DezSys-Protokoll

DezSysLabor-10 "Load Balancer"

Dezentrale Systeme
5BHITT 2015/16

Erik Brändli, Thomas Stedronsky

Version 1.0

Note:

Betreuer: Th. Micheler

Begonnen am 27. März 2016

Beendet 04.März.2016

Inhaltsverzeichnis

1 Einführung

Die Übung soll die Grundlagen von Load Balancing Methoden vertiefen. .

1.1 Ziele

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.

1.2 Aufgabenstellung

Es soll ein Load Balancer mit mindestens 2 unterschiedlichen Load-Balancing Methoden (jeweils 6 Punkte) implementiert werden (ähnlich dem PI Beispiel [1]; Lösung zum Teil veraltet [2]). 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" (2 Punkte) implementiert werden.

Vertiefend soll eine Open-Source Applikation aus folgender Liste ausgewählt und installiert werden. (2 Punkte)
https://www.inlab.de/articles/free-and-open-source-load-balancing-software-and-projects.html

2 Ergebnisse

2.1 Design

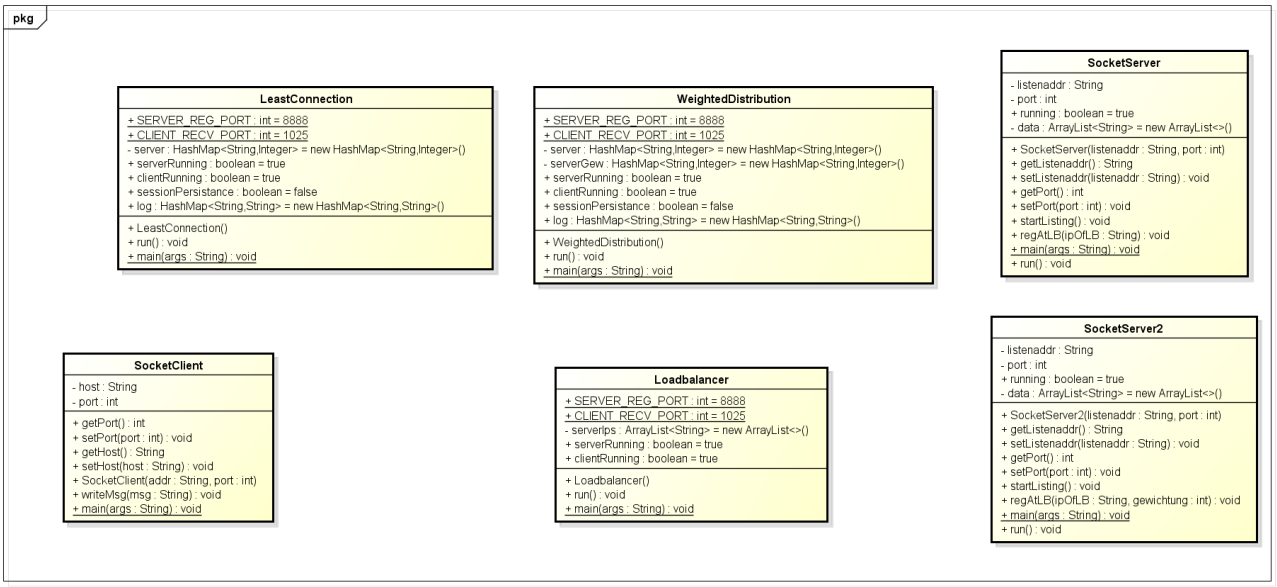

Abbildung 1 Design

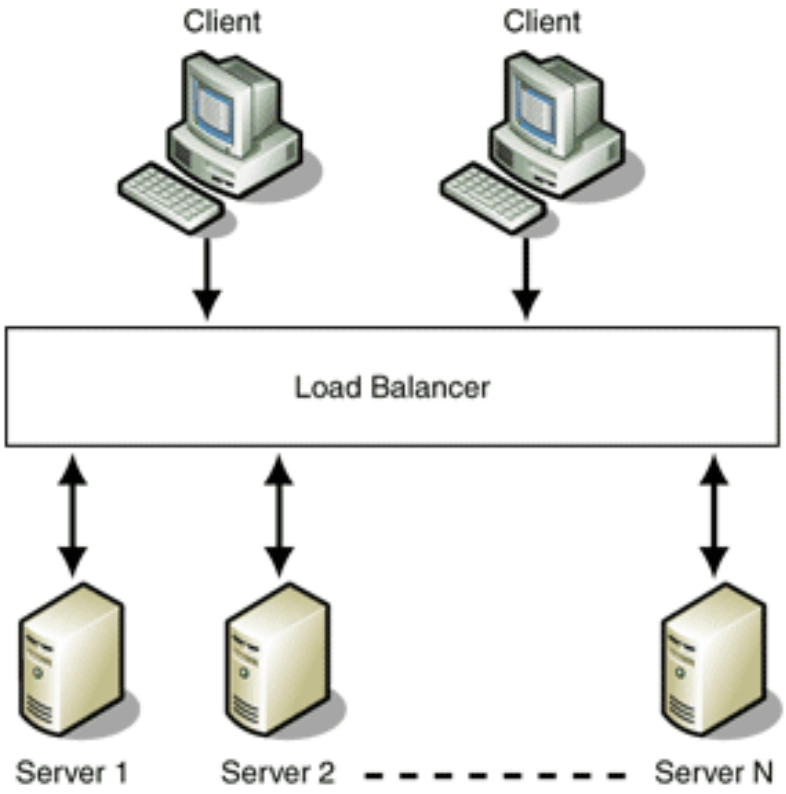

Abbildung 2 server archtitektur

2.2 Umsetzung

Die Verbindungen zwischen Server, Client und Load Balancer wurden mittels Java-Sockets implementiert.

Die Idee war es, dass sich die Server am Load Balancer mittels einer Socket Verbindung anmelden. Der Load Balancer speichert anschließend die IP-Adresse und den Port. Somit weiß der Load Balancer welche Server derzeit aktiv sind.

```
public void regAtLB(String ipOfLB){
    try {
        Socket s = new Socket(ipOfLB,8888);
        s.getOutputStream();
        PrintWriter pw = new PrintWriter(s.getOutputStream());
        pw.print(this.port);
        pw.close();
        s.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Code 1 Anmeldung am Load Balancer

Je nach Verteilungsmethode werden bestimmte Parameter wie aktive Verbindungen oder Gewichtung gespeichert.

Um als Client eine Nachricht an einen Server zu schicken wird lediglich die IP des Load Balancers benötigt. Es wird ein Socket zum Load Balancer mit der beliebigen Nachricht geöffnet und dieser leitet dann je nach Verteilungsmethode weitergeleitet.

```
public void writeMsg(String msg){
    Socket socket = null;
    try {
        socket = new Socket(host, port);
        PrintWriter pw = new PrintWriter(socket.getOutputStream());
        pw.print(msg);
        pw.close();
        socket.close();
    } catch (java.io.IOException e) {
        System.err.print(e);
    } finally {

    }
}
```

Code 2 Client Nachricht

2.3 Least Connection

Das Grundprinzip von der Load Balancing Methode Least Connection ist es, die dem Server mit den wenigsten aktiven Verbindungen die nächste Eingehende Verbindung weiterzuleiten. Der Load Balancer muss hierbei die aktiven Verbindungen immer aktuell halten. Diese Methode ist sehr effektiv und wird in der Praxis oft eingesetzt. [1]

In unserem Fall haben wir dies mittels einer HashMap am Load Balancer implementiert.

```
private HashMap<String, Integer> server = new HashMap<String, Integer>();
```

Hier wird der Key als String definiert, in diesem String werden die IP-Adresse und der Port definiert. Der Integer Wert (Value) entspricht hierbei den aktuellen Verbindungen.

```
String weiterIp="";
int weiterConnections=0;
int i =0;
for(String key : server.keySet())
{
    if(i==0){
        weiterConnections=server.get(key);
    }
    if(weiterConnections>=server.get(key)){
        weiterIp= key;
       weiterConnections=server.get(key);
    }
    ++i;
}
Socket socket = new Socket(weiterIp.split(":")[0],
Integer.parseInt(weiterIp.split(":")[1]));
server.put(weiterIp, server.get(weiterIp) + 1); //Update der Connection
PrintWriter pw = new PrintWriter(socket.getOutputStream());
pw.print(data);
```

Code 3 Least Connection Algorithmus

Bei diesem Algorithmus wird überprüft welcher Server derzeit die wenigsten Verbindungen hat. Beim erstmaligen Ausführen wird die derzeit kleinste Anzahl an Connections gesetzt und mit dieser werden anschließend die Verbindungen der anderen Server verglichen. Ist ein Server dabei dessen aktive Verbindungen kleiner sind wird die Weiterleitungs-IP auf die IP dieses Servers gesetzt und anschließen die kleinste Anzahl an Verbindungen wird die Variable `weiterConnections` aktualisiert.

Test

Bei diesem Test wurden 10 Client Anfragen an den Load Balancer geschickt. Der Load Balancer nimmt diese entgegen und tut sie anhand des oben beschriebenen Algorithmus verteilen.

Hier sieht man einen kleinen Ausschnitt des Logs vom Load Balancer, man kann erkennen, dass immer der Server mit den wenigsten Verbindungen gewählt wird, sollten 2 oder mehr Server dieselbe Anzahl an Verbindungen haben wird zufällig ausgewählt.

Ausgabe am Load Balancer:

```
IP: 192.168.0.19:1028 - Connections: 0

IP: 192.168.0.19:1026 - Connections: 0

IP: 192.168.0.19:1027 - Connections: 1


IP: 192.168.0.19:1028 - Connections: 0

IP: 192.168.0.19:1026 - Connections: 1

IP: 192.168.0.19:1027 - Connections: 1

…

IP: 192.168.0.19:1028 - Connections: 3

IP: 192.168.0.19:1026 - Connections: 3

IP: 192.168.0.19:1027 - Connections: 4
```

Beim Serverlog wird bei einer erfolgreichen Verbindung die übermittelte Nachricht ausgegeben, zur Übersichtlichkeit 0-9. Dadurch kann nachvollzogen werden wo welche Nachricht hingelangt.

Ausgabe am Server:

```
Port 1027   data: 0                    Port 1028   data: 5

Port 1026   data: 1                    Port 1027   data: 6

Port 1028   data: 2                    Port 1026   data: 7

Port 1027   data: 3                    Port 1028   data: 8

Port 1026   data: 4                    Port 1027   data: 9
```

2.4 Weighted Distribution

Mithilfe des Weighted Distribution Algorithmus kann einem Server eine bestimmte Gewichtung in der Server Architektur zugewiesen werden. So kann ein Server Beispielsweise eine Gewichtung 5 haben, der andere 3 und der Andere 2.

Es wird eine weitere HashMap benötigt um die Gewichtung für die einzelnen Server zu speichern.

```
private HashMap<String, Integer> serverGew = new HashMap<String, Integer>();
```

Außerdem werden wieder die derzeit aktiven Verbindungen zu dem jeweiligen Server gespeichert(wie schon bei Least Connection).

```
for(String key : serverGew.keySet())
{
    if(i==0){
        weiterIp=key;
    }
    if(hoechstens>=(server.get(key)/serverGew.get(key))){
        weiterIp= key;
        hoechstens=server.get(key)/serverGew.get(key);
    }
    ++i;
}
```

Code 4 weighted distribution algorhitmus

Es wird eine Gewichtungszahl berechnet, der Server mit der kleinsten Gewichtungszahl bekommt die Verbindung. Die Gewichtungszahl wird folgender Maßen berechnet:

Anzahl der aktiven Verbindungen auf dem Server / Die Gewichtung des Servers

Die Gewichtungszahl wird in `hoechstens` gespeichert und dann wird überprüft ob eine Zahl kleiner ist.

Test

Es werden 10 Anfragen vom Client an den Load Balancer geschickt dieser leitet dann mithilfe der Methode Weighted Distribution an den jeweiligen Server weiter. Bei identer Gewichtungszahl wird ein zufälliger Server ausgewählt.

Ausgabe am Load Balancer:

```
IP: 10.0.105.170:1028 - Gewichtung: 2
IP: 10.0.105.170:1026 - Gewichtung: 5
IP: 10.0.105.170:1027 - Gewichtung: 3

IP: 10.0.105.170:1028 - Connections: 0
IP: 10.0.105.170:1026 - Connections: 0
IP: 10.0.105.170:1027 - Connections: 1
…
IP: 10.0.105.170:1028 - Connections: 2
IP: 10.0.105.170:1026 - Connections: 5
IP: 10.0.105.170:1027 - Connections: 3
```

Beim Load Balancer sieht man anfangs die angemeldeten Server mit den jeweiligen Gewichtungen. Und anschließend die einzelnen Connections an den jeweiligen Servern.

```
Port 1027   data: 0
Port 1027   data: 1
Port 1027   data: 2
Port 1026   data: 3
Port 1026   data: 4
Port 1026   data: 5
Port 1026   data: 6
Port 1026   data: 7
Port 1028   data: 8
Port 1028   data: 9
```

Hier kann man nachvollziehen welche Nachricht an den einzelnen Server ankommen.

2.5 Session Persistenz

Die Session Persistenz ist sozusagen ein Log, das speichert zu welchen Server der jeweilige Client weitergeleitet wurde.

Dem Load Balancer wurde daraufhin eine weitere HashMap hinzugefügt wo die Client IP und die Server IP gespeichert wurden. Außerdem wurde eine boolean Variable hinzugefügt um die Session Persistenz zu steuern.

```
public boolean sessionPersistance = true;
public HashMap<String, String> log= new HashMap<String, String>();

if(sessionPersistance) {
    String clientIP = cs.getRemoteSocketAddress().toString().replace("/",
"").split(":")[0];
    if (log.containsKey(clientIP)) {
        Socket socket = new Socket(log.get(clientIP).split(":")[0],
Integer.parseInt(log.get(clientIP).split(":")[1]));
        PrintWriter pw = new PrintWriter(socket.getOutputStream());
        server.put(log.get(clientIP), server.get(log.get(clientIP)) + 1);
//Update der Connection
        pw.print(data);
        pw.close();
        socket.close();
    } else {
        Socket socket = new Socket(weiterIp.split(":")[0],
Integer.parseInt(weiterIp.split(":")[1]));
        PrintWriter pw = new PrintWriter(socket.getOutputStream());
        server.put(weiterIp, server.get(weiterIp) + 1); //Update der Connection
        log.put(clientIP, weiterIp);
        pw.print(data);
        pw.close();
        socket.close();
    }
    for(String key : log.keySet())
    {
        System.out.print("Client: " + key + " - ");
        System.out.print("Server: " + log.get(key) + "\n");
    }
    System.out.println();
}
else {
    Socket socket = new Socket(weiterIp.split(":")[0],
Integer.parseInt(weiterIp.split(":")[1]));
    PrintWriter pw = new PrintWriter(socket.getOutputStream());
    server.put(weiterIp, server.get(weiterIp) + 1); //Update der Connection
    pw.print(data);
    pw.close();
    socket.close();
}
```

Es wird überprüft ob der akzeptierte Socket-IP bereits im Log gespeichert wurde. Ist dies der Fall wird die Nachricht an den gleichen Server wie zuvor weitergeleitet. Ist diese IP nicht vorhanden wird die normale Verteilmethode durchgeführt, aber außerdem wird die Client IP plus Server IP in den Log geschrieben. Bei der nächsten Anfrage des Servers wird der Client dann an diesen Server weitergeleitet.

Test

```
IP: 192.168.0.101:1026 - Connections: 0
IP: 192.168.0.101:1027 - Connections: 0
IP: 192.168.0.101:1028 - Connections: 0

Client: 192.168.0.101 - Server: 192.168.0.101:1028

IP: 192.168.0.101:1026 - Connections: 0
IP: 192.168.0.101:1027 - Connections: 0
IP: 192.168.0.101:1028 - Connections: 1
…
Client: 192.168.0.101 - Server: 192.168.0.101:1028

IP: 192.168.0.101:1026 - Connections: 0
IP: 192.168.0.101:1027 - Connections: 0
IP: 192.168.0.101:1028 - Connections: 10
```

Im Load Balancer kann man dann erkennen das auf den einem Server 10 Connections liegen.

```
Port 1028   data: 0
Port 1028   data: 1
Port 1028   data: 2
Port 1028   data: 3
Port 1028   data: 4
Port 1028   data: 5
Port 1028   data: 6
Port 1028   data: 7
Port 1028   data: 8
Port 1028   data: 9
```

Anschließend kann man im Server Log erkennen (alle Anfragen vom selben Client), dass alle Anfragen an denselben Server weitergeleitet werden.

3 Load Balancing Software

Als externe Software wurde der Webserver nginx gewählt. Dieser Webserver erlaubt es einen Load Balancer mittels der Methode Round Robin einzurichten.

Es wurde auf 3 VMs nginx installiert mit apt-get install nginx (Load Balancer, Server 1, Server 2).

Auf den beiden Servern wurde im Verzeichnis *usr/share/nginx/html* ein index-html file erstellt. In diesem index.html befindet sich lediglich ein <h1> Tag mit dem Namen des Servers um ihn zu identifizieren.

Am Load Balancer musste die Methode Round Robin eingerichtet werden. Dazu musste in */etc/nginx/sites-available/default* folgendes geschrieben werden. [2]

```
upstream backend {
      server 192.168.0.24;
      server 192.168.0.23;
}
server {
      location / {
            proxy_pass http://backend;
      }
}
```

Code 5 nginx config

Anschließend wird der nginx Server am LB neu gestartet mit `service nginx restart`. Daraufhin wird im Browser die IP-Adresse des LB eingegeben. Der LB zeigt nur abwechselnd die index-Seite des Servers 1 und 2.

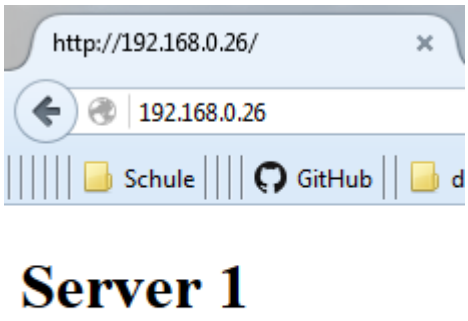

Abbildung 3 Server 1 LB

Abbildung 4 Server 2 LB

4 Codeverzeichnis

5 Abbildungsverzeichnis

6 Quellen

[1] Koelbl, Taschner, Load Balancing Ausarbeitung Kapitel 2.2, https://elearning.tgm.ac.at/mod/resource/view.php?id=46895, 2016

[2] Alex Fornuto, How to configure nginx, https://www.linode.com/docs/websites/nginx/how-to-configure-nginx, zuletzt besucht am 02.03.2016