

**DezSys-Protokoll**

**DezSysLabor-10 "Load Balancer"**

**Dezentrale Systeme**

**5BHITT 2015/16**

**Erik Brändli, Thomas Stedronsky**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 27. März 2016** |
| **Betreuer: Th. Micheler** | **Beendet 04.März.2016** |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc444626149)

[1.1 Ziele 3](#_Toc444626150)

[1.2 Aufgabenstellung 3](#_Toc444626151)

[2 Ergebnisse 4](#_Toc444626152)

[2.1 Design 4](#_Toc444626153)

[2.2 Umsetzung 4](#_Toc444626154)

[2.3 Least Connection 5](#_Toc444626155)

[2.4 Weighted Distribution 7](#_Toc444626156)

[2.5 Session Persistenz 7](#_Toc444626157)

[3 Load Balancing Software 8](#_Toc444626158)

[4 Codeverzeichnis 9](#_Toc444626159)

[5 Quellen 10](#_Toc444626160)

# Einführung

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

## 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.

## 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>

# Ergebnisse

## Design

## 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 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 Client Nachricht

## 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 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 1026 data: 1

Port 1028 data: 2

Port 1027 data: 3

Port 1026 data: 4

Port 1028 data: 5

Port 1027 data: 6

Port 1026 data: 7

Port 1028 data: 8

Port 1027 data: 9

## Weighted Distribution

## Session Persistenz

# Load Balancing Software

# Codeverzeichnis

[Code 1 Anmeldung am Load Balancer 4](#_Toc444622796)

[Code 2 Client Nachricht 4](#_Toc444622797)

[Code 3 Least Connection Algorithmus 5](#_Toc444622798)

# Quellen

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