Università degli Studi di Pisa

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea in Informatica

IMPLEMENTAZIONE DI UN DATABASE ROUND-ROBIN CON SUPPORTO PER I VETTORI

Daniele Trainini

Gestione di Rete

Anno Accademico 2009–2010

Introduzione

Questa relazione è stata realizzata a conclusione del progetto "*Implementazione di un database Round Robin con supporto per i vettori*", svolto per il corso Gestione di Rete.

Di seguito viene presentata una breve introduzione al concetto di Database Round Robin o Database circolare, descrivendo le caratteristiche che lo rendono uno strumento adatto per software di monitoraggio di rete.

Database Round Robin

Un *Round Robin Database*, comunemente abbreviato in *rrd*, è un database circolare utilizzato in scenari in cui è necessario memorizzare dati raccolti ad intervalli di tempo regolari.

Questo tipo di database permette di immagazinare i dati in un buffer circolare, di dimensione prestabilita, aggregandoli secondo una o più funzioni specificate al momento della creazione del database stesso. Quando il buffer è pieno e una nuova aggregazione deve essere salvata, l'inserimento più vecchio viene sovrascritto per fare posto al più recente.

Questa peculiarità garantisce che le dimensioni del database sul disco non crescano indefinitamente al crescere del numero di inserimenti, garantendo una dimensione e un numero di elementi memorizzati praticamente costante.

In un database di questo tipo deve essere prevista la gestione di situazioni particolari, come la mancanza di valori pervenuti durante l'intervallo di tempo prestabilito o al contrario la presenza di più dati disponibili durante uno stesso intervallo. Questi eventi possono essere gestiti aggiungendo un valore *indefinito* nel primo caso e interpolando tra loro i valori disponibili nel secondo.

Un caso tipico in cui può essere sfruttato un database con queste caratteristiche è la generazione di statistiche sull'utilizzo di una rete. Possono essere memorizzati e aggregati tra loro i dati pervenuti da vari dispositivi presenti sulla rete al fine di monitorare l'utilizzo della banda, il numero di connessioni attive, gli host che stanno effettuando più traffico, etc.

Ad oggi l'implementazione di Database Round Robin più famosa è `RRDtool`, un software open-source di Tobi Oetiker che permette di memorizzare, aggregare e produrre grafici statistici su dati raccolti ad intervalli di tempo predefiniti.

`RRDtool` è stato studiato e utilizzato come linea guida per alcune scelte implementative effettuate durante lo svolgimento di questo progetto.

Requisiti Funzionali

Verranno ora descritti i requisiti funzionali che il progetto di un Database Round Robin deve soddisfare.

Creazione del database Si dovrà sviluppare una funzione che in base ai parametri in ingresso scelti dall'utente crei un database circolare. Durante questa fase dovranno essere scelti i tipi dei dati che saranno memorizzati nel database, le funzioni mediante le quali saranno aggregati e l'intervallo di tempo al termine del quale eseguire l'inserimento dei dati stessi.

Inserimento di dati Dovrà essere sviluppata una routine tramite la quale sia possibile inserire i dati nel database. Questa funzione deve prevedere comportamenti differenti in base al tipo di dato che andiamo a gestire e ad inserire nel database. Durante questa fase, la routine deve richiamare le funzioni di aggregazione a seconda del momento in cui avviene l'inserimento.

Recupero di dati Deve essere possibile estrarre i dati memorizzati nel database. Non sono richiesti i valori delle singole letture, ma solamente quelli aggregati.

In fase di progettazione è emerso che per garantire una maggiore flessibilità al database doveva essere fornita all'utente la possiblità di utilizzare dei tipi di dato e delle funzioni di aggregazione personalizzati.

Realizzazione

Il progetto è stato interamente scritto in linguaggio Python per garantirne la portabilità. L'applicazione è stata suddivisa in tre componenti principali:

- `r2dlib` Una libreria che provvede alle principali funzioni del progetto, l'inserimento e il recupero dei dati.
- `toydb` Un un semplice database che può essere utlizzato come backend.
- `cli` Un'interfaccia a riga di comando che utilizza le precedenti componenti.

r2dlib

La componente principale del progetto è `r2dlib`. Questa libreria implementa le funzioni basilari per un database Round Robin, come l'inserimento e l'estrazione dei dati.

Anziché sviluppare un database di backend, le funzioni presenti in questo modulo sono state implementate in modo che salvino i dati in un *oggetto* che si assume esporre una determinata interfaccia e non in un database di backend predefinito. Questo svincola l'utente della libreria nella scelta del backend da utilizzare. Per fornire una linea guida ad eventuali implementazioni alternative di backend è stata realizzata una classe astratta, i cui metodi sono quelli richiesti ad un oggetto che serva da backend.

Questa componente fornisce anche alcune funzioni che permettono di utilizzare tipi di dato e funzioni di aggregazione implementate al di fuori della libreria.

toydb

Il modulo `toydb` è un semplice database di backend che realizza l'interfaccia fornita e necessaria a `r2dlib`.

Questa componente affida la gestione della persistenza dei dati al modulo per la serializzazione `Cpickle/pickle`, incluso in ogni installazione standard del linguaggio Python.

Per rendere le scritture su file atomiche, queste vengono effettuate su un file temporaneo che verrà poi rinominato col nome definitivo, assumendo che la `mv` sia un'operazione atomica sul file system sul quale è presente il database.

La gestione Round Robin degli inserimenti è realizzata inserendo le nuove letture sempre in coda ad una tabella e rimuovendo quelle più vecchie quando questa raggiunge una determinata dimensione.

È stato implementato anche un *context manager* all'interno del quale è possibile lavorare sul database in modalità esclusiva. Il blocco dell'intero database si basa sulla creazione di un file di lock, che viene rimosso all'uscita del *context manager* stesso.

Cli

La componente `cli` è un tool a riga di comando basato su `r2dlib` e `toydb`. L'interfaccia è provvista di un semplice *help*, messo a disposizione dell'utente mediante il flag `-h`, come nell'esempio seguente:

```
dani@gilean:r2dlib$ ./cli.py -h
usage: cli.py [-h] {create,update,fetch} ...

R2DLIB TOOL

positional arguments:
  {create,update,fetch}

optional arguments:
  -h, --help
```

È possibile ottenere un *help* dettagliato facendo seguire il nome di ogni comando dal flag `-h`.

```
dani@astinus:~/code/r2dlib$ ./cli.py create -h
usage: cli.py create [-h] --datasource TYPE:NAME:HEARTBEAT:MIN:MAX
                     --congregation FUNCTION:XFF:RESOLUTION:ROWS --step STEP
                     --creation CREATION --persistence PERSISTENCE

optional arguments:
  -h, --help            show this help message and exit
  --datasource TYPE:NAME:HEARTBEAT:MIN:MAX
  --congregation FUNCTION:XFF:RESOLUTION:ROWS
  --step STEP
  --creation CREATION
  --persistence PERSISTENCE
dani@astinus:~/code/r2dlib$
```

Esempi di utilizzo

Di seguito sono riportati alcuni esempi di utilizzo del tool `cli.py`.

- Creazione di un semplice database contenente una sola *datasource* di tipo non interpolabile che sarà aggregata secondo la funzione *massimo*. In questo esempio il *timestamp* di creazione del database è 1000, specificato dal parametro `--creation`. Con il parametro `--step`, invece, si specifica che l'intervallo tra due letture consecutive sia di 300 secondi.

```
dani@gilean:r2dlib$ ./cli.py create --datasource NI:speed:50:0:100\
 --congregation MAX:3:3:5 --step 300 --creation 1000\
 --persistence speed1.dat
dani@gilean:r2dlib$
```

  Lettura di alcuni dati per popolare il database appena creato. In questo esempio vengono specificati il *timestamp*, tramite il parametro `--ts`, e il file su cui memorizzare i dati con il parametro `--db`.

```
dani@gilean:r2dlib$ ./cli.py update --ts 1300 --db speed1.dat 30
dani@gilean:r2dlib$ ./cli.py update --ts 1600 --db speed1.dat 90
dani@gilean:r2dlib$ ./cli.py update --ts 1900 --db speed1.dat 50
```

  Esecuzione di una query per estrarre i dati della funzione *massimo* compresi tra l'istante 1000 e l'istante 2200.

```
dani@gilean:r2dlib$ ./cli.py fetch --start 1000 --end 2200 --cf MAX\
 --db speed1.dat

1000 None
1300 None
1600 None
1900 90
2200 None

dani@gilean:r2dlib$
```

- Creazione di un database con due *datasource*, un *contatore* e un *gauge*, aggregati secondo due funzioni: il *massimo* e il *last*.

```
dani@gilean:r2dlib$ ./cli.py create --datasource COUNTER:km:50:0:100\
 --datasource GAUGE:temp:50:0:200 --congregation MAX:3:3:5\
 --congregation LAST:2:2:5 --step 300 --creation 1000\
 --persistence speed2.dat
dani@gilean:r2dlib$
```

Lettura di alcuni dati per popolare il database.

```
dani@gilean:r2dlib$ ./cli.py update --ts 1300 --db speed2.dat 50 100
dani@gilean:r2dlib$ ./cli.py update --ts 1600 --db speed2.dat 60 200
dani@gilean:r2dlib$ ./cli.py update --ts 1900 --db speed2.dat 70 300
```

Estrazione dei valori per la funzione *massimo* nell'intervallo tra 1300 e 1900. I due risultati mostrati nel seguito corrispondono alle due *datasource* presenti nel database.

```
dani@gilean:r2dlib$ ./cli.py fetch --start 1300 --end 1900\
 --cf MAX --db speed2.dat

1300 None
1600 None
1900 200.0



1300 None
1600 None
1900 10

dani@gilean:r2dlib$
```

- Creazione di un database con una sola *datasource* di tipo *vector*, aggregata secondo la funzione *last*.

```
dani@gilean:r2dlib$ ./cli.py create --datasource VECTOR:speed:50:0:0\
 --congregation LAST:3:3:5 --step 300 --creation 1000\
 --persistence speed3.dat
```

  Riempimento del database con tre vettori. Per specificare un vettore è sufficiente suddividere i suoi campi utilizzando il simbolo ":".

```
dani@gilean:r2dlib$ ./cli.py update --ts 1300 --db speed3.dat 70:300:400
dani@gilean:r2dlib$ ./cli.py update --ts 1600 --db speed3.dat 90:200:500
dani@gilean:r2dlib$ ./cli.py update --ts 1900 --db speed3.dat 30:250:100
```

  Estrazione dei dati aggregati mediante la funzione *last* tra l'istante 1300 e l'istante 1900.

```
dani@gilean:r2dlib$ ./cli.py fetch --start 1300 --end 1900\
 --db speed3.dat --cf LAST

1300 None
1600 None
1900 ['30', '250', '100']

dani@gilean:r2dlib$
```

Sviluppo e Test

Lo sviluppo dell'applicazione è stato di tipo incrementale, seguendo lo schema del `test driven development`. Questo è un processo di sviluppo software agile che consiste nella ripetizione di cicli di sviluppo molto corti. Ognuno di questi cicli si compone di alcune fasi:

- La scrittura di un test, che definisce e mette alla prova la funzionalità che si vuole implementare. È importante verificare che inizialmente questo test fallisca, in modo da validarlo.

- L'implementazione della feature desiderata, fino al superamento del test precedente.
- L'eventuale refactoring del codice.

Strumenti utilizzati

Per la realizzazione di questo progetto sono stati utilizzati alcuni strumenti che ora verranno elencati e brevemente descritti.

pep8

`Pep8` è un tool che permette di verificare che le linee guida riguardanti lo stile, consigliate nella *Python enhancement proposal* numero otto, siano rispettate nel codice.

pylint

`Pylint` è un analizzatore statico di codice Python, che oltre a verificare che siano rispettati molti standard di codifica, permette di trovare automaticamente alcuni tra i bug più comuni.

py.test

La componente `py.test` di `py.lib` è un tool di unit testing contenuta nella suite `py lib`.

```
dani@astinus:~/code/r2dlib$ py.test
session starts
python: platform linux2 -- Python 2.6.6 -- pytest-1.2.1
test object 1: /home/dani/code/r2dlib

test_10_toydb.py .
test_20_fetch.py ...
test_30_ni_last.py ....
test_40_gauge.py .....
test_50_average.py ...
test_60_max.py ..
```

```
test_70_sum.py .
test_80_counter.py ..
test_90_xff.py .....
test_91_persistence.py .
test_92_total.py ..

29 passed in 2.57 seconds
```

argparse

`Argparse` è una libreria di Python che permette di effettuare in maniera molto semplice il parsing della linea di comando.