PARIWEB: MODULO PHP

RELATORE: Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Dario Visonà

Corso di laurea triennale in Ingegneria Informatica

A.A. 2009-2010

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

PARIWEB: MODULO PHP

RELATORE: Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Dario Visonà

A.A. 2009-2010

Ai miei genitori
che hanno compiuto
innumerevoli sacrifici
per permettermi di raggiungere
questo traguardo.

Indice

Sommario

Le reti peer-to-peer sono in continua espansione. Con il passare degli anni sono venute a crearsi nuove architetture che permettono a gruppi di computer connessi tra loro formando una rete distribuita, di sfruttare la potenza di calcolo e la condivisione dei file (o qualsiasi altro tipo di risorsa).

PariPari è un progetto ambizioso che si propone di realizzare una rete peer-to-peer sicura, robusta e completamente serverless che garantisca l'anonimato. Per il progetto il team di PariPari si affida al modello di sviluppo Extreme Programming[1] che ha permesso di lavorare al meglio e di gestire nel miglior modo possibile le risorse umane.

In questa tesi di laurea triennale è presentato il lavoro svolto dal gruppo Web nella gestione e nel miglioramento del plug-in. Il lavoro svolto durante l'elaborato consisteva, nell'integrazione del plug-in Web con la libreria DiESeL[2] in modo da creare un sistema distribuito di nodi secondo la filosofia per creare un Web Server Distribuito. L'altra parte del lavoro consiste nel fornire al plug-in un motore PHP che permetta l'elaborazione di file dinamici per soddisfare le richieste che arrivano per questo tipo di file dai client.

[1] [1]
[2] http://paripari.it/mediawiki/index.php/DiESeL_(distributore)

1. Introduzione

1.1. Reti peer-to-peer e PariPari

Una rete peer-to-peer (o P2P) è una qualsiasi rete di nodi senza client o server fissi, ma con un numero di nodi equivalenti (appunto "peer") che fungono sia da client sia da server verso altri nodi della rete.

Inizialmente le reti P2P, anche se basate sul principio di decentramento, non erano completamente autonome. Si basavano, infatti, su un server per mettere in contatto tra loro i vari nodi della rete, senza essere però utilizzati per il trasferimento file che avveniva tra i nodi. Ovviamente se il server non era raggiungibile, queste prime applicazioni non erano più funzionanti in quanto ai nodi mancava un mezzo per comunicare. La diffusione di queste prime applicazioni si è verificata inizialmente per lo scambio di file musicali (MP3). In seguito il miglioramento e l'aumento di utilizzo delle reti P2P hanno permesso lo scambio di ogni genere di file (es. video, programmi, immagini). Oggi questo tipo reti sono utilizzate in programmi di uso comune dalla quasi totalità di utenti di Internet (come ad esempio eMule, aMule o Azureus).

Per risolvere il problema della comunicazione tra nodi, sono stati sviluppati protocolli che prevedono di assegnare a ognuno degli elementi partecipanti alla rete l'indicizzazione di alcuni file, in modo tale che ognuno punti a un nodo attivo, rendendolo rintracciabile. Questo protocollo è noto come Kademlia.

PariPari è una rete serverless che si pone l'obiettivo di raccogliere i benefici delle reti p2p e di migliorarne gli aspetti che ne compromettono la sicurezza, come la mancanza di anonimato tra i nodi, infatti, sono piuttosto semplice ricavare gli indirizzi IP degli altri nodi.

L'aspetto più innovativo che questo progetto è la multifunzionalità, che mira a distribuire sulla rete i servizi più comuni, rendendoli disponibili anche a computer esterni alla rete di PariPari.

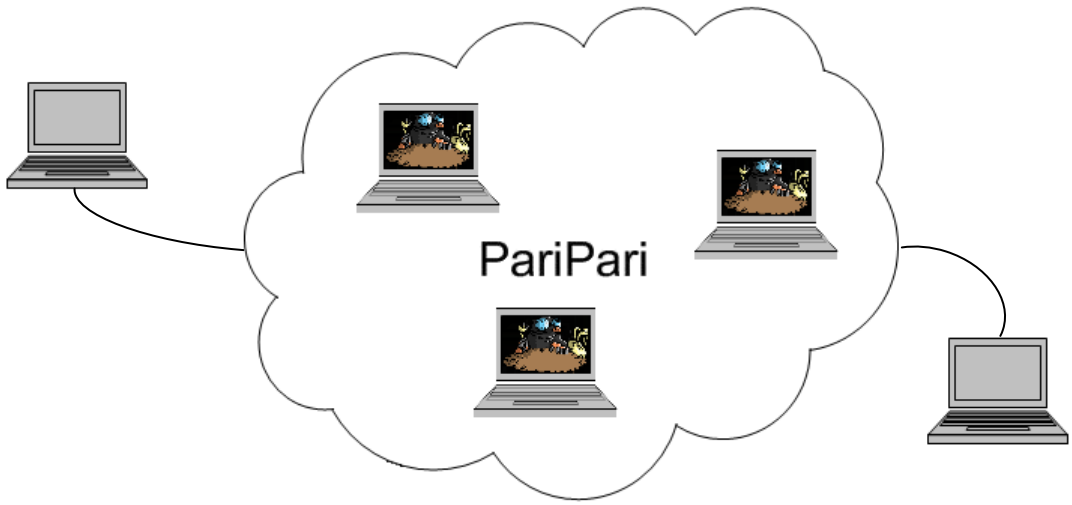

Figura 1: La rete PariPari e gli host esterni

PariPari, come già detto, cerca di fornire ogni genere di servizio comunemente usato e per raggiungere quest'obiettivo è utilizzata un'architettura a plug-in. Questi ultimi sono realizzati secondo una specifica interfaccia e interagiscono tra loro tramite il ***Core***. Per la gestione delle risorse sono utilizzati plug-in definiti "interni", tra questi ci sono DHT, Local Storage e Connectivity. L'utilizzo delle risorse è fornito da questi ultimi tramite le API.

A regolare lo scambio di "prestazioni" è stato implementato anche un sistema di crediti virtuali intelligente che tiene conto della richiesta e dell'utilizzo di risorse delle varie applicazioni, cercando di equilibrarle senza lasciare che qualche plug-in né "monopolizzi" l'utilizzo generando situazioni di starvation.

Il modulo crediti è implementato all'interno del Core e gestisce l'utilizzo delle varie risorse, rappresentate dai plug-in interni, dello stesso nodo perché tutte le interazioni passano obbligatoriamente da qui.

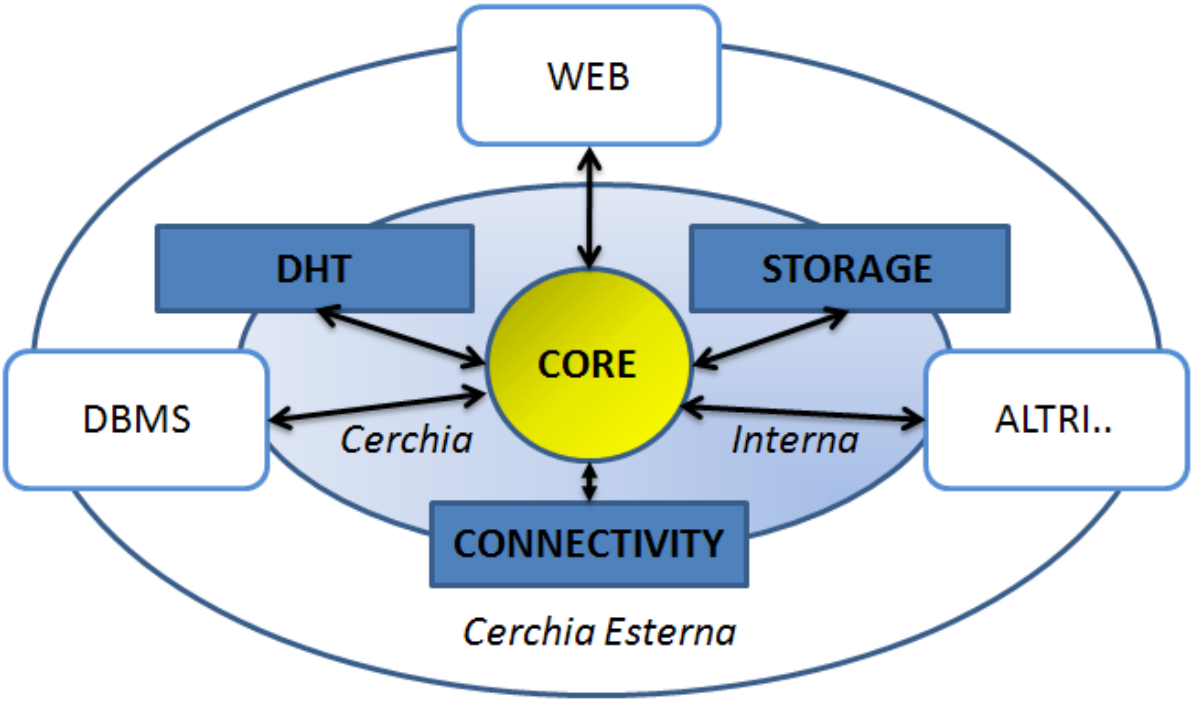

Figura 2: Architettura di PariPari

1.2. Extreme Programming in PariPari

PariPari è basato sul modello di sviluppo Extreme Programming (XP), il quale cerca di rendere lo sviluppo del software più agile. Essendo un tipo di metodologia agile, il processo è caratterizzato da rilasci frequenti, costituiti da piccoli cicli che mirano a migliorarne la produttività. Ci sono quattro linee guida principali da seguire per raggiungere quest'obiettivo:

1. Dare la maggior importanza possibile alla *comunicazione* tra sviluppatori e utilizzatori del software, inserendo gli utilizzatori all'interno del team.

2. *Feedback* frequenti e costanti da parte degli utilizzatori durante tutta la vita del software, per riuscire a governare i possibili e inevitabili cambiamenti.

3. *Semplicità* per mantenere design sistema e codice più pulito possibile, per favorire modifiche e manutenzione.

4. *Coraggio* nel modificare il sistema, per uso di pratiche di verifica di corretto funzionamento sistema anche dopo numerose modifiche.

Con queste linee guida si mira ad avere una pronta risposta ai cambiamenti rispetto all'esecuzione di un piano e lo sviluppo di un software che soddisfi il più possibile le necessità dell'utilizzatore che possono variare nel tempo.

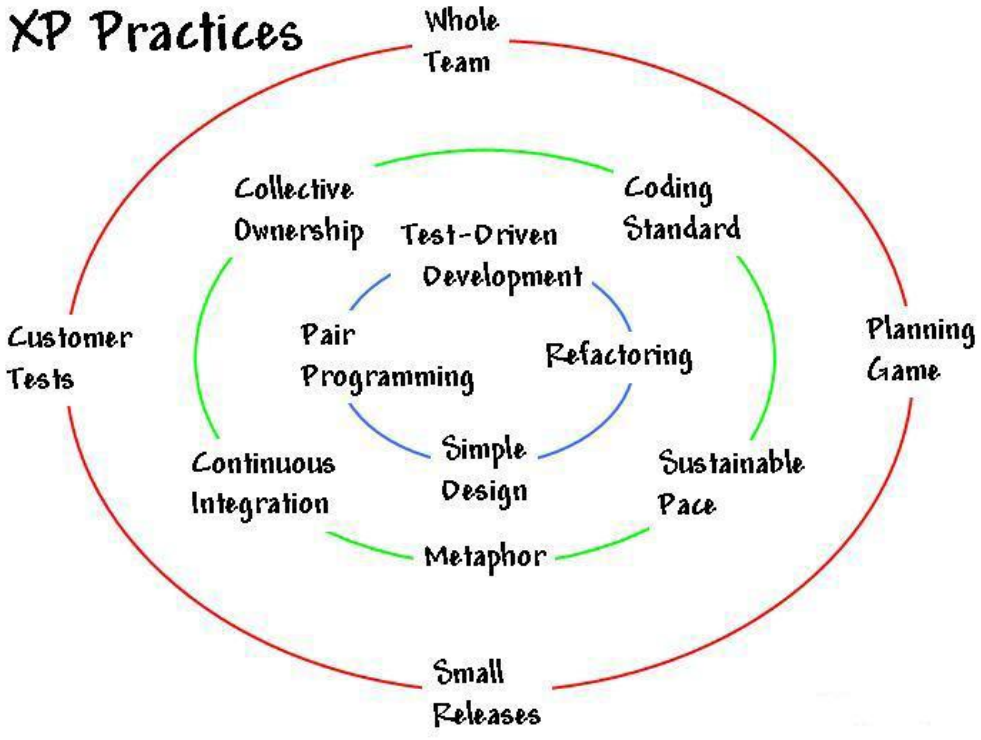

Figura 3: Extreme Programming, la metodologia di sviluppo utilizzata

Per quanto riguarda la comunicazione e il Feedback tra sviluppatori e utilizzatori nel team di PariPari, si può pensare come assoluta, infatti, le funzionalità che si cercano di sviluppare sono di uso comune e nel team gli stessi sviluppatori utilizzano spesso queste applicazioni e capiscono immediatamente i requisiti necessari per il software che si vuole implementare. La semplicità del codice va curata con un refactoring periodico, e con un corretto utilizzo degli standard di programmazione. Il design deve essere semplice, tale da permettere a un utilizzatore di capire senza difficoltà come va utilizzato il software. Sono poi effettuate piccole release incrementali, che a ogni ciclo migliorano le funzionalità del software. Per lo sviluppo s'iniziano a implementare le funzioni base, passando prima per un'attenta specifica e analisi dei requisiti del software. Durante l'implementazione si utilizza molto il testing, che aiuta a creare classi funzionanti, a misurare l'integrazione tra loro e a verificare il corretto funzionamento subito dopo le modifiche.

È utilizzata una programmazione a coppie (Pair Programming), che consente di sviluppare con la sicurezza del testing incrociato, in altre parole il secondo sviluppatore esegue Unit Test sulla programmazione del primo e viceversa. Uno dei punti critici in questo progetto è il variare continuo dei membri del team, infatti, trattandosi di studenti, a fine del corso di laurea lasciano l'università e quindi inevitabilmente anche il progetto. Per integrare al meglio i nuovi entrati si utilizzano strumenti come i gruppi internet in cui si possono porre domande e dubbi agli altri utenti iscritti (in questo caso tutti gli utenti sono i partecipanti al progetto PariPari).

PariPari inoltre è suddiviso in vari sottogruppi in cui ognuno ha il compito di gestire un plug-in. A ogni gruppo è assegnato un team leader che ha il dovere di sorvegliare l'avanzamento dei lavori e di prendere parte a riunioni con tutti i leader per discutere dell'avanzamento e della pianificazione dei lavori futuri. Anche il compito di motivare i membri del gruppo è compito del team leader, quest'ultimo non deve sopravvalutare la produttività di elementi non motivati, specialmente perché sono studenti e non sono retribuiti, con la pressione constante di esami che ruba loro molto tempo.

2. Plug-in Web Server

2.1. Protocollo http 1.1

Il protocollo HTTP (o Hyper Text Transfert Protocol) è un protocollo che sta alla base del World Wide Web, è utilizzato per la comunicazione tra client e server web. HTTP è generico e senza stato, e può rendere disponibile anche la gestione di oggetti distribuiti.

Queste specifiche sono definite nel RFC 2616 attraverso la realizzazione di metodi (come HEAD, POST, GET) e la trasmissione di messaggi di errore. L'insieme di computer che implementano questo protocollo per comunicare (chiamati client e server) danno vita al World Wide Web.

Spieghiamo nello specifico, cosa permettono i metodi esistenti:

- GET: permette il recupero di risorse da un server remoto che verrà specificata nella comunicazione
- HEAD: strutturata come il GET ma richiede solamente l'header e non la risorsa. Utile per la diagnostica.
- POST: serve per inviare al server dei dati che devo essere processati dalla risorsa specificata (es. un form html). Può anche servire per creare una risorsa o al suo aggiornamento.
- PUT: serve a caricare nel server una risorsa da client specificata nella comunicazione.
- DELETE: serve a eliminare una risorsa specificata all'interno della comunicazione.
- OPTIONS: serve a richiedere al server quale tra questi metodi sono disponibili per una risorsa, anche per richiedere le funzionalità del web server.
- TRACE: serve a vedere i cambiamenti che sono eseguiti nel formato della richiesta nel passaggio tra i server intermedi, infatti, restituisce la richiesta com'è arrivata.
- CONNECT: server a convertire la connessione richiesta in un tunnell tcp/ip trasparente, normalmente per facilitare connessioni SSL criptate (https) attraverso Proxy.

2.2. Request & Response

Le comunicazioni tra client e server sono chiamate request e response. Le richieste sono quelle che sono inviate dal client verso il server, mentre le risposte vanno nel verso contrario.

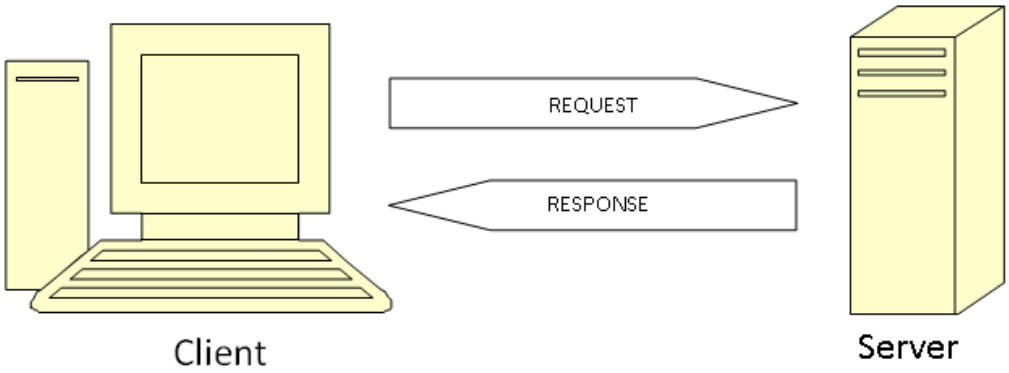

Figura 4: Schema Client/Server per il protocollo HTTP

La struttura della ***Request*** è la seguente:

Request-line Headers CRLF Message-body

La *Request-line* è composta dal metodo invocato più la *Request-URI*[3], la versione del protocollo usato e termina con il valore *CRLF*[4] tra loro sono separati dal carattere *SP* che indica il carattere di spaziatura:

Metodo SP Request-URI SP HTTP-Version CRLF

Gli *Headers* sono composti da tre campi, *general-header*, *request-header* ed *entity-header*. Permettono di inviare informazioni aggiuntive al server riguardanti la richiesta o il client stesso. Tra i più comuni ci sono:

- Host: nome del server cui si riferisce l'URI;
- User-Agent: identificazione del tipo di client;
- Accept: specifica i tipi di contenuti accettati. (es. MIME);
- Content-Type: specifica il tipo di MIME del contenuto della richiesta PUT o POST.

[3] Vedi standard RFC 2369
[4] CR *carriage return*, LF *line feed*

Il *Message-body* è un campo utilizzato per specificare il file da caricare nel server dal metodo PUT o un parametro nel caso del metodi POST.

I MIME (o Multipurpose Internet Mail Extension) sono degli indicatori di formato che inizialmente erano utilizzati solo nelle mail. In seguito sono stati contesi anche da altri protocolli come HTTP e SIP. Sono costituiti da varie parti: una specifica il tipo di versione utilizzata (*MIME-Version: 1.0*), una specifica un insieme di tipi e sottotipi contenuti (*Content-Type: image/gif*), un'altra ancora il tipo di codifica usata (*Content-Transfer-Encoding: Base64*).

La struttura della ***Response*** è molto simile:

Status-line Headers CRLF Message-body

La *Status-line* è composta dalla versione del protocollo utilizzato seguito da un codice numerico di tre cifre e una frase assegnata allo stato della risposta. Anche qui è strutturata con il carattere di spaziatura SP e conclusa con un carattere CRLF:

HTTP-Version SP Status-Code SP Reason-phrase CRLF

Qui di seguito è riportata la tabella con le categorie degli *Status-Code* usati:

Status-Code	Categoria	Esempi di motivazione
1xx	Informazione	Richiesta ricevuta, l'elaborazione sta continuando.
2xx	Successo	L'azione è stata ricevuta con successo, compresa e accettata.
3xx	Redirezione	Per portare a termine la richiesta sono richieste azioni ulteriori.
4xx	Errore del Client	Richiesta contente errori di sintassi o non soddisfabile.
5xx	Errore del Server	Server non in grado di soddisfare una richiesta apparentemente valida.

Tabella 1: Cinque categorie di codici di risposta

Tra i codici di stato più comuni ci sono:

200 OK

400 Bad Request

401 Unauthorized

404 Not Found

Tra gli *Headers* delle risposte più comuni ci sono:

- Server: indica il tipo e la versione del server;
- Content-Type: indica il tipo di contenuto restituito come MIME.

Nella figura 5 è indicato un esempio di possibile scambio di messaggi tra client e server:

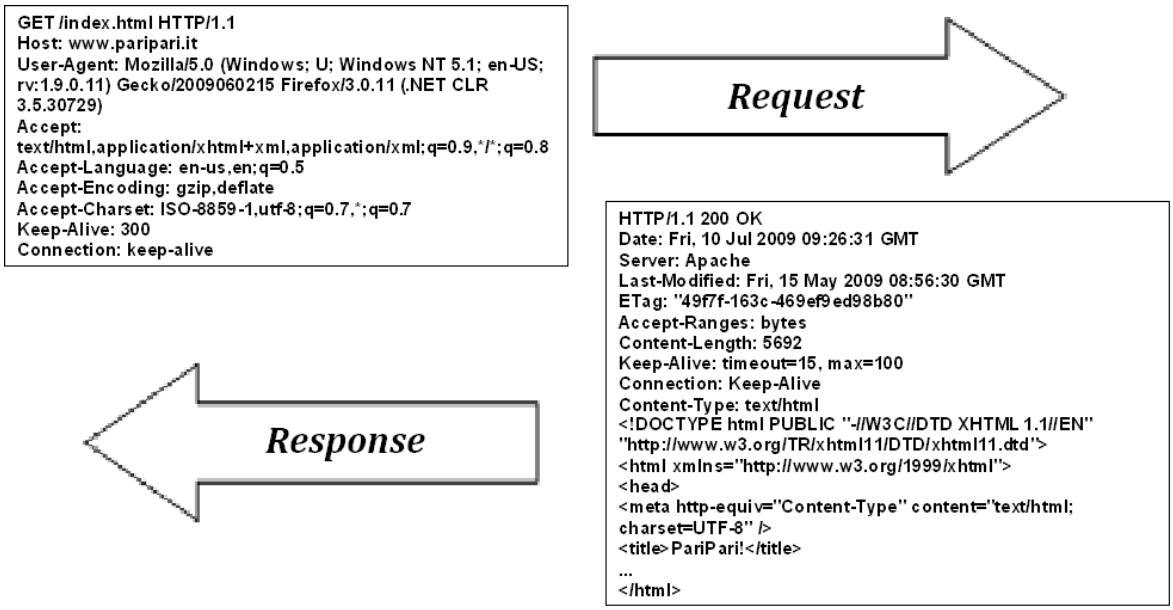

Figura 5: esempio di richiesta/risposta tra Client e Server.

2.3. *Funzionamento del web server*

Un web server è un programma che si occupa di accettare le richieste (*Request)* secondo il protocollo (RFC 2616), che arrivano dai client e di inviare le relative risposte (*Response).* Può avere però anche un secondo significato, ovvero la macchina su cui è fatto girare il precedente programma.

Il plug-in s'impone l'obiettivo di implementare questa funzionalità, e di migliorare il servizio applicando un servizio serverless. Ovvero creare un web server distribuito nella rete che gestisca al meglio la banda disponibile, lo spazio per le risorse e la disponibilità del servizio senza interruzioni.

Webserver è diviso in due modalità:

- Modalità in locale (si appoggia a una sola macchina);
- Modalità distribuita (verrà approfondita in seguito).

Quando si avvia il plug-in la prima volta, è necessario regolare alcuni parametri dal file di configurazione che consentirà di sceglierne le proprietà. Il file di configurazione si trova seguendo il percorso *PariPari/Webserver/conf/WebServer_properties.conf*. Aprendo il file con un normale editor di testo ci troviamo davanti alla seguente schermata:

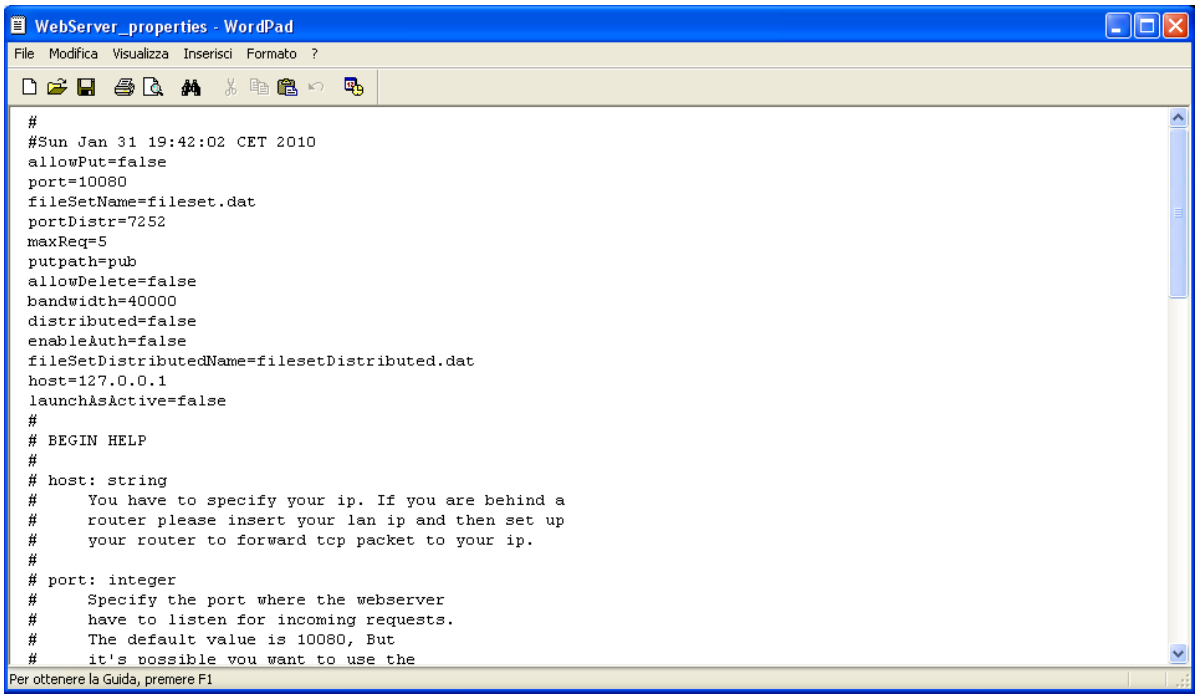

Figura 6: File di configurazione del web server.

I vari parametri sono così settati di default. La voce *Distributed* è un valore *boolean* che indica quale modalità tra le due precedenti, si vuole utilizzare (*true* per la distribuita, *false* per l'altra). I

parametri *launchAsActive*, *PortDistr* e *fileSetDistributedName* sono considerati solo per la modalità distribuita. Gli altri parametri valgono per il funzionamento di entrambe i modi.

Lanciando dalla console di PariPari il plug-in web con il comando *add webserver*, il Core eseguirà la classe principale *WebServer*.

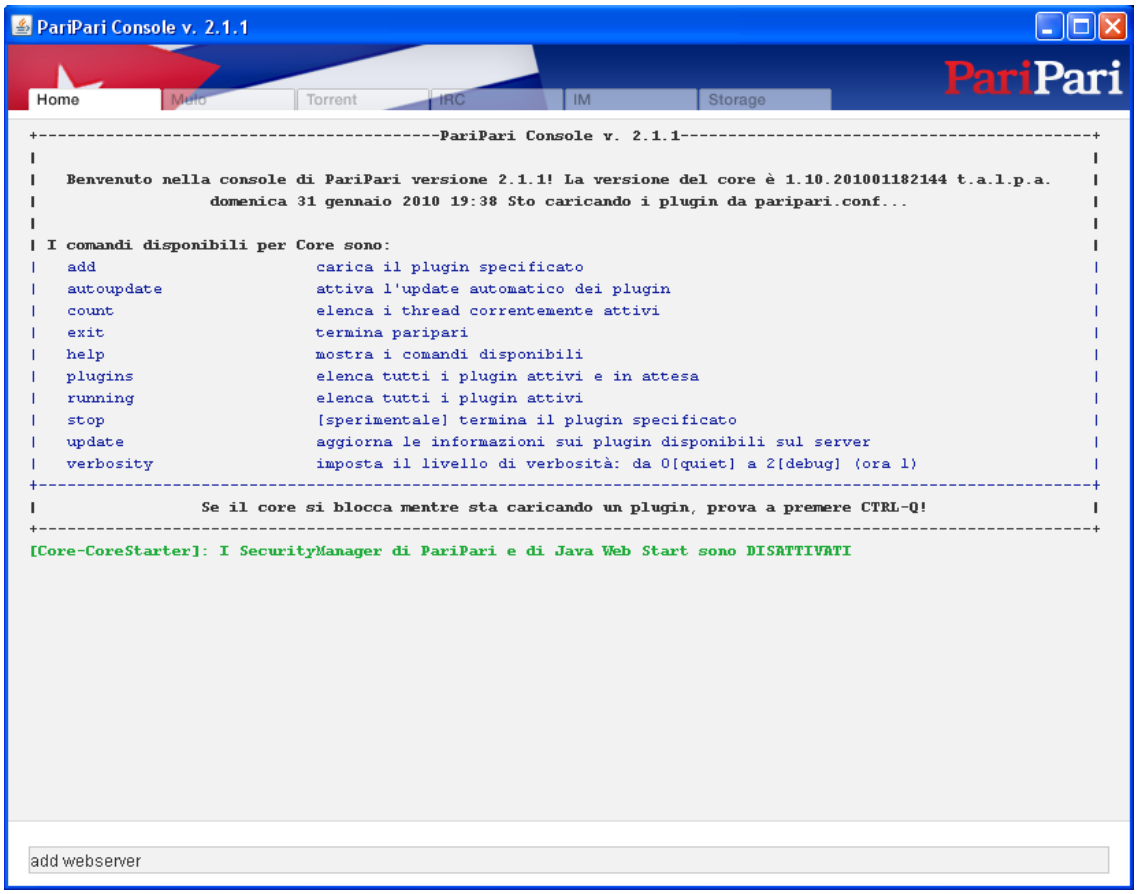

Figura 7: Console di PariPari

Questa classe ha il compito di recuperare le proprietà dal file di configurazione, con i parametri specificati in *host*, *port* e *bandwidth*, richiede un *LimitedServerSocketAPI* al Core tramite le API messe a disposizione dal plug-in interno Connectivity. Per richiederlo si usa la seguente implementazione:

```
SocketAddress ip_port = new InetSocketAddress(host, port);
LimitedServerSocketAPI socket = pluginSender.requestSingleServerSocketAPI(bandwidth, bandwidth);
bindSock(socket, ip_port);
```

PluginSender è una libreria molto utile implementata per rendere più semplice le richieste delle API al Core. La classe *WebServer* inoltre deve creare o recuperare le liste di domini e dei file relativamente contenuti. Come ultima cosa lancia le classi *WebConsole* e *ManageIncomingRequest* che si occuperanno di gestire le richieste in arrivo. *WebConsole* gestisce i comandi inseriti tramite la

console di PariPari, mentre l'altra si occupa della gestione delle richieste che arrivano al server dai client.

La classe *ManageFileSetNew* si occupa della gestione del FileSet[5] per il funzionamento in locale. Se c'è né uno già esistente sarà caricato, mentre per il primo avvio ne sarà creato uno nuovo. La classe *ManageList* si occupa invece della gestione dei domini con lo stesso principio, se esiste una lista già esistente sarà caricata altrimenti ne verrà creata una nuova. FileSet e la lista dei domini (domains.dat) vengono salvati nel disco (*PariPari/Webserver/data/domains.dat*). Per la gestione dei dati sul disco vengono utilizzate le API fornite dal plug-in interno LocalStorage. Per caricare un file da disco si utilizza la seguente implementazione, sempre servendosi della libreria PluginSender:

```
FileAPI file = pluginsender.requestSingleFileAPI(path+"/"+name, AccessMode.RW);
file.createNewFile();
FileInputStreamAPI fis = file.getQuotedFileInputStream();
ObjectInputStream ois = new ObjectInputStream(fis);
fileSet = (Vector<String>) ois.readObject();
ois.close();
```

Path e *name* indicano il percorso e il nome del file che si vuole caricare. Mentre per la creazione di un file si utilizza la seguente implementazione:

```
FileAPI file = pluginsender.requestSingleFileAPI(path+"/"+name, AccessMode.RW);
file.createNewFile();
FileOutputStreamAPI fos = file.getQuotedFileOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(fileSet);
oos.close();
```

Dal protocollo HTTP 1.1 il web server deve dare la possibilità di gestire più domini sulla stessa macchina (funzione di virtual host), infatti, l'*header host* è obbligatorio in questa versione. Permettendo quindi di capire direttamente dalla richiesta che arriva al server, il dominio a cui è richiesta la risorsa.

Nel plug-in web i domini, per comodità di gestione, sono le cartelle. Nel caso di un unico dominio non specificato, le risorse sono contenute in PariPari/web server/data/htdocs. Mentre nel caso di più domini, vanno singolarmente creati dal proprietario del web server attraverso la console (comando *add domain*). A ogni dominio creato, web server associa delle cartelle e vi salverà dentro le relative risorse.

[5] Elenco dei file salvati nel web server.

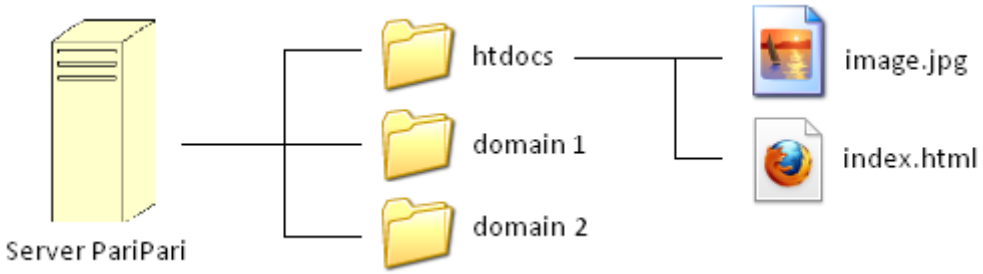

Figura 8: Organizzazione dei domini

Quando arrivano le richieste al server, la classe *ParserHTTP* crea una Hashtable che contiene tutte le informazioni utili alla generazione della risposta (ad esempio URI, Method, HTTP-Version). Dopo di che la classe *ServeMethod* in base al valore del campo Method estratto dalla richiesta si comporterà di conseguenza. Dal file di configurazione (parametro *maxReq*) si può scegliere il massimo numero di richieste contemporanee che il server deve accettare.

Infine la classe *TableResponse* si occuperà di generare la corretta risposta da inviare al client e la classe *HTTPResponse* invierà al client il flusso di byte attraverso il socket.

2.4. Comandi della console

In attesa della creazione della GUI di PariPari, la console è lo strumento con cui web server interagisce con il proprietario. Con questo strumento, infatti, si possono aggiungere domini, utenti, e altre funzioni per sfruttarne appieno le funzionalità.

Si può interagire con il plug-in digitando webserver dopo averlo già avviato. I comandi disponibili per la modalità locale sono:

- *help*: stampa tutti di comandi disponibili;
- *lsauth*: stampa gli utenti autorizzati;
- *lsprop*: stampa le proprietà impostate correntemente (dal file di configurazione);
- *lsfs*: stampa il nome dei file presenti nel webserver;
- *addfile* <*domain*> <*path*>: per aggiungere un file a un dominio, va specificato il path assoluto del file;
- *adduser* <*user*> <*password*>: per aggiungere un utente autorizzato;
- *rmuser* <*user*> <*password*>: per rimuovere un utente autorizzato;
- *lsdom*: stampa una lista con I domini esistenti;
- *adddom* <*domain*> <*folder*>: per aggiungere un dominio e il nome della cartella associata;
- *rmdom* <*domain*>: per rimuovere un dominio.

Per la modalità distribuita si è scelto di ridurre i comandi, e di inserirli a poco a poco in base alle necessità e alle scelte di implementazione future. Il comando *ipactive* permette di ricavare l'IP e la porta del nodo che ha generato il web server distribuito.

2.5. *Web server distribuito*

In precedenza abbiamo visto com'è sviluppata la modalità locale. Purtroppo l'utilizzo del web server in un unico nodo ha molte limitazioni (es. banda, spazio, raggiungibilità). E' stata implementata la seconda funzionalità per risolvere questi problemi. La gestione di un server attraverso una rete di nodi, infatti, lo rende migliore, non avendo lo spazio definito dalla singola macchina, non avendo limitazioni alla banda poiché si potrà dividere tra i nodi il compito di rispondere e quindi non intasando la comunicazione su uno solo. Per ultimo, molto importante, la probabilità che una rete di nodi non sia raggiungibile da una rete è pari alla probabilità che ogni singolo nodo non sia raggiungibile. Significa che abbiamo una probabilità molto maggiore che il server sia disponibile rispetto alla modalità in locale. Come già detto, questo è uno dei principi chiave del progetto PariPari (serverless).

Per raggiungere quest'obiettivo il plug-in sfrutta le funzionalità della libreria DiESeL. Inizialmente ideato per consentire la creazione di un server distribuito per IRC, con opportune modifiche è stato reso possibile l'integrazione anche con il web server.

Più approfonditamente DiESeL (o distributore) crea una serie di nodi, uno di questi in modalità attiva, cioè ha il compito di inviare dei ping verso tutti i nodi in modalità di ascolto (o passiva). Questi nodi una volta contattati entrano a far parte della nostra rete. Ovviamente ogni nodo, nel nostro caso, rappresenta un plug-in web che lancia un'istanza di Distributore. Questa libreria utilizza due porte, una per inviare i ping e nell'altra rimane in ascolto. Le porte sono specificate tramite il file di configurazione con la voce *portDistr*. Va specificata una sola porta, la seconda è presa per default di un numero successivo per comodità. Il funzionamento era prima implementato per considerare in tutta Internet, una sola rete esistente. Ora sono state apportate modifiche dal team di DiESeL per rendere possibile la creazione di più reti e quindi avere più server distribuiti coesistenti.

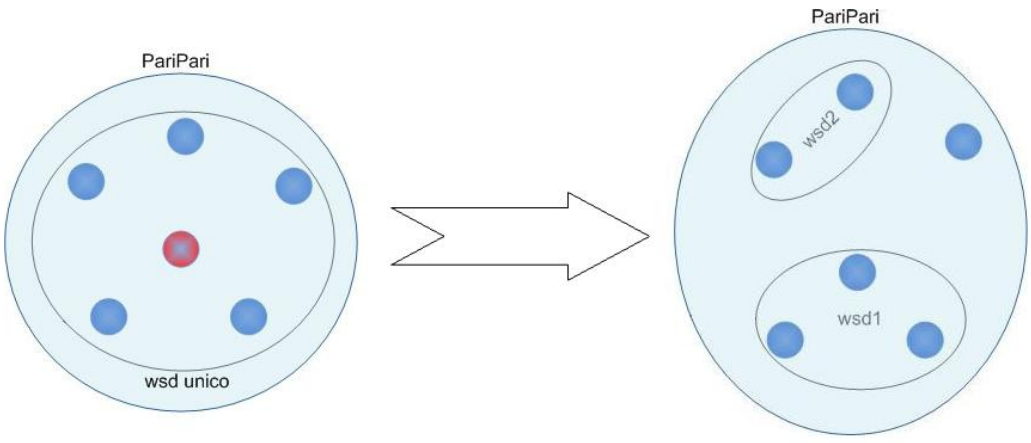

Figura 9: Funzionamento del web server prima e dopo le modifiche.

I problemi attuali sono la mancanza del DNS[6] di PariPari, che ci limita all'utilizzo del server dalla stessa macchina da cui viene lanciato, e il non ancora effettivo funzionamento di Distributed Storage che ci obbliga a salvare tutte le risorse del web server in ogni nodo locale.

[6] http://www.paripari.it/mediawiki/index.php/Dns_en

3. Modulo PHP

3.1. Introduzione al PHP

PHP sta per *Hypertext Preprocessor* ed è un modulo aggiuntivo per web servers che permette la creazione di pagina dinamiche. Il linguaggio PHP, infatti, è un linguaggio detto lato server (server side) perché è elaborato da quest'ultimo.

Una pagina php è composta da tag html e da parti di programmazione in php. Il codice php è contenuto dentro ai tag "<php" e "?>", e deve essere processato dal server prima di essere inviato al client. Il server elabora il file e lo trasforma in uno html che è leggibile dal normale browser che utilizziamo per navigare in internet.

Figura 10: elaborazione dei file php da parte del server

L'utilità di questo linguaggio è la sua dinamicità rispetto il codice html, permette, infatti, di gestire i siti in maniera efficiente e senza troppo lavoro. Il problema principale che possono avere i siti composti da molte pagine è dover fare dei piccoli cambiamenti grafici con corrispondente aggiornamento di tutti i file e una grande quantità di lavoro per il webmaster.

L'impiego più importante del php si ottiene nelle interrogazioni ai database presenti sul server. Questo linguaggio è con numerosi database, tra i quali:

- PostgreSQL;
- MySQL;
- Oracle;
- Adabas;
- filePro;
- ODBC.

Un altro vantaggio dell'utilizzo di questo linguaggio è che il file originale non è leggibile dal client, infatti, se si cerca di visualizzare il codice sorgente della pagina, non troviamo traccia del php,

poiché già elaborato e trasformato prima della richiesta. Può servire a un programmatore per tenere segreto il codice del suo sito.

3.2. Quercus

Quercus[7] è un'implementazione in java di PHP 5 rilasciato sotto licenza Open Source GPL dalla Caucho Technology[8]. Quercus è reso disponibile come servlet avviabile da qualunque application servers come Tomcat o GlassFish e non richiede nessun tipo d'installazione. Grazie all'implementazione in java, s'infondono i vantaggi di questo linguaggio di programmazione in PHP. Riuscire a infondere i vantaggi della JVM in PHP significa aumentarne prestazioni e sicurezza.

Ho scelto di utilizzare questo progetto come motore PHP per il plug-in. Non ci sono altre alternative valide se non Project Zero[9] della IBM, ma il tipo di licenza non è appropriato. Partire da zero con l'implementazione di un motore PHP richiederebbe troppo lavoro per riuscire a implementare tutte le parti del progetto, inoltre si rischierebbe di avere un prodotto poco affidabile.

Il lavoro svolto è stato quello di modificare il web server in maniera tale che potesse supportare la servlet e quindi permettere di elaborare file PHP.

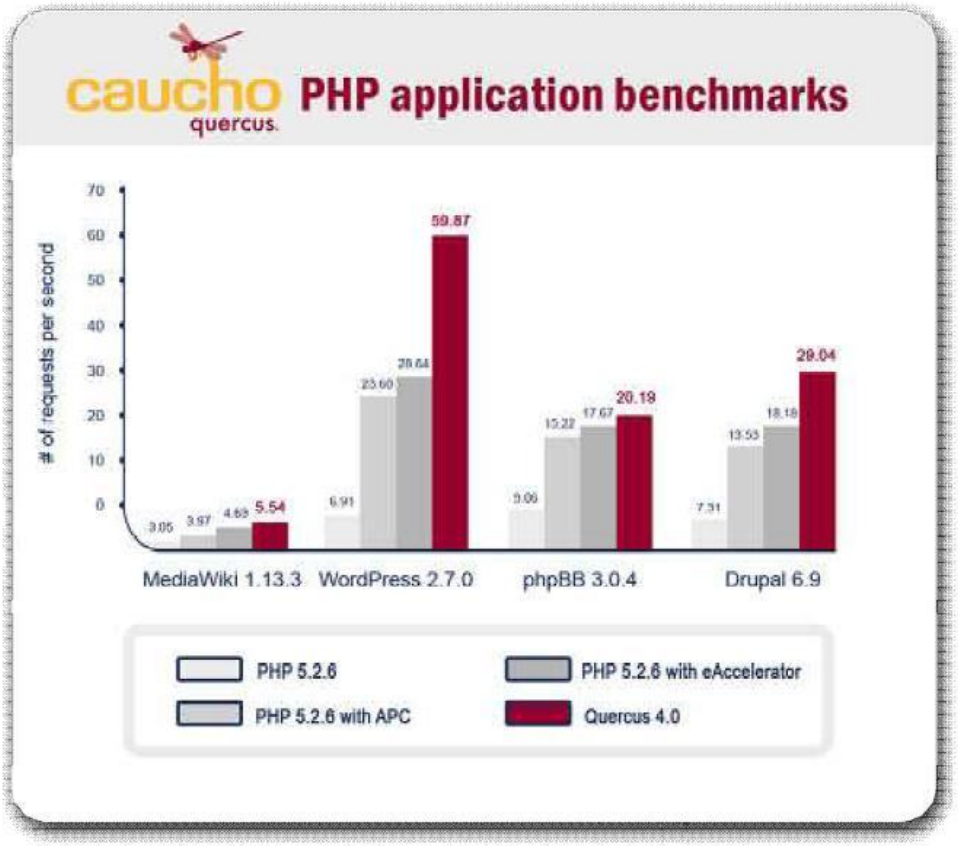

Figura 11: Istogramma delle prestazioni

[7] http://quercus.caucho.com/
[8] http://www.caucho.com/
[9] http://www.projectzero.org/

3.3. Gestione servlet

Dato il successo riscontrato dalle tecnologie *Java* e *Java applet*[10] presso la comunità di sviluppatori, gli ingegneri di Sun Microsystem hanno pensato di sviluppare una nuova tecnologia Java da applicare ai web server, così da poter estenderne semplicemente le funzionalità. Un oggetto simile alle applet ma installabile dal lato server: le servlet. Il ciclo di vita di una servlet prevede:

- invocazione del metodo init();
- passaggio delle richiesta alle servlet attraverso il metodo service(HttpServletRequest request, HttpServletResponse response);
- invocazione del metodo destroy().

La servlet, inoltre, contine un file web.xml che è detto descrittore di applicazione, all'interno vengono specificate le proprietà della servlet tramte dei tag.

Per essere eseguite devono essere gestite da un *servlet container*, che fa da tramite tra la servlet e il web server. Il funzionamento del servlet container si può descrivere in 4 fasi:

1) Intercetta la richiesta HTTP e crea gli oggetti request e response.

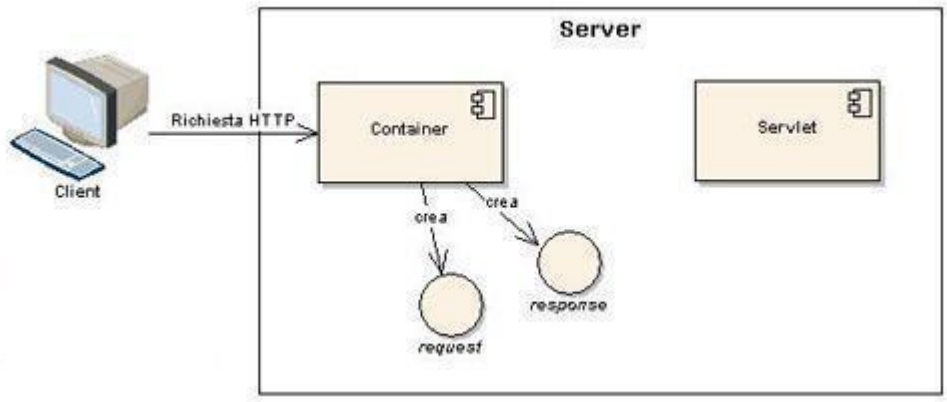

Figura 12: Gestione della richiesta da parte del servlet container

[10] Programmi in Java eseguibili da web browser.

2) Crea un thread della servlet per servire la richiesta e fornisce in input al metodo service gli oggetti request e response.

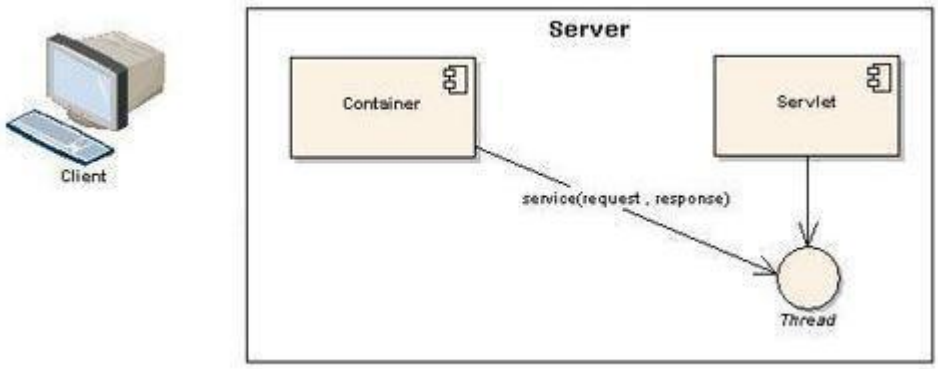

Figura 13: Invocazione del metodo service

3) Il metodo service del thread utilizza popola l'oggetto response che sarà poi utilizzato per la risposta al client.

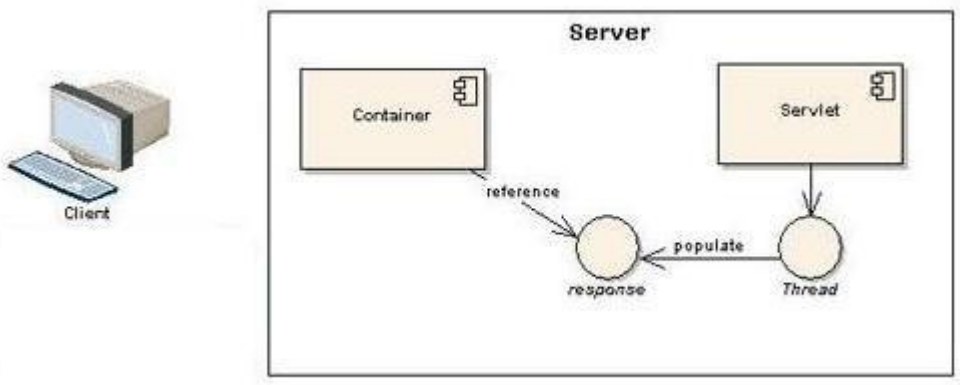

Figura *14: Il thread popola l'oggetto response*

4) Il container invia la risposta http ed elimina dalla memoria gli oggetti request e response precedentemente creati ed il thread che ha servito la risposta.

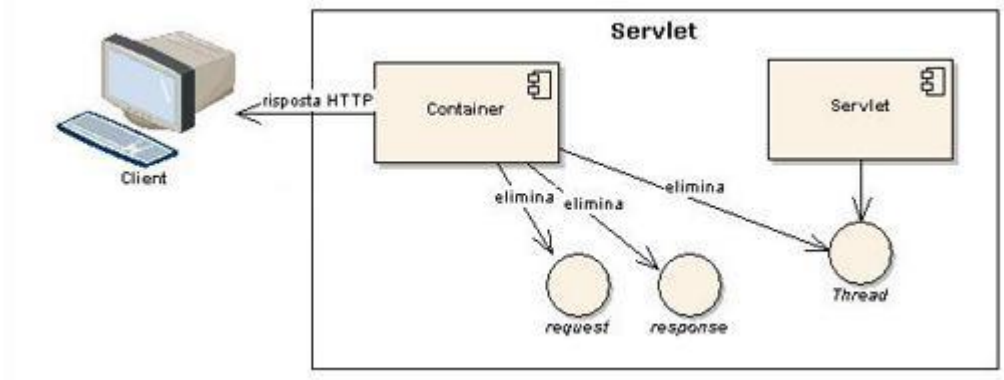

Figura 15: Creazione della risposta da inviare al client

4. Sviluppi futuri

Integrazione con distributed storage e web cache

Uno degli aspetti principali da risolvere è la ridondanza dei file che attualmente vengono salvati in tutti i nodi del web server distribuito tramite DiESeL. Questo può avvenire tramite il plug-in Distributed Storage[11], che mette a disposizione delle API che permettono di salvare (*store*) e recuperare (*retrive*) le risorse nella rete, che è formata da nodi pubblicati su DHT. Così facendo risolveremo il problema, e inoltre avremo a disposizione una quantità di spazio molto più elevata. Questa implementazione verrà aggiunta quando il plug-in distributed storage sarà perfettamente funzionante.

S'introduce, però un aspetto secondario non trascurabile relativo al tempo impiegato per il recupero dei file, infatti, questo tempo si somma al tempo che l'utente deve attendere prima di veder visualizzata la risorsa sul suo schermo. Se il tempo è troppo, viene avvertito dall'utente come un malfunzionamento del server. Per cercare di risolvere il problema si può creare in futuro una *web cache*. In base alle scelte di funzionamento della cache si può tenere in memoria ad esempio le risorse più richieste recentemente in modo tale da ridurre questi tempi di attesa.

Integrazione con Dns&Login

L'integrazione con DNS e Login permetteranno agli utenti di registrare il proprio dominio. Quindi vi sarà registrato l'indirizzo IP del nodo del server in modo tale da poter finalmente essere utilizzato. Questa integrazione permetterà inoltre di avere dei sottodomini.

Digest Access Authentication

Le specifiche del protocollo RFC 2616 prevedono che qualora un metodo invochi (PUT o DELETE) una risorsa contenuta nel server, necessiti di un'autenticazione dell'utente, il server deve rispondere con un messaggio di notifica nella status line: 401 unauthorized. Nel prossimo futuro questo metodo di autenticazione potrà essere migliorato e sostituito con *Digest Access Authentication* o con *Strong Access Authentication* (come il *public key authentication*). Inoltre bisogna suddividere in gruppi gli utenti per definire quali siano abilitati a modificare quali domini.

[11] http://www.paripari.it/mediawiki/index.php/Distributed_Storage_en

Conclusioni

Webserver è ancora lontano dal livello di concorrenza sul mercato, ma con queste funzionalità penso ci sia una buona base per lo sviluppo futuro, inoltre rendere il server distribuito aggiunge una grande potenzialità. Webserver inoltre ci da la possibilità di "hostare" già da ora il sito di PariPari senza ricorrere ad altri software. Una volta che il plug-in sarà pronto, dovrà passare la difficile prova dei test dell'utente che serviranno per scovare i problemi e i bugs che non sono stati rilevati durante la fase d'implementazione. Il lavoro svolto fin qui è servito per dare un'impronta di base al plug-in. I prossimi sviluppi perciò saranno agevolati e incentrati nell'ampliamento della gamma di servizi erogati.

PariPari è un progetto ambizioso ma dalle grandi potenzialità, grazie alla principale caratteristica della multifunzionalità. Le difficoltà che incontra sono dovute al realizzo del progetto, ma soprattutto dagli sviluppatori che ha a disposizione, essendo studenti universitari che devono dedicare molto tempo allo studio. Inoltre la maggioranza di questi entra nel progetto con poca esperienza nella programmazione e quando il loro livello si alza, spesso arriva il momento della laurea e abbandonano il progetto apportando solo un piccolo contributo.

Personalmente penso che lavorare in PariPari sia un'esperienza formativa, che aiuta a entrare nella mentalità del lavoro di squadra che sarà utile per lavori futuri. Il lavoro svolto è ripagato dalla soddisfazione nel vedere il progetto funzionante.

Appendice: le classi

Elenco qui di seguito le principali classi con relativa descrizione:

Webserver.java

È la classe principale e ha il compito di avviare tutte le funzionalità di base. Inoltre gestisce le richieste delle API al Core.

WebConsole.java

Si occupa dell'interazione tra l'utente e il plug-in. Gestisce la console in due modalità: locale o distribuita. Ovviamente i comandi differiscono, e saranno sicuramente aumentati in futuro in base alle scelte di sviluppo e realizzazione.

WebServerLayerNoDStorage.java

È stata creata tenendo conto della mancanza del plug-in Distributed Storage, infatti, tiene tutti gli oggetti in una struttura dati che viene mandata attraverso DiESeL a tutti i nodi del server distribuito. In futuri i file non saranno più mandati così, ma salvati nella rete con le API messe a disposizione dallo storage distribuito.

RequestPort.java

Questa classe richiede il *LimitedServerSocketAPI* a Connectivity e si occupa di eseguire il *bind* sulla porta richiesta.

DateRFC.java

Questa classe rielabora la data fornita dal *GregorianCalendar* di NTP[12] mettendola nel formato che viene specificato nell'RFC 1123.

[12] http://www.paripari.it/mediawiki/index.php/NTP

DistrObjNoDStorage.java

Questa classe tenendo conto della mancanza di Distributed Storage crea una struttura dati che viene utilizzata nella classe *WebServerLayerNoDStorage*. In futuro sarà utilizzata per inviare i file nella rete utlizzando una *hashtable* contente il nome del file e un array di byte.

HTTPResponse.java

Questa classe ha il compito di inviare la risposta http al client attraverso il *LimitedSocketAPI*.

InfoThread.java

Questa classe viene utilizzata all'interno di *Table Thread*. Contiene le informazioni relative ai thread che si occupano della gestione della richiesta. Tiene in memoria il nome, il puntatore al thread in esecuzione (nel nostro caso *ManageRequestThread*) e il momento in cui è arrivata la richiesta.

ManageFileSet.java

ServeMethod.java

Questa classe ha il compito di svolgere le funzioni richieste dal client. Qui infatti sono implementati i metodi definiti dal protocollo HTTP 1.1

TableResponse.java

Setta la risposta da restituire al client nel formato richiesto nel protocollo. Con il metodo *SetOK()* imposta a "200 OK" la status line, con il metodo *setBadRequest*() imposta la status line a "400 Bad Request", etc.

TableThread.java

Crea una tabella di di *InfoThread* per la gestione delle richieste in arrivo.

ManageList.java

La classe gestisce le diverse liste di dati (domini, user autorizzati) necessarie per le funzionalità del web server. Le strutture dati sono contenute in *hashtable*.

ManageFileSetNew.java

È una sottoclasse di *ManageList*, e gestisce il fileset, in azioni di aggiunta e rimozione delle risorse. All'avvio del plug-in inoltre legge i file presenti e sincronizza la lista.

ManageRequestThread.java

Riceve in ingresso una richiesta dal client tramite il *LimitedSocketAPI*. Questa classe è un thread, in modo da poter gestire più richieste in contemporanea.

Base64.java

Questa classe è stata introdotta per la gestione dei salvataggi dei dati e delle password. Non è sicuramente il modo più sicuro per tenere privati i dati, ma è sempre meglio che lasciarli visibili all'utente che apre il file.

ManageProperties.java

Gestisce il file di properties, nome e dove viene salvato. Inoltre stampa al suo interno una spiegazione dei parametri affinché siano più comprensibili.

ParserHTTP.java

Questa classe prende il flusso di dati che arrivano dall'*InputStream*. Svolge un parsing della richiesta HTTP. Una volta che ha riconosciuto la richiesta del client, compila una *hashtable* con i dati.

Utilities.java

Questa classe contiene alcuni metodi che possono essere utili allo sviluppatore.

Bibliografia

[1] Roger S. Pressman *Principi di ingegneria del software, quinta edizione,* McGraw-Hill (2008).

[2] Paolo Bertasi *Progettazione e realizzazione in Java di una rete peer to peer anonima e multifunzionale*, Università degli Studi di Padova (2006).

[3] *PariPari mediwiki* http://paripari.it/mediawiki/index.php/Main_Page.

[4] Paolo Malacarne *Java Servlet,* Apogeo (2004).

[5] *Wikipedia* http://en.wikipedia.org/wiki/Web_server.

[6] *Wikipedia* http://it.wikipedia.org/wiki/PHP.

[7] *Wikipedia* http://en.wikipedia.org/wiki/Java_Servlet.

[8] *Quercus* http://quercus.caucho.com.

[9] *RFC 2616* http://www.ietf.org/rfc/rfc2616.txt.

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

- I compagni di corso per l'aiuto a preparare gli esami e per i divertenti momenti di svago.
- I compagni del Team di PariPari.
- Tutti coloro che hanno contribuito al raggiungimento di questo risultato.

Grazie davvero