SCP: SCHEDULER LAYER

a cura di

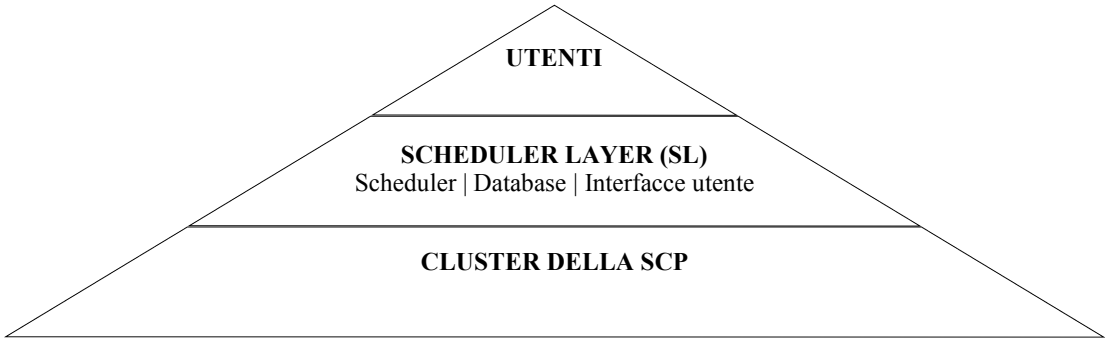
Alberto Boccato

PREMESSA:

Negli ultimi tre anni la nostra scuola ha portato avanti un progetto al quale ho partecipato chiamato SCP (Scuola di Calcolo Parallelo). Di fatto si è trattato di un corso pomeridiano di 3 ore settimanali in cui ci sono state illustrate le basi del calcolo parallelo e ci è stato permesso di provare "sul campo" quanto appreso attraverso lo sviluppo di alcuni interessanti programmi. Superficialmente, il termine "calcolo parallelo" si riferisce alla possibilità di distribuire i calcoli contenuti in un singolo programma a più calcolatori connessi tra loro in modo che collaborino parallelamente nel tempo per il raggiungimento del risultato finale, in tempi molto più rapidi rispetto alle possibilità del singolo. La scuola ha così messo a nostra disposizione alcune vecchie macchine e uno switch per connetterle fra loro. Ci siamo quindi costruiti un piccolo cluster dove poter testare i nostri lavori e per far questo ci siamo serviti di software completamente open source, in particolare delle MPICH2, delle speciali librerie che permettono di sviluppare programmi paralleli.
Durante il primo anno gli obiettivi della SCP sono stati la predisposizione del cluster e lo sviluppo di alcuni semplici programmi come la stima di pi-greco o la generazione di numeri pseudo casuali sfruttando i metodi di Montecarlo. Nel secondo anno, poiché il progetto era già avviato, alcuni ragazzi hanno potuto dedicarsi allo studio di problemi veramente complessi, primo fra tutti la generazione di un orario per la nostra scuola. Un secondo gruppo di nuove reclute invece, dopo un'infarinatura generale di linguaggio C e fondamenti di calcolo parallelo, ha potuto sviluppare altri interessanti programmi. Fra questi ricordiamo il crivello di Eratostene per il conteggio dei numeri primi e la previsione della popolazione italiana in un certo istante di tempo sfruttando la matrice di Lesile come modello di sviluppo della popolazione. Infine in quest'ultimo anno è nata la necessità di riorganizzare e documentare il tutto, ed è proprio per questo che abbiamo deciso di creare un sistema che gestisce l'accesso degli utenti al cluster. Il fine è simulare il funzionamento di un cluster reale come potrebbe essere quello di un centro di ricerca dove i vari enti, società o privati richiedono l'esecuzione dei loro programmi. Allo stesso tempo, in questo modo, possiamo mostrare i lavori da noi svolti nel corso dei tre anni.

PANORAMICA SUL SISTEMA DI ACCESSO E GESTIONE DEL CLUSTER:

Dunque ciò di cui abbiamo bisogno è uno strato che permetta a più utenti di comunicare contemporaneamente con il cluster e richiedere l'esecuzione di programmi mascherando i problemi di attivazione e gestione del cluster stesso. Abbiamo deciso di chiamarlo Scheduler Layer (SL). Ecco un diagramma per rendere meglio l'idea.



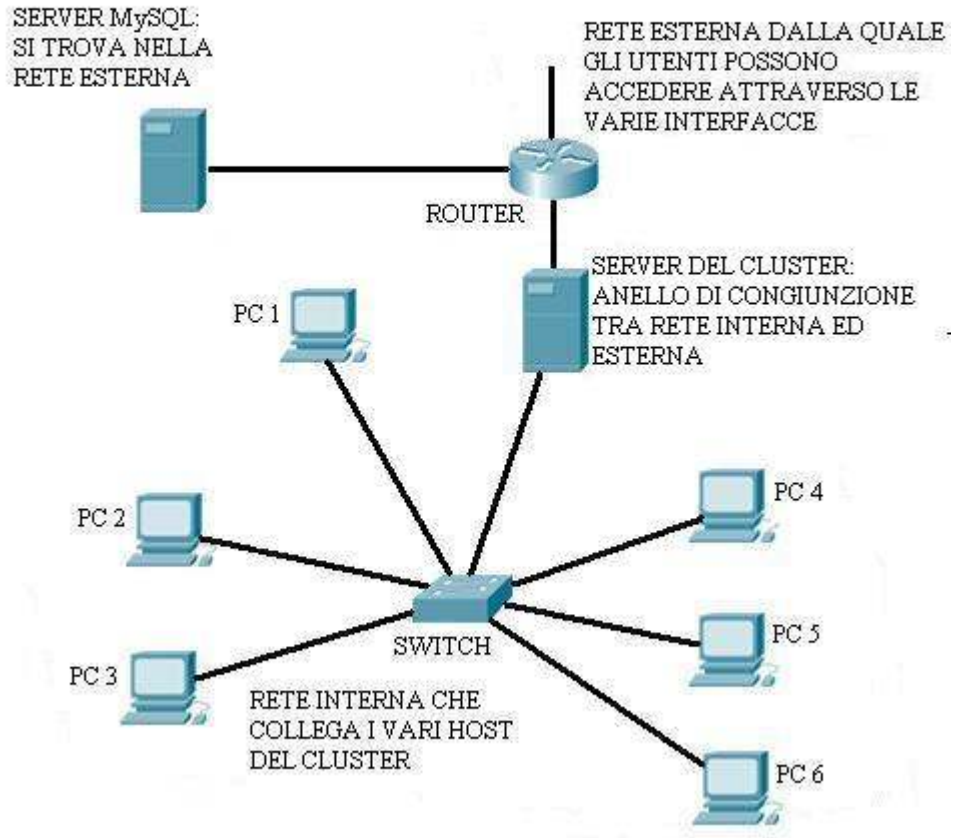
Come si può osservare lo Scheduler Layer è composto fondamentalmente da tre parti:

- lo scheduler ossia il software incaricato di:
  - ricevere le richieste di esecuzione di programmi provenienti dalla rete esterna al cluster;
  - gestirle secondo una disciplina di tipo FIFO (first in first out);
  - avviare il software di comunicazione del cluster che permette l'esecuzione parallela verificando quali sono le macchine attive;
  - lanciare l'esecuzione parallela del programma richiesto dall'utente;
  - controllare il corretto funzionamento del programma e tenere traccia di eventuali malfunzionamenti;

  lo scheduler è completamente indipendente da database e interfacce utente perciò funziona anche senza il loro supporto
- il database dove sono contenute le informazioni riguardanti gli utenti e i relativi permessi, i programmi a disposizione, i lavori eseguiti (che da ora in poi chiameremo job) e i lavori in coda;
- le interfacce utente infine sono i diversi software che permettono agli utenti di inoltrare le proprie richieste dalla rete esterna.

UN'OCCHIATA ALLA STRUTTURA FISICA DELLA NOSTRA RETE:

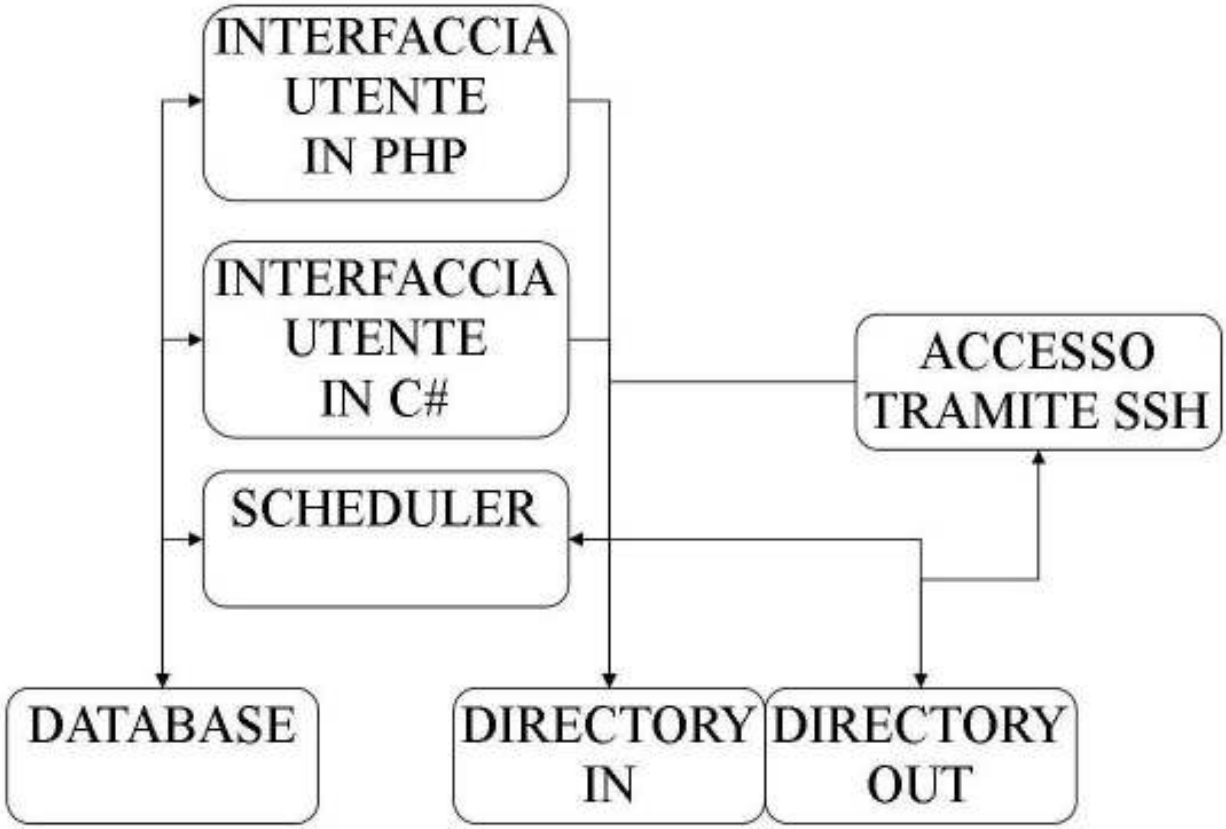
Le scelte a livello implementativo dipendono anche dall'organizzazione fisica dell'hardware dunque è necessario descriverla in modo rapido.



Come si osserva dalla figura disponiamo di una rete interna, identificata dall'indirizzo 172.16.20.0, e una esterna identificata dall'indirizzo 172.16.47.0. Queste comunicano tra di loro attraverso il server del cluster che è dotato di due interfacce di rete.
La rete interna è costituita dai vari host del cluster sui quali è installato il software che permette l'esecuzione di programmi in parallelo. Quella esterna, invece, comprende il server in cui è presente il database MySQL ed è la rete a cui gli utenti devono collegarsi per poter inoltrare la loro richiesta.

FUNZIONAMENTO E IMPLEMENTAZIONE DELLO SCHEDULER LAYER:

Schema della struttura dello scheduler layer.

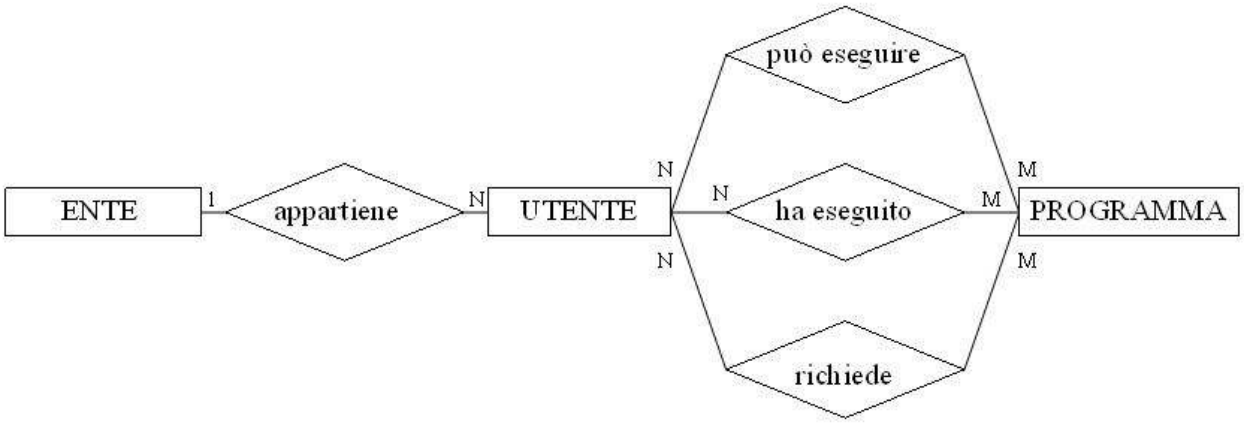


Come abbiamo già detto lo Scheduler Layer è composto fondamentalmente da tre elementi: il database, lo scheduler e le interfacce utente.
Lo *scheduler* è un programma scritto in python sempre attivo sul server del cluster che può funzionare in due modalità: con database oppure senza. In ogni caso è composto di 2 thread. Il primo controlla ad intervalli di tempo regolari una directory che chiameremo "IN" così da verificare se si sono presentate nuove richieste di lavoro. La richiesta consiste in un file opportunamente formattato dove compaiono il nome dell'eseguibile da lanciare, username e password dell'utente che fa la richiesta ed eventuali parametri necessari al programma. Dunque se si presenta un file in questa directory lo scheduler se ne accorge e lo elimina dopo aver inserito il relativo job in coda. La coda può essere una tabella del database o una lista in ram, dipendentemente dalla modalità in cui lo scheduler sta funzionando. Mentre il primo processo fa questo, il secondo, contemporaneamente, si occupa di controllare la presenza di job in coda e, dopo aver verificato quali sono le macchine disponibili, lancia la sua esecuzione. Quando un lavoro viene completato, lo scheduler va ad aggiungerlo ad uno storico dei job, incaricandosi inoltre di verificarne la corretta esecuzione ed in caso contrario segnalarlo. I risultati dei programmi paralleli vengono scritti in un file nella cartella OUT così come i file di log, che contengono delle informazioni relative all'esecuzione ed a eventuali malfunzionamenti.

Il *database* è di supporto sia allo scheduler (se questo è abilitato per usarlo), sia alle interfacce grafiche. Abbiamo scelto di usare MySQL come DBMS in quanto è open-source e permette di gestire connessioni multiple. Dobbiamo supporre, infatti, che più utenti si possano connettere al nostro sistema contemporaneamente simulando un cluster di un centro di ricerca. Per poter eseguire un programma, perciò, è necessario essere registrati nella tabella degli utenti e disporre così di un utente e una password per l'accesso. Per ogni utente vengono inoltre salvati i dati fiscali: della persona se si tratta di un privato, dell'ente se si tratta soltanto di un dipendente di un'azienda. Bisogna inoltre tener traccia dei programmi a disposizione (cioè quelli da noi svolti nel corso degli anni) e dei permessi di esecuzione che ogni utente ha. Inoltre vogliamo mantenere uno storico dei job in modo che un utente possa successivamente visionare i lavori da lui eseguiti in precedenza con i relativi dettagli. Infine si vuole mantenere la coda dei job, che sarà di uso esclusivo dello scheduler. Quando un job termina, infatti, lo scheduler si occupa di eliminato dalla coda e inserirlo nello storico.
Infine abbiamo deciso di implementare delle *interfacce utente* che devono permettere all'utente di inviare le proprie richieste nel modo più semplice e intuitivo possibile, mascherando come questo avvenga effettivamente, ed eventualmente di visualizzare e modificare i propri dati personali. Come si può vedere dallo schema precedente abbiamo realizzato un'interfaccia via browser in PHP e una in C#. Da qui è nata la necessita di installare e configurare un web server (Apache) per la prima, e un server FTP per la seconda (dato che è l'unico modo per copiare file da windows a linux usando C# senza dover installare altri software o librerie). La locazione di queste due componenti è di nuovo il server del cluster poiché è qui che lo scheduler andrà a controllare la presenza di nuove richieste.

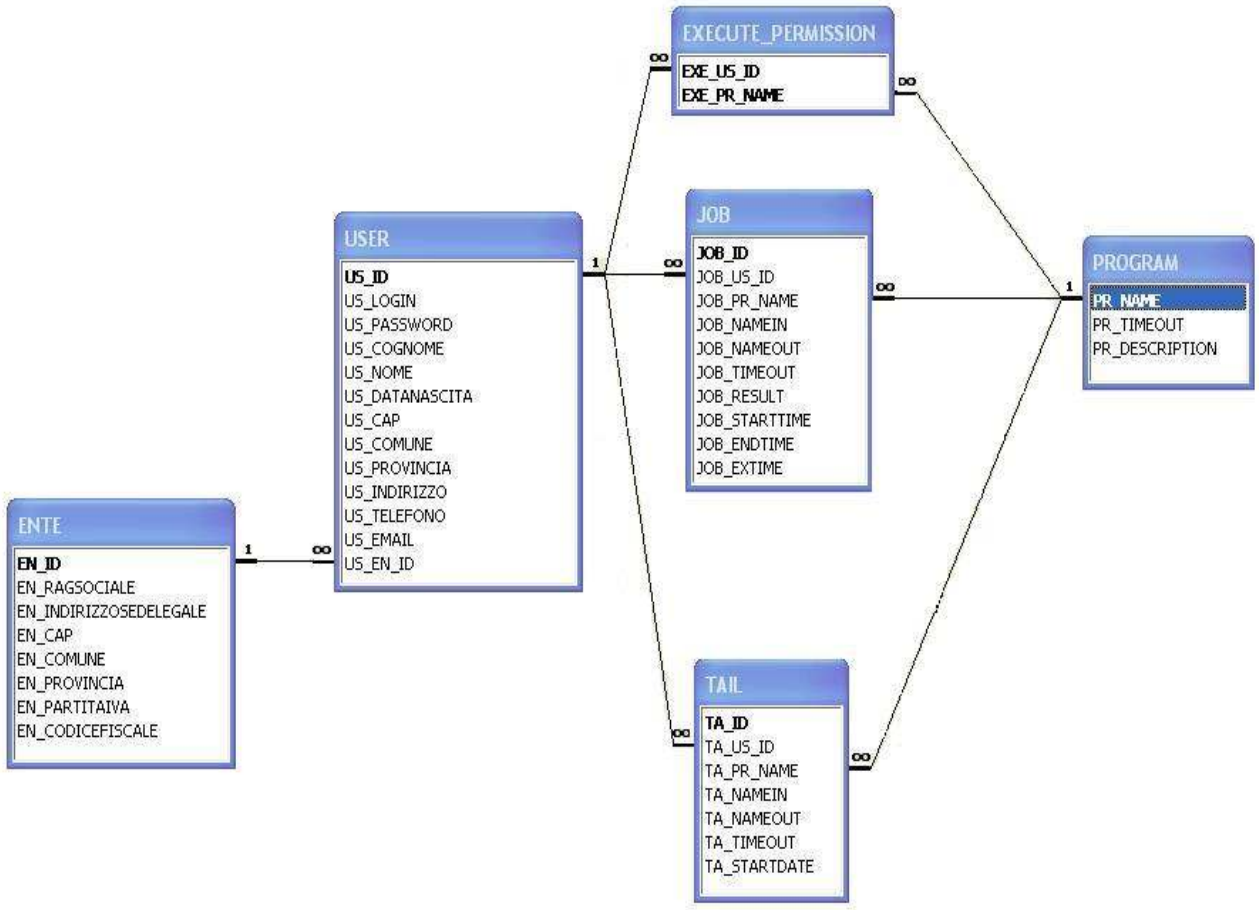
DETTAGLI DATABASE:

Passiamo ora ad esaminare i dettagli del database. Lo schema concettuale che rappresenta la nostra realtà è il seguente:



Come si può vedere le entità identificate sono tre: ente, utente e programma. L'entità ENTE rappresenta i dati fiscali degli utenti. Un utente può appartenere ad un solo ente ed un certo ente può comprendere più utenti. Dunque sussiste una relazione uno a molti. Nel caso un utente sia un privato nell'entità ENTE compariranno comunque i suoi dati fiscali.
L'entità PROGRAMMA identifica i programmi paralleli dai noi svolti nel corso degli anni. Fra UTENTE e programma sussistono 3 relazioni molti a molti. Dalla relazione "può eseguire" risulteranno i permessi di esecuzione dei vari utenti mentre dalle relazioni "ha eseguito" e "richiede" risulteranno rispettivamente lo storico dei lavori svolti e la coda dei lavori in attesa. Come abbiamo già detto, l'idea è che nel momento in cui si presenta una nuova richiesta, questa viene messa in coda e successivamente eliminata al momento dell'esecuzione, divenendo infine un elemento dello storico appena l'elaborazione ha termine.
Vediamo ora gli attributi delle varie entità. Per quanto riguarda l'ENTE abbiamo individuato i seguenti attributi: id (chiave primaria), ragione sociale, indirizzo sede legale, CAP, comune, provincia, partita IVA, codice fiscale. Per l'UTENTE: id (chiave primaria), login, password, cognome, nome, data di nascita, CAP, comune, provincia, indirizzo, numero di telefono, e-mail. Per l'entità PROGRAMMA: nome (chiave primaria), timeout, descrizione.
Infine per le relazioni "ha eseguito" e "richiede" abbiamo individuato alcuni attributi di relazione. Per quanto riguarda "ha eseguito": id (chiave primaria), nome del file di richiesta, nome del file di output, timeout del programma, stato del lavoro (ad esempio se è stato completato correttamente oppure no), data di inizio, data di ultimazione, stima del tempo impiegato. La relazione "richiede" dato che mantiene traccia della coda dei lavori presenta attributi molto simili: id (chiave primaria), nome del file di richiesta, nome del file di output, timeout del programma, data di inizio. Come si può vedere c'è una ridondanza nei dati ma questo è necessario poiché in caso di malfunzionamenti le informazioni relative alle richieste devono essere presenti anche nella coda.

Il codice SQL per la creazione delle tabelle potete trovarlo all'interno del sito dunque presento soltanto il modello logico in forma grafica:

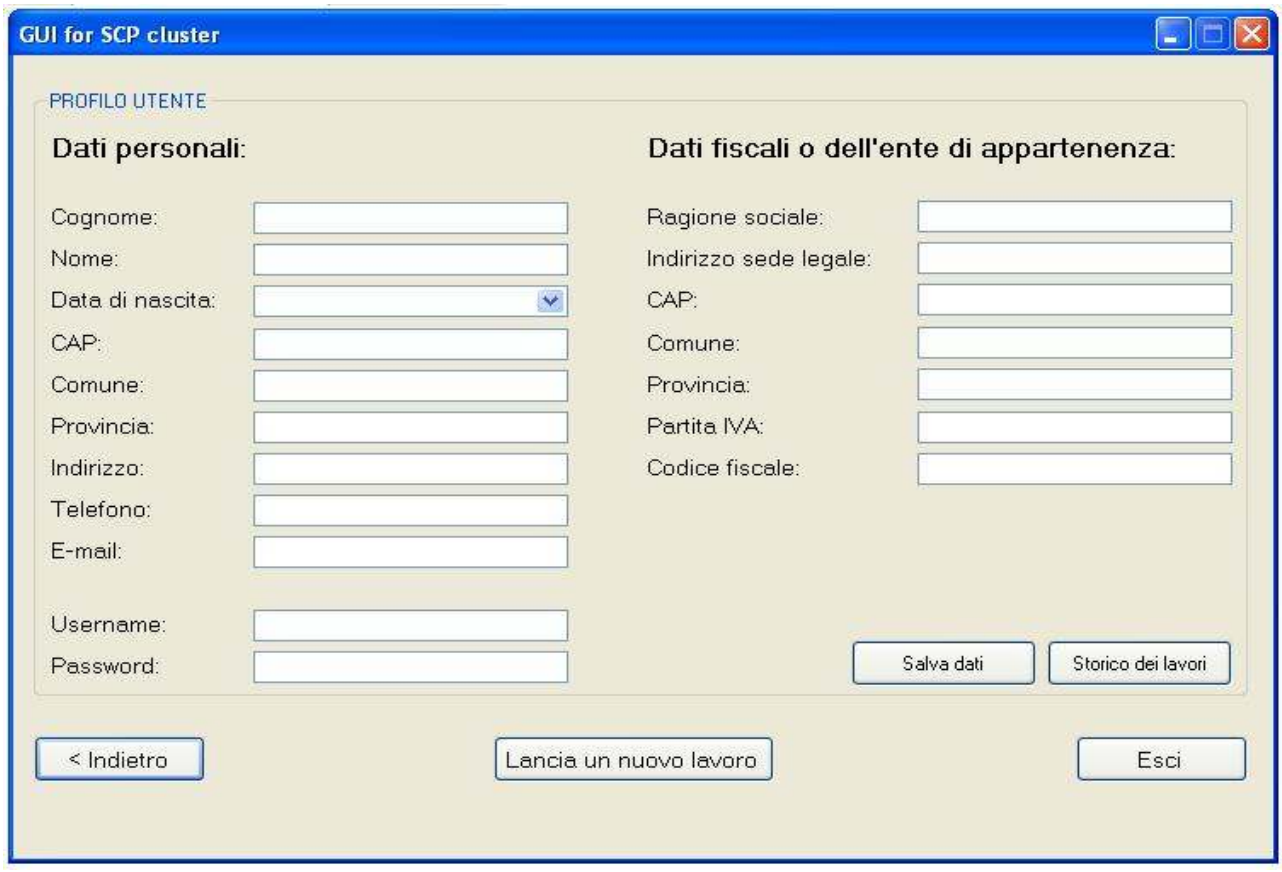


Come si osserva è stato deciso per convenzione che i primi caratteri di ogni campo ne identificano la tabella di appartenenza. Dalle tre relazioni molti a molti sono nate tre tabelle EXECUTE_PERMISSION (relativa ai permessi), JOB (lo storico dei lavori) e TAIL (la coda dei lavori). Sempre all'interno del sito potrete trovare il codice SQL che permette di inserire dei dati di partenza come i lavori disponibili e gli utenti indispensabili allo scheduler per funzionare. Per informazioni più specifiche riguardo allo scheduler vi rimando alla sezione relativa.

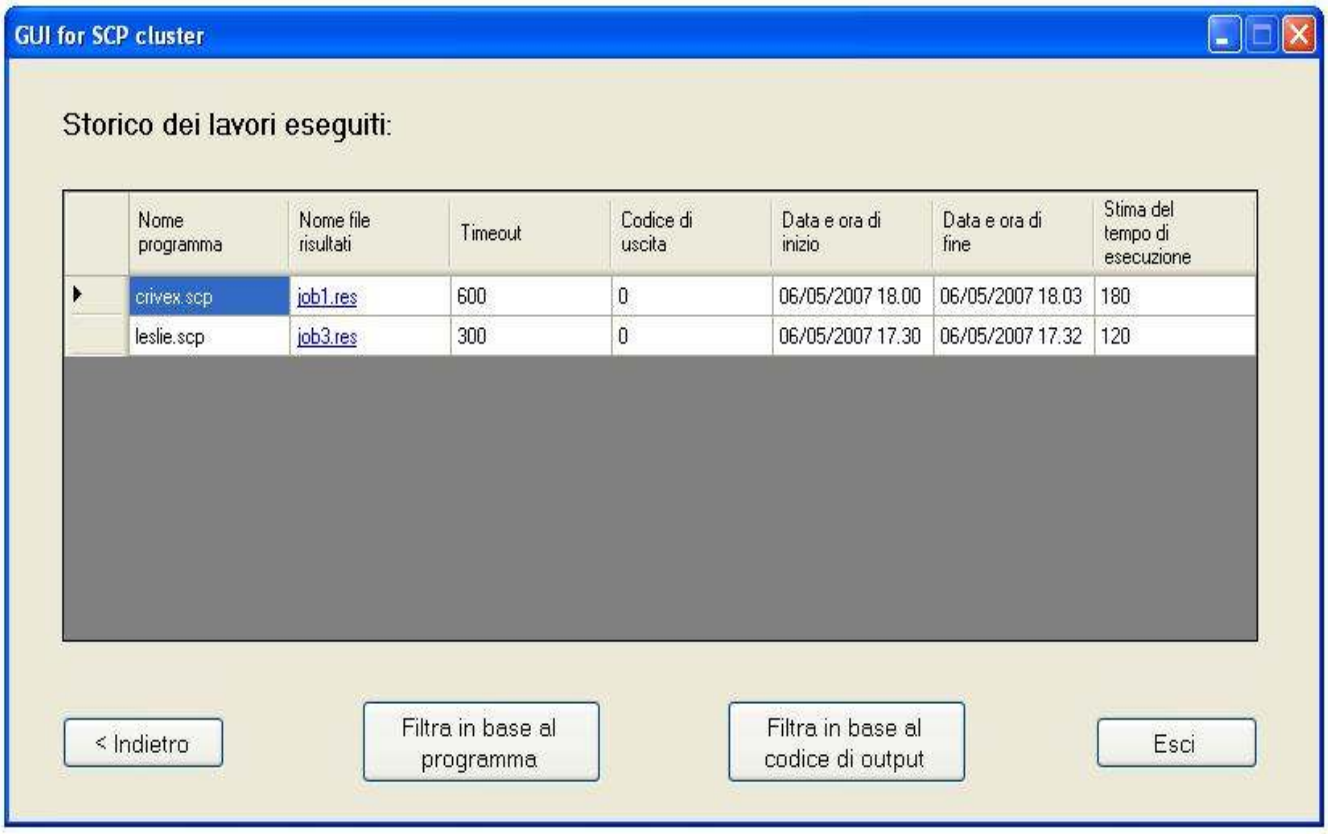
DETTAGLI INTERFACCIA C#:

L'interfaccia in C# richiede che nel computer dell'utente sia installato il framework ".Net 2.0" e il "Mysql Connector" (la nostra versione è 5.0.6). Una volta che si dispone di queste componenti si può avviare il file eseguibile GUIforSCPcluster.exe . Con la prima schermata di login viene interrogata la tabella USER del database per verificare l'esistenza dell'utente e la corrispondenza della password inserita.

A questo punto interrogando le tabelle USER ed ENTE all'utente viene visualizzato il proprio profilo composto dei dati personali e dei dati fiscali. È inoltre implementata la possibilità di modificarli.

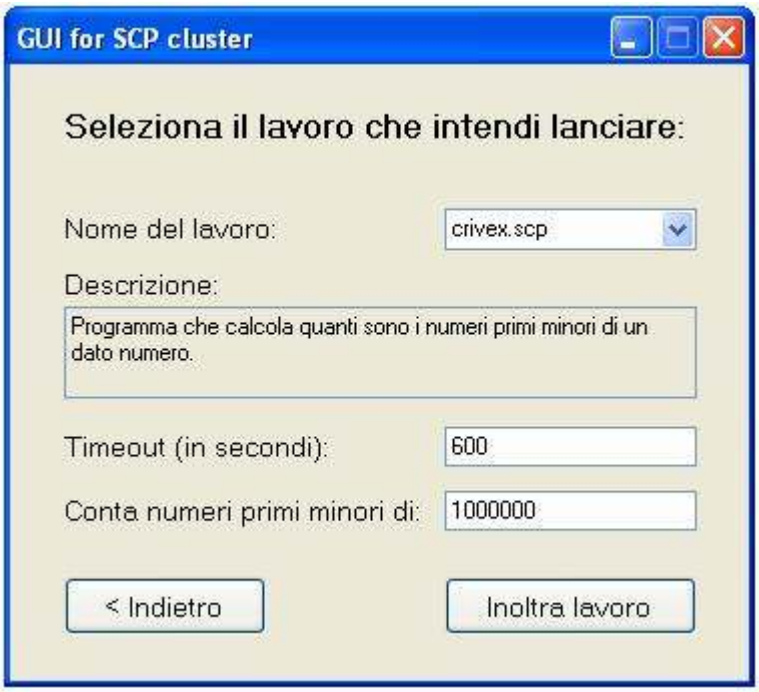


L’utente può ora decidere se visionare lo storico dei lavori oppure lanciarne uno nuovo. Nel primo caso si apre una schermata dove è presente una griglia contenente i precedenti lavori eseguiti e i relativi dettagli andando ad interrogare la tabella JOB. L’utente non ha qui la possibilità di modificare i dati poiché, come è logico pensare, un centro di ricerca potrebbe voler fare successivamente delle indagini su di essi.



Inizialmente erano state pensate anche alcune funzionalità avanzate come la possibilità di filtrare i dati o di aprire i risultati ciccando sul nome del file dei risultati. Purtroppo la mancanza di tempo ci ha impedito di completare queste migliorie.

Nel secondo caso, si passa ad una nuova finestra che permette di selezionare il programma da eseguire in base ai permessi posseduti dall’utente ed inoltrare la richiesta attraverso il protocollo ftp al server del cluster nella cartella “IN”. Viene dunque interrogata la tabella EXECUTE_PERMISSION in modo da ottenere la lista di tutti i programmi disponibili all’utente attuale e successivamente eseguito un controllo sulla tipologia del programma. Se questo richiede dei parametri, come nel caso del crivello di Eratostene, compare una textbox dove questi vengono richiesti. Infine il bottone “Inoltra lavoro” permette di creare il file di richiesta nella directory da dove l’applicazione è stata lanciata, copiarlo via ftp nel server del cluster e successivamente eliminarlo.



È dunque l'interfaccia che si occupa di creare il file di richiesta, formattarlo in modo che lo scheduler possa successivamente gestirlo, e copiarlo nella directory "IN". A questo punto lo scheduler si prenderà carico della richiesta e l'utente potrà trovare il file contenente i risultati nella cartella "OUT" appena gli eventuali lavori in coda saranno completati.

Un ringraziamento speciale va alle matricole della SCP per il loro apporto alla realizzazione del progetto. In particolare a Sessolo Mattia, sviluppatore dello scheduler, e Peretti Giulio, creatore dell'interfaccia in php.