Protokoll

Loadbalancer

Dezsys
5BHITT 2015/16

Niklas Hohenwarter, Selina Brinnich

Version 1.0

Note:

Betreuer: Thomas Micheler

Begonnen am 26. Februar 2016

Beendet am 03. März 2016

Inhaltsverzeichnis

1 Einführung

1.1 Aufgabenstellung

Es soll ein Load Balancer mit mindestens 2 unterschiedlichen Load-Balancing Methoden (jeweils 6 Punkte) implementiert werden. Eine Kombination von mehreren Methoden ist möglich. Die Berechnung bzw. das Service ist frei wählbar!

Folgende Load Balancing Methoden stehen zur Auswahl:

- Weighted Distribution
- Least Connection
- Response Time
- Server Probes

Um die Komplexität zu steigern, soll zusätzlich eine "Session Persistence" implementiert werden.

Vertiefend soll eine Open-Source Applikation aus folgender Liste ausgewählt und installiert werden.

https://www.inlab.de/articles/free-and-open-source-load-balancing-software-and-projects.html

1.2 Auslastung

Es sollen die einzelnen Server-Instanzen in folgenden Punkten belastet (Memory, CPU Cycles) werden können.

Bedenken Sie dabei, dass die einzelnen Load Balancing Methoden unterschiedlich auf diese Auslastung reagieren werden. Dokumentieren Sie dabei aufkommenden Probleme ausführlich.

1.3 Tests

Die Tests sollen so aufgebaut sein, dass in der Gruppe jedes Mitglied mehrere Server fahren und ein Gruppenmitglied mehrere Anfragen an den Load Balancer stellen. Für die Abnahme wird empfohlen, dass jeder Server eine Ausgabe mit entsprechenden Informationen ausgibt, damit die Verteilung der Anfragen demonstriert werden kann.

2 Ergebnisse

2.1 Loadbalancing Software

Als Software zum Loadbalancing bzw. für die Webserver wurde nginx gewählt. In diesem Beispiel werden 3 Server verwendet (1 LB, 2 Webserver). Zu Beginn muss nginx auf allen Servern installiert werden.

```
apt-get install nginx
```

Um danach beim Zugreifen den Unterschied zwischen den beiden Servern zu merken, wird ein einfaches *index.html* File im Webroot erstellt. Auf Server 1 enthält das File

```
Hallo, ich bin Server 1!
```

und auf Server 2

```
Hallo, ich bin Server 2!
```

Dazu werden diese Files in */usr/share/nginx/html* erstellt. Ruft man nun die IP des Servers auf, erhält man folgende Antwort:

Abbildung 1: Antwort Server1

Die gleiche Konfiguration wird nun auch für Server 2 vorgenommen. Danach muss der Loadbalancer konfiguriert werden [1]. Dazu wird die default site config geändert. Es muss folgendes geändert werden:

```
upstream backend  {
  server 10.0.106.210;
  server 10.0.106.220;
}
server {
  location / {
    proxy_pass  http://backend;
  }
}
```

Anschließend muss Nginx mittels `service nginx restart` neu gestartet werden. Danach funktioniert das Loadbalancing automatisch mit dem Algorithmus Round Robin. Durch das Aufrufen der IP Adresse des Loadbalancers und mehrmaliges Refreshen wechselt der Text immer zwischen „Hallo, ich bin Server 1!" und „Hallo, ich bin Server 2!" hin und her.

2.2 Loadbalancer

Als zu implementierende Algorithmen wurden Weighted Distribution und Response Time gewählt. Anfangs gab es Überlegungen Sockets zu verwenden, die Idee wurde jedoch später verworfen. Der Grund dafür war, dass die Aufgabe unnötig verkompliziert wurde, da jeder Socket in einem eigenen Thread laufen müsste um mit socket.accept() nicht das ganze Programm zu blockieren. Stattdessen wurde Java RMI verwendet.

Dazu wurde das Beispiel des PI-Calculators aus dem letzten Jahr als Ausgangspunkt verwendet. Am Loadbalancer mussten nur die Algorithmen dazuprogrammiert werden, da letztes Jahr nur Round Robin implementiert wurde. Der Loadbalancer gibt in der Konsole aus, an welchen Server er die Anfrage weitergeleitet hat und wie viele Stellen von Pi berechnet werden sollen. Beim Starten des Loadbalancers muss der gewünschte Algorithmus übergeben werden.

Weighted Distribution

```
/**
 * Implementiert einen Weighted Distribution Algorithmus
 * @return den Namen des zu verwendenden Servers
 */
public synchronized String balance_weighteddistrib(){
    if(!it_current_server_weighted.hasNext()) {
        it_current_server_weighted = server_weighted.iterator();
    }
    return it_current_server_weighted.next();
}
```

Abbildung 2: Weighted Distribution Algorithmus

Wenn der Loadbalancer den Algorithmus Weighted Distribution verwendet, dann wird jedes mal, wenn in der RMI Registry ein Server registriert wird, eine ArrayList erstellt/aktualisiert. In dieser Liste ist der Name des Servers so oft enthalten, wie gewichtet er ist. Dabei haben wir uns dazu entschieden, dass die Server statisch aufsteigend gewichtet sind (d.h. der erste Server hat die Gewichtung 1, der zweite Server die Gewichtung 2, usw.). In der im Bild zu sehenden Methode wird dann diese ArrayList durchiteriert. Zurückgegeben wird der Name des Servers (sozusagen der Primärschlüssel der RMI Registry).

Response Time

```
/**
 * Implementiert einen Response Time Algorithmus
 * @return den Namen des zu verwendenden Servers
 */
public synchronized String balance_responsetime(){
    long least = -1;
    String least_name = null;
    it_server_ips = server_ips.entrySet().iterator();
    while(it_server_ips.hasNext()) {
        Entry<String,String> next = it_server_ips.next();
        String ipAddress = next.getValue();
        try {
            InetAddress inet = InetAddress.getByName(ipAddress);

            long finish = 0;
            long start = System.nanoTime();

            if (inet.isReachable(5000)){
                finish = System.nanoTime();
                long time = finish-start;
                if(time < least || least == -1){
                    least = time;
                    least_name = next.getKey();
                }
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return least_name;
}
```

Abbildung 3: Response Time Algorithmus

Bei diesem Algorithmus wird die Response Time von jedem Server gemessen. Dazu wird die aktuelle Zeit in Nanosekunden gespeichert, geprüft ob der Server erreichbar ist und dann nochmal die aktuelle Zeit in Nanosekunden gespeichert. Die Differenz der beiden Zeiten ist dann ungefähr die Dauer der Anfrage. Der Server mit der geringsten Anfragedauer wird ausgewählt.

2.3 Server

Der Server wurde nicht verändert und vom PiCalculator Beispiel des letzen Jahres übernommen. Dieser Berechnet demnach Pi bis auf eine übergebene Anzahl an Stellen. Um den Server zu starten, muss die IP des Loadbalancers übergeben werden.

2.4 Client

Der Client wurde nicht verändert und vom PiCalculator Beispiel des letzen Jahres übernommen. Dieser schickt Anfragen an den Loadbalancer, mit dem Auftrag, Pi bis zu einer beim Start übergebenen Genauigkeit zu berechnen. Sobald die Berechnung abgeschlossen ist, wird das Ergebnis angezeigt

2.5 Tests

Getestet wurde mit folgendem Aufbau:

- Der Loadbalancer und die Server befanden sich auf einem Laptop in VMs die mittels VirtualBox virtualisiert wurden. Diese VMs haben alle Debian 8 installiert und sind ins Host WLAN gebridged. Die Java Version ist 1.8.0_74-b02
- Die Clients wurden auf einem anderen Laptop ausgeführt. Auf diesem Laptop is Xubuntu 14.04 installiert und Java hat die Versionsnummer 1.80_66-b17. Dieser Laptop kommunizierte über WLAN mit den VMs.

Weighted Distribution

```
tgm@loadbalancer:~$ java -jar loadbalancer.jar 2
Balancer successfully bound with IP 172.16.0.7
Balancer using algorithm Weighted Distribution
Registered server "srv1".
Registered server "srv2".
Server "srv1" started calculating pi to 50000 digits.
Server "srv2" started calculating pi to 25000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv1" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 25000 digits.
Server "srv1" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv1" started calculating pi to 10000 digits.
```

Abbildung 4: Loadbalancer mit Weighted Distribution

Wie auf dem Screenshot zu erkennen, bekommt Server 2 doppelt so viele Anfragen zugeteilt wie Server 1.

```
tgm@srv1:~$ java -jar server.jar 172.16.0.7 srv1
Server successfully registered with name "srv1"
Calculating pi to 50000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 50000 digits.
Calculating pi to 10000 digits.
```

Abbildung 5: Weighted Distribution Server 1

```
tgm@srv2:~$ java -jar server.jar 172.16.0.7 srv2
Server successfully registered with name "srv2"
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
```

Abbildung 6: Weighted Distribution Server 2

Response Time

```
tgm@loadbalancer:~$ java -jar loadbalancer.jar 1
Balancer successfully bound with IP 172.16.0.7
Balancer using algorithm Response Time
Registered server "srv1".
Registered server "srv2".
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv1" started calculating pi to 10000 digits.
Server "srv1" started calculating pi to 25000 digits.
Server "srv1" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 50000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 25000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 10000 digits.
Server "srv2" started calculating pi to 50000 digits.
```

Abbildung 7: Loadbalancer mit Response Time

Hier scheint die Verteilung der Anfragen zufällig. Zu erkennen ist, dass Server 2 scheinbar eine geringere Latenz hat als Server 1.

```
tgm@srv1:~$ java -jar server.jar 172.16.0.7 srv1
Server successfully registered with name "srv1"
Calculating pi to 10000 digits.
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
```

Abbildung 8: Response Time Server 1

```
tgm@srv2:~$ java -jar server.jar 172.16.0.7 srv2
Server successfully registered with name "srv2"
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 50000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 25000 digits.
Calculating pi to 10000 digits.
Calculating pi to 10000 digits.
Calculating pi to 50000 digits.
```

Abbildung 9: Response Time Server 2

3 Probleme

Problem: Access Denied Exception

Lösung: Um keine "Access Denied-Exception" zu bekommen, muss als erstes das java.properties File im Installationsordner des JRE unter jre/lib/security bearbeitet werden. Der Pfad muss dem Working-Directory des JAR-Files entsprechen. Das "-" zum Schluss bedeutet, dass die Unterordner rekursiv miteinbezogen werden:

```
grant codeBase "file:/home/selina/git/loadbalancer/jars/-" {
        permission java.security.AllPermission;
};
```

Abbildung 10: java.policy Änderung

4 Zeitaufzeichnung

Datum	Dauer	Name	Beschreibung
12.02.2016	2h	Hohenwarter	Protokoll erstellt Loadbalancing mit nginx Skizze Client Dokumentation Client
16.02.2016	2h	Brinnich	Java Client mit GUI & Sockets
20.02.2016	4h	Brinnich	Response Time, Deployment, Tests
26.02.2016	4h	Hohenwarter	Weighted Distribution, Dokumentation, Tests

5 Quellen

[1] Etel Sverdlov, How To Set Up Nginx Load Balancing, https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-load-balancing, zuletzt besucht: 26.02.16

6 Abbildungsverzeichnis

Anmerkung: Screenshots wurden invertiert, um Druckertinte zu sparen!