JSync

Progetto di Laboratorio di Sistemi Operativi A.A. 2014-2015

Saverio Giallorenzo

1 Descrizione

JSync è un programma di sincronizzazione di files. Un sistema distribuito JSync è composto da Clients e Servers. I files vengono raggruppati in repositories.

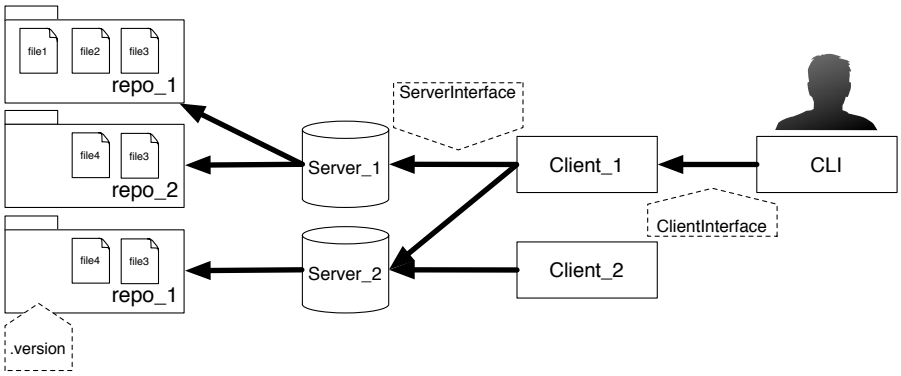

Figure 1: Rappresentazione di un sistema distribuito JSync

Un Server può:

- creare un nuovo repository;
- ritornare i files in un repository;
- aggiornare i files di un repository esistente;
- cancellare un repository (e i relativi files).

Un Client può:

- aggiungere un nuovo Server alla propria lista di Servers registrati;
- rimuovere un Server presente nella propria lista di Servers registrati;

- creare un nuovo repository su un Server registrato. Ad un nuovo repository corrisponde una directory locale (localDir). La localDir conterrà i files da sincronizzare col Server;
- registrare un repository già esistente su un Server. Il repository registrato corrisponderà ad una directory local (localDir). La localDir conterrà i files da sincronizzare col Server;
- mandare (`push`) un aggiornamento di un repository al Server;
- richiedere (`pull`) un aggiornamento di un repository al Server;
- cancellare un repository su un Server.

Per garantire la consistenza degli aggiornamenti, Servers e Clients mantengono una versione (`version`) del repository. Se un Client invia una `push` con una versione precedente del repository, il Server deve negare la scrittura e notificare il Client. La versione corrente di ogni repository è mantenuta in un file `.version` all'interno del repository stesso.

L'interazione con l'utente avviene tramite una Command-Line Interface (CLI) (implementato direttamente dal Client o tramite un servizio a parte.)

1.1 Interazione con l'utente

L'utente interagisce con il proprio Client tramite un'interfaccia a linea di comando. I comandi sono riportati in Tabella 1.

Ogni repository mantiene un file `.version` con l'ultima versione del repository (e.g., un intero). Le operazioni di `pull` e `push` aggiorneranno (incrementeranno) rispettivamente il `.version` locale e quello nel Server ospitante il repository.

E.g., per fare `push` di una nuova versione del repository:

- il Client deve avere una versione (`.version`) del repository aggiornata rispetto a quella presente sul Server;
- se si, il Client deve inviare i files della nuova versione;
- dopo aver ricevuto e sincronizzato i files, il Server ritorna al Client l'incremento di versione da scrivere in `.version`.

2 Consegna del Progetto

2.1 Implementazione

Il progetto deve essere sviluppato utilizzando il linguaggio Jolie.

Non ci sono requisiti riguardo ai protocolli (*protocols*) e i media (*locations*) utilizzati per realizzare la comunicazione tra Servers, Clients e CLIs.

Comando	Parametri	Effetto
`list servers`		visualizza la lista di Servers registrati
`list new_repos`		visualizza la lista di repositories disponibili nei Server registrati
`list reg_repos`		visualizza la lista di repositories registrati localmente
`addServer`	serverName serverAddress	aggiunge un nuovo Server alla lista dei Servers registrati
`removeServer`	serverName	rimuove "serverName" dai Servers registrati.
`addRepository`	serverName repoName localPath	se "repoName" non esiste sul server "serverName", il repository viene creato. Se non esiste, crea la directory "localPath". Infine viene registrato il repository localmente.
`push`	serverName repoName	fa `push` dell'ultima versione di "repoName" locale sul server "serverName"
`pull`	serverName repoName	fa `pull` dell'ultima versione di "repoName" dal server "serverName"
`delete`	serverName repoName	rimuove il repository dai repo registrati, cancella il repository "repoName" sul server "serverName" e la cartella locale corrispondente.

Table 1: Tabella dei comandi CLI di JSync

L'obiettivo dell'implementazione riguarda la gestione concorrente di risorse condivise (i files). Non è richiesta una gestione avanzata dei files, i.e., che versioni differenti dello stesso file siano in qualche modo unite (merge). L'importante è dimostrare che il comportamento implementato da Servers (e Clients) gestisca i tipici problemi di concorrenza su risorse condivise (i.e., Readers-Writers).

2.2 Interfacce e `push`

Di seguito si propone un possibile punto di inizio per l'implementazione dei Servers e dei Clients. La proposta è un *draft* (può essere incompleta e contenere errori e imprecisioni) e si limita a 1) la descrizione delle interfacce del Server (implementate da Servers e Clients) e del Client (implementate dal Client e la CLI) e 2) una breve discussione su come orientare lo sviluppo del comando di `push`.

2.2.1 Interfacce

```
type RepoType: void {
  .repoName*: string
}

type FilesType: void {
  .repoName: string
  .version: int
```

```
  .file*: void {
    path: string
    content: raw
  }
}

type UpdateRequest: void {
  .version: int
}

interface ServerInterface {
  RequestResponse:  getRepositories( void )( RepoType )
                    createRepository( RepoType )( void )
                    pull( RepoType )( FilesType )
                    push( PushRequest )( FilesType )

  OneWay:           pushRequest( UpdateRequest )
                    deleteRequest( UpdateRequest )
}

type AddServerType: void {
  .serverName: string
  .address: string
}

type AddRepoType: void {
  .serverName: string
  .address: string
  .localPath: string
}

type CommandResponse: void {
  .response: string
}

type RepoRequestType: void {
    .serverName: string
    .repoName: string
}

interface ClientInterface {
  RequestResponse:  addServer( AddServerType )( bool )
                    listNewRepos( void )( RepoType )
                    listRegRepos( void )( RepoType )
                    listServers( void )( RepoType )
                    addRepository( AddRepoType )( bool )
                    removeRepository( AddRepoType )( bool )
                    pull( RepoType )( response )
                    push( RepoType )( response )
                    delete( RepoType )( response )
}
```

2.2.2 Reading-Writing files

Uno degli argomenti di discussione per lo sviluppo del progetto è la gestione delle scritture sui repositories. Il problema è riconducibile al Reader-Writer. È importante pensare a come sviluppare questa funzionalità tra il Server e più Clients che, eventualmente, tentano di aggiornare (`push`) contemporaneamente il repository.

È importante discutere e riportare le scelte fatte per implementare questo meccanismo.

Esempio sulla discussione per le gestione delle `push`.

> Si è considerato di implementare la `push` con una coda di richieste. Per definizione, dopo che un Client ha aggiornato il repository, tutti gli altri Clients avranno una versione vecchia. Prima di procedere col proprio aggiornamento dovranno aggiornare la propria versione locale (`pull`) e solo in seguito mandare il proprio aggiornamento (`push`) al Server.
>
> Invece di una coda, si è preferito consentire la scrittura solo al primo Client che fa `push`, mentre gli altri vengono notificati della scrittura in corso e che dovranno aggiornare la propria versione prima di mandare un aggiornamento.

Importante: uno dei punti della valutazione riguarda il grado di parallelismo consentito dal progetto. E.g., usare `execution { sequential }` per prevenire scritture (e letture) parallele è una soluzione al problema considerato, ma rappresenta anche una notevole perdita in termini di parallelismo, incidendo negativamente sulla valutazione del progetto.

2.3 Files di Configurazione

Non è richiesto, a livello di consegna progettuale, l'implementazione del salvataggio e il caricamento di files di configurazione. Ad ogni modo avere dei files di configurazione può velocizzare sviluppo e testing dell'applicazione (e.g., basta immaginare di sviluppare il Client dopo il Server e di dover riconfigurare tutte le volte un nuovo repository per interagirci).

La standard library di Jolie mette a disposizione le operazioni `valueToXml` e `xmlToValue` di `XmlUtils` e `read` e `write` di `File`. Con poco sforzo è possibile implementare salvataggio di strutture dati Jolie in XML e viceversa la lettura da XML e il caricamento (i.e., conversione) dei files XML in strutture dati Jolie.

2.4 Report

Circa 8-10 pagine – singola colonna, font-size 12 – in formato PDF, dove vengono discusse:

- la struttura del progetto (la divisione delle funzionalità tra i servizi e la loro gerarchia);
- le scelte più importanti fatte sul progetto (e.g., qual'è la logica dietro alla funzione di `push`) e come sono state sviluppate le principali funzionalità (anche con snippets di codice);
- i problemi principali riscontrati, le alternative considerate e le soluzioni scelte;
- le istruzioni per eseguire il progetto (in generale) e quelle per eseguire una demo di esecuzione. La demo deve contenere almeno 2 Clients ed un Server. Il Server parte, uno dei due Clients (Client1) aggiunge il Server, crea un repository (Repo) e invia (`push`) il contenuto locale del repository (`file`, vuoto). Il secondo Client (Client2) aggiunge il Server, richiede i files contenuti in Repo (`pull`), aggiunge un nuovo file (`new_file`, vuoto) e aggiorna il Server (`push`).

All'indirizzo http://cs.unibo.it/~sgiallor/teaching/report_template.md è disponibile un canovaccio, in markdown, con la struttura del report. (Pandoc e altri tools permettono di creare PDF da files markdown). È possibile scrivere il report nel formato sorgente preferito (.doc, .tex), l'importante è che il PDF generato rispetti la struttura del succitato template.

Il report di progetto è molto importante e buoni progetti con report scadenti possono risultare insufficienti. Il report non deve essere la ripetizione delle specifiche, deve contenere le idee, le discussioni e le scelte progettuali, soprattutto per quelli considerati punti chiave.

Mantenere una documentazione completa e in linea con lo sviluppo è di estrema importanza per costituire uno storico delle scelte fatte, utile anche come knowledge base per nuovi progetti (e.g., riuso di soluzioni, oltre che di codice).

È buona norma scrivere un draft del report *durante* lo sviluppo del progetto, segnando di volta in volta le cose importanti di cui si discute. Impaginazione e rifiniture sono lavoro di pochi giorni prima della consegna.

2.5 Gruppi

I gruppi possono essere costituiti da un minimo di 3 a un massimo di 5 persone.

I gruppi che intendono svolgere questo progetto, devono comunicare via email a saverio.giallorenzo@gmail.com entro l'**11 Maggio 2015** la composizione del gruppo. L'email deve avere come oggetto "GRUPPO LSO" e contenere il **nome del gruppo** e una riga per ogni componente del gruppo, con "cognome, nome, matricola". Inoltre deve essere presente un **indirizzo email di riferimento** a cui mandare le notifiche al gruppo.

Inoltre, ogni componente di un gruppo deve iscriversi sul tool all'indirizzo http://esami.int.nws.cs.unibo.it/cgi-bin/Main.php inserendo l'appartenenza (il nome) al proprio gruppo di progetto.

L'appello è unico, indipendentemente dalla data di consegna del progetto e dalla data effettiva della discussione.

Chi non darà comunicazione della composizione del gruppo entro l'11 Maggio non potrà fare il progetto.

Nel caso, **chi non riuscisse a trovare un gruppo**, lo comunichi il prima possibile, entro e non oltre il **9 Maggio 2015**, a saverio.giallorenzo@gmail.com, con una mail con oggetto "CERCO GRUPPO LSO", specificando i propri dati (nome-cognome-matricola-email) ed eventuali preferenze legate a luogo e tempi di lavoro. Si cercherà di costituire gruppi di persone con luoghi e tempi di lavoro compatibili. Le persone senza un gruppo verranno raggruppate il prima possibile (e non sarà possibile modificare i gruppi formati).

2.6 Consegna e Date

Per la consegna si deve inviare una email con soggetto "CONSEGNA LSO", con contenuto il nome del gruppo a saverio.giallorenzo@gmail.com.

In allegato alla mail dovrà essere incluso uno archivio zip nominato `NOME_GRUPPO_LSO.zip`. L'archivio deve contenere il progetto, sotto la directory "project", e la relazione, nominata `REPORT_LSO.pdf`.

Il report va anche consegnato in forma cartacea nella casella del prof. Sangiorgi (piano terra del dipartimento di informatica, a fianco dell'ufficio).

È possibile inserire un README in "project" che riporti come lanciare gli eseguibili del progetto.

Ci sono due date di consegna:

- 23:59 di Mercoledì 1 Luglio 2015, con discussione entro lo stesso mese;
- 23:59 di Lunedì 21 Settembre, con discussione entro lo stesso mese o quello successivo;

In seguito alle consegne, verranno fissate data e ora della discussione del progetto, cercando di rispettare le tempistiche sopraccitate, compatibilmente coi tempi di correzione (FCFS). La data di discussione verrà notificata ai gruppi tramite la mail di riferimento.

2.6.1 Valutazione del progetto

Il progetto deve rispettare la consegna descritta nella Sezione 1 (sottosezioni comprese).

La Sezione Implementazione contiene consigli su una possibile implementazione del progetto, ma non sono requisiti formali che il progetto deve soddisfare. Se si trovano soluzioni alternative – e migliori – possono essere implementate liberamente.

Il progetto verrà valutato mediante una discussione di gruppo, con tutti i componenti del gruppo presenti.

Non è possibile per i componenti di un gruppo effettuare la discussione in incontri separati. Al termine della discussione, ad ogni singolo componente verrà assegnato un voto in base all'effettivo contributo dimostrato nel lavoro di progetto. La valutazione del progetto è indipendente dal numero di persone che compongono il gruppo.

2.6.2 Note Importanti

- non si accettano email o richieste di eccezioni sui progetti con motivazioni legate a esigenze di laurearsi o di non voler pagare le tasse per un altro anno.
- chi copia o fa copiare, anche un sola parte del progetto, si vedrà invalidare completamente il progetto senza possibilità di appello. Dovrà quindi rifare un nuovo progetto l'anno successivo.

2.7 Domande sul progetto e ricevimento

Per le domande sul progetto si consiglia di usare il newsgroup del corso `unibo.cs.scienzeinternet.so`. Risposte corrette alle domande dei colleghi sul newsgroup verranno valutate positivamente in sede di esame. È possibile chiedere informazioni o un ricevimento via email, specificando la motivazione della richiesta, a saverio.giallorenzo@gmail.com.