
storageVecchio = null
storageNuovo = null
storageTabella = null

for (comando c ∈ listacomandi){
        storageNuovo = c.getStorage();
        tabella = c.getTable();
        storageTabella = tabella.getStorage();

        /*caso in cui si passa una tabella
        temporanea tra due diversi storage*/
        if (storageNuovo != storageVecchio AND
            storageVecchio != null AND
            tabella == "temporanea"){
                storageVecchio.export();
                storageNuovo.import();
        }
        /*caso in cui si da in input una tabella
        ad un comando di un diverso storage*/
        if (tabella != "temporanea" AND
            storageTabella != storageNuovo){
                storageTabella.load(tabella);
                storageTabella.export();
                storageNuovo.import();
                storageNuovo.exec(c,"temporanea");
        }
```

```
        else
                storageNuovo.exec(c, tabella);

        storageVecchio = storageNuovo;
}
risultato = storageVecchio.getData();
```

3.2.1 Estendibilità del programma: creazione di nuovi storage

BiDAl è stato progettato per essere estendibile fornendo la possibilità di implementare facilmente nuovi contenitori dati. Per astrarre dal tipo di storage utilizzato, è stata implementata una classe astratta chiamata *GenericStorage* che viene utilizzata per gestire un qualsiasi storage concreto. Pertanto, per creare un nuovo contenitore dati è sufficiente estendere la classe GenericStorage implementandone i metodi astratti. Essi devono fornire le seguenti funzionalità:

- *loadData(tablename, files)*: crea una tabella chiamata *tablename* contenente i dati presenti in *files*;
- *getTableNames()*: restituisce la lista di tabelle presenti nello storage;
- *getStorageName()*: restituisce il nome specifico dello storage in uso;
- *getGenericName()*: restituisce il nome generico del tipo di storage utilizzato;
- *getCommands()*: fornisce i comandi disponibili per lo storage corrente
- *execCommand(command, analysisSource)*: esegue il comando *command* sulla tabella *analysisSource*; se tale tabella non viene specificata, il comando viene eseguito sul risultato temporaneo;
- *importTemp()*: carica da disco il risultato temporaneo;

- *exportTemp()*: esporta su disco il risultato temporaneo;
- *previewTemp(n)*: fornisce le prime n righe del risultato temporaneo;
- *getPreviewForTable(table, n)*: fornisce le prime n righe della tabella *table*;
- *removeTable(table)*: elimina la tabella specificata;
- *getMappedQuery(dest, querytree)*: converte l'albero SQL *querytree* in una stringa rappresentante il comando da eseguire sullo storage corrente, con *dest* come tabella di destinazione;
- *executeMappedQuery(query)*: esegue il comando SQL convertito, specificato da *query*.

Si noti che per realizzare il metodo getMappedQuery è necessario fornire un'implementazione dell'interfaccia *Visitor*, volta a generare il comando corretto, specifico per ogni tipo di storage, per ogni nodo trovato sull'albero di partenza.

3.2.2 Estendibilità degli storage: creazione di nuovi comandi

Ogni contenitore dati, in base al linguaggio da esso utilizzato, contiene un insieme di comandi applicabili ai dati contenuti. Per agevolare la facile creazione di nuovi comandi, è stato scelto di salvarli su file esterni al programma. Saranno poi gli storage ad occuparsi di caricare questi file.

Ogni contenitore dati ha a sua disposizione una cartella generale in cui sono contenute altre sottocartelle indicanti la categoria dei comandi in esso contenuti. All'utente vengono mostrati i comandi divisi per gruppi: nel caso diversi storage contengano le stesse categorie, all'utente ne viene mostrata un'unica contenente il loro *merge*. È stato scelto questo approccio affinché venga fornita all'utente una visione che astragga dal tipo di contenitore dati e che invece offra una divisione più funzionale.

Il formato per rappresentare i comandi su file è di seguito mostrato mediante grammatica *BNF*:

⟨*file*⟩ ::= ⟨*nome comando*⟩ ⟨*newline*⟩
⟨*numero parametri*⟩ ⟨*newline*⟩
⟨*lista parametri*⟩ ⟨*newline*⟩
⟨*codice comando*⟩

⟨*lista parametri*⟩ ::= ⟨*descrizione parametro*⟩ ⟨*newline*⟩ ⟨*lista parametri*⟩
| ⟨*empty*⟩

⟨*codice comando*⟩ ::= ⟨*testo*⟩
| ⟨*codice comando*⟩ ⟨*parametro*⟩ ⟨*codice comando*⟩
| ⟨*empty*⟩

⟨*parametro*⟩ ::= '$PAR' ⟨*numero parametro*⟩ '$'

Ad esempio, il comando R che genera un istogramma a partire da un insieme di dati datogli in input, ha la seguente forma:

```
histogram
1
COLUMN
png(filename="temp.png");
hist(t[[$PAR1$]]);
dev.off();
```

Conclusioni e sviluppi futuri

È stato realizzato un programma (BiDAl), scritto in linguaggio Java, in grado di permettere analisi su dati dell'ordine dei Big Data, offrendo la possibilità di esaminare l'input in maniera automatica. Il programma riesce ad astrarre dal tipo di storage in cui risiedono i dati, rendendo le numerosi operazioni disponibili trasparenti all'utente. Infatti esso permette di processare i dati utilizzando un certo linguaggio per poi analizzarne il risultato mediante uno diverso, supportando così il passaggio da un tipo di contenitore dati all'altro in maniera automatica.
Sono state implementate funzioni in grado di eseguire query SQL, comandi R, nonché comandi di tipo Hadoop/MapReduce per poter gestire i dati in maniera distribuita.

BiDAl è stato validato mediante analisi delle tracce rilasciate da Google, producendo risultati identici a quelli presenti in letteratura. Inoltre, il programma è stato utile per effettuare un'analisi mirata alla comprensione delle politiche adoperate per la gestione dei data center di Google, nonché per la creazione delle tracce sintetizzate a partire da quelle esistenti a nostra disposizione. Queste ultime sono state adoperate da un modello volto a simulare un cluster di Google.

L'architettura del software garantisce una divisione logica basandosi sul pattern Model View Controller. Ad ogni livello, BiDAl esibisce una struttura facilmente estendibile sia a nuovi linguaggi e tipi di storage, che ad ulteriori comandi disponibili per ogni contenitore dati.

In futuri sviluppi, le funzionalità del programma possono essere estese

ad altri tipi di storage e linguaggi utili, inserendo l'opportunità di effettuare in modo automatico un'analisi incentrata allo studio delle correlazioni fra i dati.

Bibliografia

[1] Zikopoulos P., Eaton C, deRoos D., Deutsch T., and Lapis G. *Understanding Big Data: Analytics for Enterprise Class, Hadoop and Streaming Data*.

[2] Eric schmidt: Every 2 days we create as much information as we did up to 2003. Posted at `http://techcrunch.com/2010/08/04/schmidt-data/`,.

[3] Data on big data. Posted at `http://marciaconner.com/blog/data-on-big-data/`,.

[4] Alan Turing. Computing machinery and intelligence, mind 49(1949), 433-460. Technical report.

[5] Summary of the december 24, 2012 amazon elb service event in the us-east region. Posted at `http://aws.amazon.com/message/680587/`,.

[6] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2`.

[7] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical Report ISTC–CC–TR–12–101, Intel science and technology center for cloud computing,

Carnegie Mellon University, Pittsburgh, PA, USA, April 2012. Posted at http://www.istc-cc.cmu.edu/publications/papers/2012/ISTC-CC-TR-12-101.pdf.

[8] The University of WAIKATO. Weka 3: Data mining software in java. Posted at http://www.cs.waikato.ac.nz/ml/weka.

[9] Oracle data mining. Posted at http://www.oracle.com/technetwork/database/options/advanced-analytics/odm/index.html.

[10] Ibm db2 intelligent miner tutorials. Posted at http://www.ibm.com/developerworks/data/library/tutorials/iminer/iminer.html.

[11] The r project for statistical computing. Posted at http://www.r-project.org.

[12] W. N. Venables, D. M. Smith, and the R Core Team. An introduction to r. Version 3.0.2 (2013-09-25).

[13] Vito Ricci. Fitting distributions with r. Technical report. Release 0.4-21 February 2005.

[14] hadoop. http://hadoop.apache.org.

[15] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a Google cluster. In *8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)*, Pittsburgh, PA, USA, September 2012.

Ringraziamenti

Ringrazio il professor Ozalp Babaoglu per il grande supporto a questo lavoro. Ringrazio inoltre il professor Moreno Marzolla per avermi seguita con attenzione e per i preziosi consigli.
Un ringraziamento speciale va alla mia famiglia, che con il loro appoggio ed il loro spirito critico hanno reso migliore questo percorso.