SCP - Scuola di Calcolo Parallelo - Scheduler per programmi paralleli

Mattia Sessolo
I.T.I.S. V.Volterra
San Donà di Piave

2006-2007

Introduzione

Questo programma è stato ideato per facilitare e automatizzare l'esecuzione di alcuni programmi paralleli (job) su un cluster di pc attraverso la gestione di una coda per uno scheduling del tipo FCFS (First Come First Served).

Esso può funzionare in due diverse maniere: quella base e quella completa. La sua versione base non necessita di un database ma solo di un cluster funzionante; il programma agisce prelevando i file di input (che dovranno essere opportunamente formattati per contenere le informazioni necessarie allo scheduler stesso), analizzandoli per capire di cosa si tratta e poi inserendoli in una coda. Una volta eseguito il job, il suo output viene prodotto sotto forma di due file: uno contenente le informazioni relative al tempo di esecuzione e lo stato di uscita del job (errore o terminazione corretta) e uno contenente i risultati effettivi del programma.

Sarà poi lo scheduler stesso ad eseguire tutte le operazioni di routine come verificare il numero di pc che sono effettivamente attivi nel cluster o attivare l'anello nel caso non sia in esecuzione.

La versione completa offre in più, oltre alle funzionalità descritte in precedenza, anche la possibilità di inserire in un database tutte le informazioni prodotte o necessarie allo scheduler come lo stato (coda dei processi in attesa in tempo reale) o il numero e le caratteristiche di tutti i job passati al programma.

L'utilizzo di un database consente inoltre includere nel sistema un meccanismo di autenticazione mediante login e password e anche una lista dei programmi disponibili necessari alla protezione dello scheduler da eventuali abusi da parte di soggetti o programmi indesiderati. Inoltre, per una rapida configurazione e/o selezione della modalità , sarà sufficiente modificare le scheduler.conf che sarà dettagliatamente descritto in seguito.

Capitolo 1

Installazione & Configurazione

1.1 Prerequisiti

Per poter utilizzare lo scheduler(almeno nella sua versione base) sono necessari alcuni requisiti fondamentali:

- Un cluster perfettamente funzionante
- Python 2.3 o superiore(essendo il programma scritto in python)

I requisiti per utilizzare lo scheduler al pieno delle sue funzionalità sono, oltre al necessario per la versione base anche:

- Un server Mysql
- Libreria per Python MySQLdb(installata e funzionante necessaria per l' interfacciamento di Python con il database mysql).
- Ovviamente un minimo di conoscenza sui database...

1.2 Installazione

1.2.1 Versione base

Per installare lo scheduler nella sua vesione base è necessario scompattare l'archivio .tar.gz nella directory di installazione con il comando *tar xfsz scheduler.tar.gz*

e una volta estratto l'archivio, è richiesta la confgurazione del file scheduler.conf con i seguenti parametri(gli altri possono essere lasciati vuoti in quanto non necessari per la versione base):

```
IN
OUT
EXE
MPDHOST
DB=NO
```

Con i parametri così inseriti lo scheduler è già pronto all' uso nella sua versione base... Passare alla sezione *Istruzioni per l' uso* per sapere come farlo funzionare...

1.2.2 Versione database

Per installare questa versione bisogna inanzitutto scompattare l' archivio .tar.gz nella directory desiderata per l' installazione con il comando *tar xfsz scheduler.tar.gz*

Per installare il database necessario al funzionamento dello scheduler è opportuno verificare che il server MySQL sia attivo, quindi procedere alla creazione del database eseguendo lo script CREASTRU.SQL

Una volta completata, senza errori, l' installazione del database è sufficiente editare il file scheduler.conf con i seguenti parametri:

```
IN
OUT
EXE
MPDHOST
DB=SI
USER
DB\_NAME
PASSWD
HOST
```

Capitolo 2

Istruzioni per l'uso

2.1 scheduler.conf

scheduler.conf è il file di configurazione dello scheduler che consente di impostare il programma a piacimento: questo file è costituito da diverse righe contententi i seguenti campi che dovranno poi essere editati per poter utilizzare correttamente il programma:

- **IN=** cartella dove si trovano i file di input;
- **OUT=** cartella dove si trovano i file risulato dell' elaborazione;
- **EXE=** cartella dove si trovano gli eseguibili dei programmi(la cartella condivisa in rete dove si mettono gli eseguibili);
- **MPDHOST=** path del file mod.hosts;
- **DB=** in questo caso si hanno due possibilita o SI o NO; nel caso di SI si utilizzerà la versione avanzata dello scheduler(con database) altrimenti la versione base;
- **DB_NAME=** nome del database;
- **HOST=** indirizzo IP del computer dove è attivo un server mysql;
- **USER=** username per poter accedere al database;
- **PASSWD=** password per accedere al database;

Con il carattere # posto all' inizio di una riga si identificano i commenti. Dopo il carattere uguale si possono inserire le directory o parametri necessari: per la versione base dello scheduler bastano i primi 4.

Esempio di scheduler.conf

```
\#File di configurazione dello scheduler
\#cartelle di I/O
IN=/home/scp/in/
OUT=/home/scp/out/
EXE=/scambio/
\# percorso mpd.host
MPDHOST=/home/scp/mpd.hosts
\#presenza db
DB=SI
\#parametri di connessione
HOST=172.16.47.45
USER=scp
PASSWD=scp123
nome database
DB\_NAME=scp
```

Caratteristiche dei programmi paralleli

I programmi paralleli, per poter essere utilizzati devono essere scritti con degli accorgimenti:

- Output su un file chiamato OUT(case-sensitive)
- Eventuali parametri in ingresso

Far pervenire allo scheduler i parametri tramite un file che deve essere indicato come parametro: Es: [nome_eseguibile] [par_1...par_n] [path del file input]

2.1.1 Versione base

Come detto in precedenza, lo scheduler riceve come input per poter iniziare il lavoro, un file contentente le seguenti righe: *[nome eseguibile] par1 par2... timeout*

Il primo campo serve a dire allo scheduler qual'è il programma(già compilato) da eseguire seguito dai suoi parametri(superfluo dire che l' eseguibile dovrà essere posto nella cartella specificata nel campo EXE nel file di configurazione) mentre la seconda riga contiene il timeout entro il quale, se il programma non è terminato varrà killato. Esempio di file di input:

nome_eseguibile par1 par2 par3 ... 10

Una volta messo un file di questo tipo nella cartella IN lo scheduler lo leggerà, capirà di cosa si tratta e lo metterà in coda per l' esecuzione, non

appena il programma verrà eseguito e terminato, verranno inseriti nella cartella OUT i file risultato del lavoro che avranno come nome quello del file di input(ATTENZIONE! Se viene messo nella cartella IN un file con un nome già presente in OUT il file presente nella cartella OUT viene sovrascritto); il file [nome_eseguibile].log contiene alcune informazioni sul job(data inizio e data di fine job, exit status...), il file [nome_eseguibile].res contenente il risultato del job.

2.1.2 Versione con database

Inanzitutto diamo una rapida occhiata al database necessario per poter usare questa versione dello scheduler(La versione del database qui descritta è una versione minimale che potrà essere modificata a piacere con l' accorgimento di non cambiare i campi qui descritti)

Tabella PROGRAM: contiene i nomi dei programmi che è possibile mandare in esecuzione nel cluster:

```
PR\_NAME : nome programma
Tabella USER: contiene nome utente e password degli utenti autorizzati a
utilizzare i programmi nello scheduler:
US\_ID: intero auto\_increment, chiave primaria che identifica un utente
US\_LOGIN: username dell' utente;
US\_PASSWD: Password dell' utente.
Tabella JOB: contiene lo storico dei programmi:
JOB\_ID: intero che identifica univocamente un job;
JOB\_US\_ID: chiave che referenzia l' utente che ha eseguito il job,
referenzia il campo US\_ID di JOB
JOB\_PR\_NAME: chiave che identifica il programma eseguito, contenuto nella
tabella PROGRAM
JOB\_NAMEIN: Nome del file di input;
JOB\_NAMEOUT: Nome del file di output;
JOB\_TIMEOUT: Timeout del job;
JOB\_RESULT: Exit status del JOB
JOB\_STARTTIME: Stringa di tipo datetime(AAAA:MM:DD HH:MM:SS) che
rappresenta la data e ora di inizio elaborazione
JOB\_ENDTIME: Stringa di tipo datet('ping\_pong.scp'),ime che rappresenta la
data e ora di fine job
JOB\_EXTIME: Float corrispondente alla durata dell' esecuzione
Tabella TAIL: contiene la coda dello scheduler cioè una rappresentazione
della tabella JOB, tolti i campi JOB\_ENDTIME,JOB\_RESULT,JOB\_EXTIME
poichè vengono direttamente scritti sullo storico(tabella JOB)...Di
```

seguito elenco i campi dalla tabella TAIL(cambia il nome ma hanno la stessa funzione di quelli della tabella JOB):
TA_ID: chiave esterna che referenzia il campo JOB_ID;
TA_US_ID;
TA_PR_NAME;
TA_NAMEIN;
TA_NAMEOUT:
TA_TIMEOUT;
TA_STARTDATE;
Ordinando in modo crescente i record di questa tabella secondo il campo TA_ID otteniamo la coda dello scheduler dove, il primo record rappresenta il job attualmente in esecuzione.

Lo scheduler riceve come input per poter iniziare il lavoro un file contentente le seguenti righe:

- nome eseguibile
- Username per accedere allo scheduler
- password per accedere allo scheduler
- timeout
- Eventuali parametri del programma passati riga sotto riga

Esempio di file di input:

```
crivex.scp
mario.rossi
raperonzolo
10
par1
par2
....
```

Una volta letto il file, il job viene messo in coda e la coda viene rappresentata nella tabella TAIL; Una volta completato il job, viene cancellato il record corrispondente a quel job dalla tabella TAIL e viene completata la tabella dello storico(inserendo i tempi di elaborazione e altre informazioni...). Ovviamente, come per la versione base, vengono scritti i file .res e .log. Per inserire nuovi utenti autorizzati all'utilizzo dello scheduler si deve inserire nel database le nuove informazioni con la query:

```
INSERT INTO USER(US\_LOGIN,US\_PASSWDORD) VALUES ("username","password");
```

Mentre, per inserire nuovi programmi:

```
INSERT INTO PROGRAM(PR\_NAME) VALUES ("prog.scp")
```

Ora, il file scheduler.conf è stato modificato correttamente si è pronti per avviare lo scheduler con il comando:

```
python scheduler.py &
```

In ambedue le versioni per far terminare l' esecuzione dello scheduler sarà sufficiente digitare da console:

```
killall python
```

2.2 Un po più di dettagli...

Lo scheduler è composto da due thread che lavorano contemporaneamente:lettore ed esecutore. Il primo ha il compito di leggere ogni 4 secondi la cartella dove risiedono i file di input e, nel caso che ci siano nuovi file, li analizza e li mette nella coda; il secondo invece è la parte di scheduler che si preoccupa di eseguire i job nella coda e di scrivere i risultati in output. Il programma è costituito di varie sezioni: **conf.py**Questo file contiene la procedura per analizzare il file scheduler.conf e, restituire sotto forma di dizionario tutti i parametri che poi verranno analizzati dai thread; **l2.py** Questo file contiene il thread che si preoccupa di analizzare i file in input:

Non appena viene avviato il thread, viene scelta la modalità da avviare in base al dizionario di parametri passati all'avvio(da conf.py): se DB=NO allora viene avviata la versione senza database, quindi ogni 4 secondi viene scandita la cartella IN e, se ci sono file presenti, vengono aperti e letti i parametri contenuti all' interno del file e, se la la sintassi del file è corretta, il job viene messo in coda. Se invece siamo nel caso in cui è richiesta la presenza del database, prima di mettere il job in coda vengono eseguiti alcuni controlli tramite il database per verificare che l' utente abbia i permessi per eseguire quei determinati programmi; se tutti i controlli sul database hanno esito positivo allora viene scritto nella tabella JOB e nella TAIL un record contenente tutte le informazioni sul job, altrimenti, se non si hanno i permessi, viene scritto un record nella tabella in cui troviamo la descrizione degli errori(ovviamente ci sono anche i file di log)

coda.py

Definisce una classe con un attributo statico cioè una lista che implementa la coda con cui i due thread operano.

e2.py

Thread che si occupa dell' esecuzione del job: preleva dalla coda il job in testa e lo esegue, nel frattempo che il file è in esecuzione, questo thread si preoccupa di controllare se il programma parallelo ha superato il timeout, se si verifica tale condizione lo killa e alla fine di ciò scrive i due file di output. Nella versione database inoltre aggiorna la tabella JOB e TAIL all' inizio del job per poi scrivere i risultati finali e eliminare dalla tabella TAIL il record corrispondente al job appena terminato.

ringstat.py

Classe che rappresenta il cluster: questa classe si preoccupa di controllare di quanti pc è composto il cluster in modo da determinare quali sono effettivamente utilizzabili. Poi si preoccupa di alzare e abbassare il cluster e infine di verificare lo stato dello stesso.