Project Heimdall

Proposta di implementazione per un web switch concorrente two-way di livello 7 (OSI) con politiche di bilanciamento del carico stateless e stateful

Alessio Moretti - 0187698
Andrea Cerra - 0167043
Claudio Pastorini - 0186256

Corso di Ingegneria di Internet e del Web - A.A. 2014/2015
Università di Tor Vergata
Ingegneria Informatica

Roma, 8 febbraio 2016

Indice

1 Example section

Yggdrasil, l'albero del mondo, che congiunge i nove regni del cosmo con Asgard, la dimora degli dei.

Heimdall, custode del Bifröst

Sample text and a reference[1]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec at lorem varius, sodales diam semper, congue dui. Integer porttitor felis eu tempor tempor. Proin molestie maximus augue in facilisis. Phasellus eros dui, blandit eu nibh ut, pharetra porta enim. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam ullamcorper risus pretium est elementum, eget egestas lorem fermentum. Etiam auctor nisi purus, vitae scelerisque augue vehicula sed. Ut eu laoreet ex. Mauris eu mi a tortor gravida cursus eget sit amet ligula.

Figura 1: Thor di Asgard, *figlio di Odino*

2 Introduzione

2.1 Perchè Heimdall?

Heimdall è il personaggio dell'universo Marvel, ispirato all'omonimo dio della mitologia norrena, egli è il guardiano del regno di Asgard e del Bifröst. Quest'ultimo è il ponte che unisce la Terra alla dimora degli dei ed Heimdall, come suo custode, ha il compito di aprirlo ed indirizzarlo verso gli altri mondo, permettendo solamente a chi è degno di attraversare le distese dello spazio.
Ci piace pensare che questo sia un po' il ruolo del software nato dal nostro progetto: che sia in grado di scegliere come meglio indirizzare le connessioni in arrivo, ponendosi come 'guardiano' di un cluster di server che fa ad esso capo. Quindi un **web switch** che sia funzionale sia per ricevere o trasmettere pacchetti di un regolare traffico HTTP che per bilanciare il carico dello stesso traffico in arrivo sulle varie macchine.

2.2 Web switch di livello 7

Nella terminologia delle reti informatiche uno **switch** è un commutatore a livello datalink, ovvero un dispositivo che si occupa di instradare opportunamente, attraverso le reti LAN, selezionando i frame ricevuti e reindirizzandoli verso la macchina appropriata a seconda di una propria tabella di inoltro. Un **web switch**, a livello applicativo, è capace di reindirizzare i dati in funzione dei pacchetti che riceve, analizzandone il contenuto e decidendo opportunamente la destinazione, occupandosi allo stesso tempo di reinoltrare anche l'eventuale risposta della macchina selezionata verso il client che l'ha generata.
Le applicazioni sono molteplici per l'implementazione a livello applicativo: può essere considerato un **proxy**, oppure, selezionando opportunamente la macchina con più velocità di risposta o con minore pressione, può agire come **bilanciatore di carico**. Infatti ognuno dei client che fa richiesta, ad esempio, per uno specifico sito web, invia un pacchetto ad un indirizzo IP pubblico che corrisponde a quello del nostro switch applicativo. Questi, dopo aver correttamente letto il pacchetto, si occupa di consultare una tabella

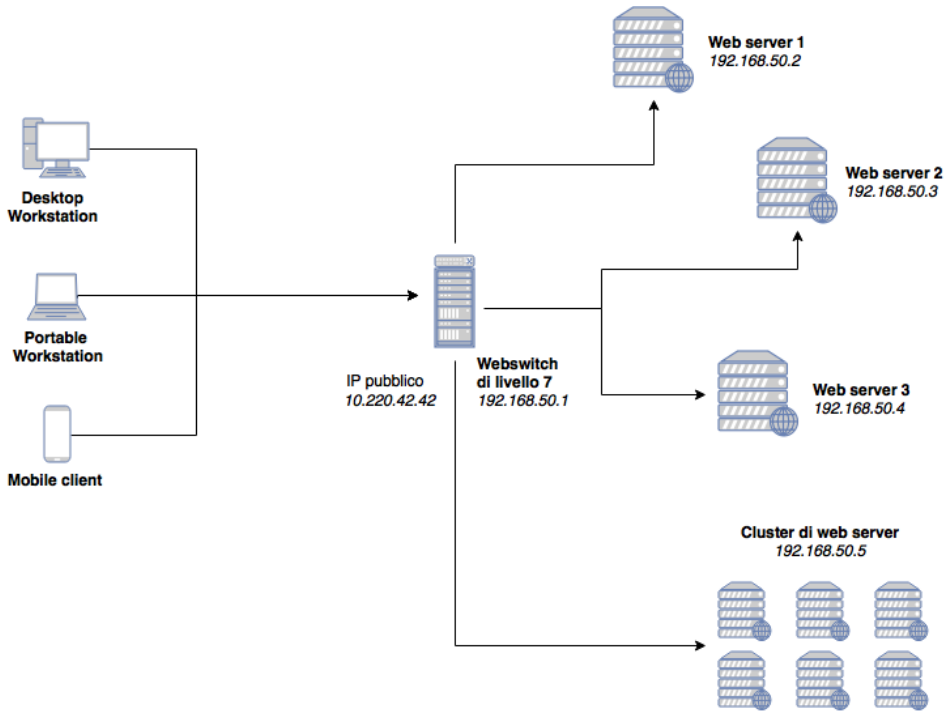

Figura 2: Esempio di uno *switch di livello 7 (OSI)*

di inoltro generata con una determinata **politica di scheduling** e quindi gestire l'inoltro della richiesta ed il reinoltro della risposta del webserver. tutto questo in maniera totalmente trasparente al client, qualsiasi sia la macchina che ha effettivamente risposto, che sia un web server oppure un cluster di macchine associate ad un ulteriore switch.

3 Architettura

3.1 Server in ascolto

3.1.1 File di configurazione

3.1.2 Logging

3.1.3 Gestione degli errori

3.2 Pool manager

3.3 Scheduler

3.4 Worker

3.4.1 Gestione delle richieste

Coda delle richieste

Chunk di dati

3.4.2 Gestione delle connessioni

Connessione

Richieste HTTP

Risposte HTTP

3.4.3 Thread di lettura

3.4.4 Thread di scrittura

3.4.5 Thread di richiesta

3.4.6 Thread di watchdog

4 Ulteriori proposte

5 Politiche di scheduling

La schedulazione permette la selezione della macchina predisposta a rispondere alla richiesta HTTP appena arrivata da parte del client, si basa su una tecnica nota come **bilanciamento del carico**, ovvero la distribuzione del carico, solitamente di elaborazione o di erogazione di uno specifico servizio, tra più server. Questo permette di poter **scalare** sulla potenza di calcolo del cluster dietro al web switch, lasciando che siano diverse macchine a rispondere a seconda di quella che è più veloce, più performante, oppure monitorando costantemente lo stato dei server e scegliendo quello meno sottoposto ad una pressione del carico di lavoro.

Nella nostra implementazione **thread scheduler** si occupa di fornire, ogni volta che viene invocato, una macchina selezionata secondo una delle due politiche che andremo ora a spiegare nel dettaglio più avanti.

5.1 State-less: implementazione con Round-Robin

L'algoritmo di scheduling Round-Robin (da adesso RR, *n.d.r.*) è un algoritmo che agisce con prelazione distribuendo in maniera equa il lavoro, secondo una metrica stabilita in partenza.
L'algoritmo funziona utilizzando un buffer circolare come possiamo vedere in *figura 3*: questo permette di iterare la selezione su una lista di elementi precedentemente caricata. E' necessario quindi specificare due passi per il corretto funzionamento, dopo aver dato un rapido sguardo alla struttura che lo rappresenta nella nostra implementazione.

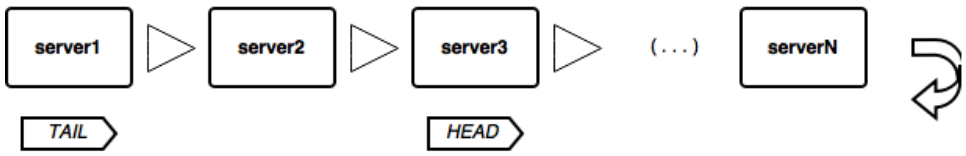

Figura 3: Schema di funzionamento del buffer circolare

```
typedef struct circular_buffer {
    Server      *buffer;
    int         buffer_position;
    int         buffer_len;

    Server      *head;
    Server      *tail;

    pthread_mutex_t mutex;

    ThrowablePtr (*allocate_buffer)(CircularPtr *circular, Server **servers, int len);
    ThrowablePtr (*acquire)(struct circular_buffer *circular);
    ThrowablePtr (*release)(struct circular_buffer *circular);
    void         (*progress)(struct circular_buffer *circular);
    void         (*destroy_buffer)(struct circular_buffer *circular);
} Circular, *CircularPtr;
```

Inizializzazione del buffer: in questa fase la struttura dati che rappresenta il buffer circolare, che mantiene due puntatori di *testa* e *coda*, viene inizializzata associandovi un array di puntatori di strutture di tipo *Server* e viene eseguita la seguente funzione:

```
/* inside allocate_buffer ... */
// allocating the buffer
circular->buffer = *servers;
circular->buffer_len = len;
// setting params
circular->head = circular->buffer;
circular->tail = circular->buffer + (len - 1);
```

In un'ottica di *produttore vs consumatore*

5.2 State-aware: implementazione con monitor di carico

5.2.1 Modulo ApacheStatus

6 Performance

6.1 Test di carico

6.2 Comparazione con Apache

7 Future implementazioni

7.1 Analisi della richiesta

7.2 Webserver performante

Annotazioni

[1] Leslie Lamport, *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.