



# DEZSYS10

# Load Balancing

**Systemtechnik**
**5BHITT 2015/16**

**Philipp Adler**
**Adin Karic**

**Version 1.0**

**Note:**

**Betreuer: Borko, Micheler**

**Begonnen am 19. Februar 2016**

**Beendet am 3.März 2016**

# Inhaltsverzeichnis

1 Einführung ..... 3
1.1 Ziele ..... 3
1.2 Voraussetzungen ..... 3
1.3 Aufgabenstellung ..... 3
2 Ergebnisse ..... 5
2.1 Designüberlegungen ..... 5
2.2 Umsetzung ..... 6
2.3 Testen der Umsetzung ..... 10
Testen von Least Connection ..... 10
Testen von Weighted Distribution ..... 12
2.4 Installation einer externen LoadBalancing-Software (Balance) ..... 14

# 1 Einführung

Diese Übung zeigt die Umsetzung von Load Balancing in Java

## 1.1 Ziele

Das Ziel dieser Übung ist es einen LoadBalancer in Java umzusetzen. Dabei sollen zwei verschiedene LoadBalancing-Methoden benutzt werden. In unserem Fall sind die Least Connections und Weighted Distribution. Durch die Installation einer externen LoadBalancing-Software (in unserem Fall "Balance") sollen wir uns noch ein wenig näher mit dem Thema Lastverteilung auseinandersetzen.

## 1.2 Voraussetzungen

- Grundlegendes Verständnis von Load Balancing
- Kenntnisse in Netzwerk-Kommunikation (Java)

## 1.3 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" (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

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

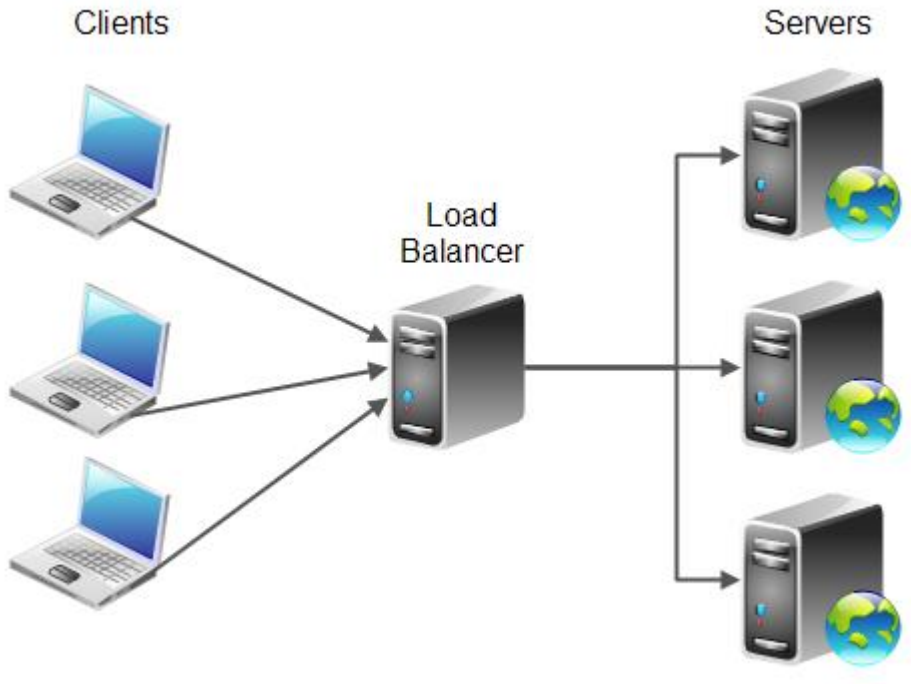
Bedenken Sie dabei, dass die einzelnen Load Balancing Methoden unterschiedlich auf diese Auslastung reagieren werden. Dokumentieren Sie dabei aufkommende Probleme ausführlich.

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.

Modalitäten

Gruppenarbeit: 2 Personen
Abgabe: Protokoll mit Designüberlegungen / Umsetzung / Testszenarien, Sourcecode (mit allen notwendigen Bibliotheken), Java-Doc, Build-Management-Tool (ant oder maven), Gepackt als ausführbares JAR



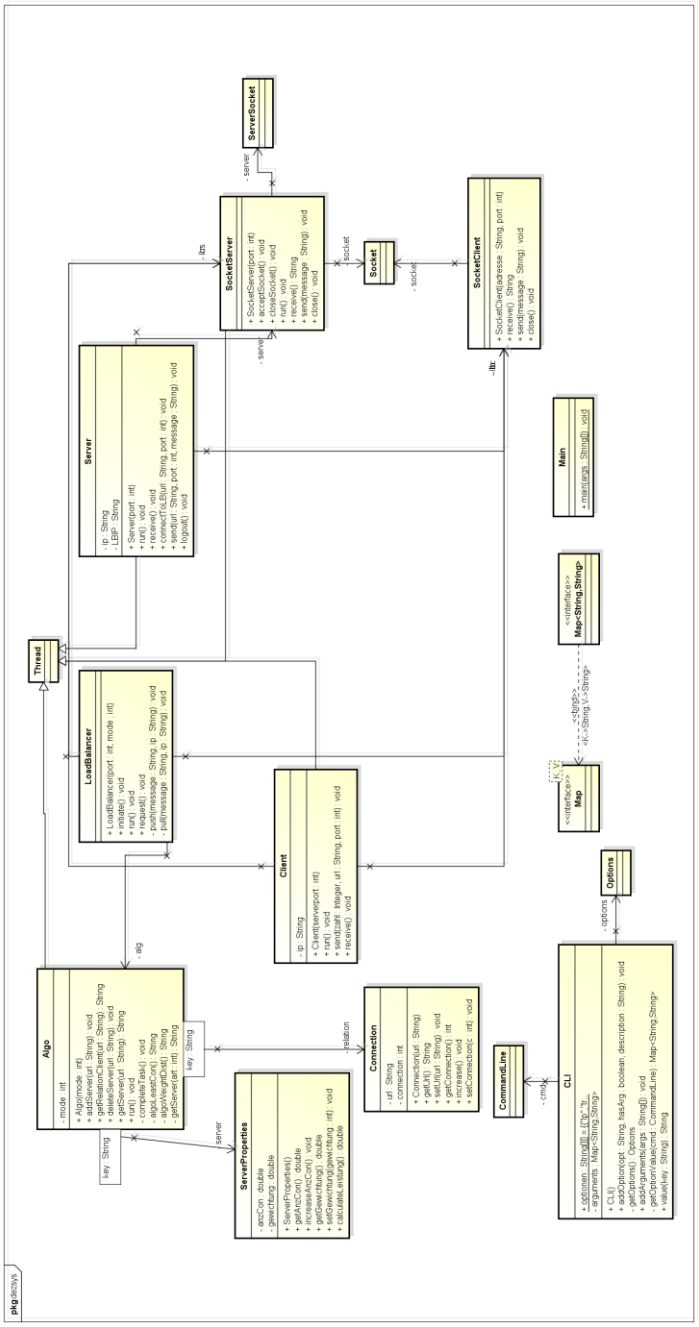
# 2 Ergebnisse

Alle Ergebnisse der Übung sind zudem in folgendem Github-Repo zu finden:

https://github.com/padler-tgm/LOAD_BALANCING

## 2.1 Designüberlegungen

Hier sieht man zunächst einmal das UML-Diagramm unserer Umsetzung:




(Bild sowie .astah datei sind im Repository zu finden)

Bei den Überlegungen zum Design haben wir uns zunächst den Pi-Calculator aus dem letzten Jahr angesehen und haben bemerkt dass dieser nicht optimal umgesetzt wurde bzw. viel zu viele Abhängigkeiten entstanden. Hier haben wir angesetzt und uns folgende Ziele bezüglich dem Design gestellt:

- unabhängiger LoadBalancer
- eigene Klasse mit Algorithmen zum Loadbalancen (erweiterbar!)
- Eingabe/Ausgbae getrennt vom Rest behandeln und auswerten(CLI)
- Server-Eigenschaften(Anzahl der Connections und Gewichtung) möglichst getrennt vom Server

## 2.2 Umsetzung

Wie schon im vorigen Design-Unterkapitel haben wir besonders auf eine saubere Trennung im SW-Design geachtet. Dabei wurden schließlich folgende zehn Klassen implementiert:

- Main.java
- CLI.java
- Client.java
- Server.java
- SocketClient.java
- SocketServer.java
- ServerProperties.java
- Connection.java
- LoadBalancer.java
- Algo.java

Beginnen wir bei der Main.java –Klasse. Diese Klasse dient als Startklasse. Beim Ausführen des Programms wird die main-Methode in dieser Klasse zuerst ausgeführt. Dabei werden zunächst Informationen über mögliche Parameter gegeben:

```
public static void main(String[] args) {
    //Hinweise zur Ausfuehrung
    System.out.println("Ich empfehle Ihnen nur eine Maschine zu starten:"+"\n"+
            "Entweder den LoadBalancer -lp PORT (-least ODER -weight)"+"\n"+
            "Oder Client -cp PORT -m ZAHL -u URL:PORT"+"\n"+
            "Oder -sp PORT -u URL:PORT");
```

Anschließend werden die eigegebenen Parameter mit einem CLI-Objekt verwaltet. Die Klasse CLI wird an dieser Stelle nicht noch näher beschrieben da wir schon sehr oft so eine Klasse genutzt haben und diese selbst aus dem Beispiel "Rückwärtssalto" aus dem 4.Jahrgang benutzen konnten (mit leichten Adaptierungen).

Als Nächstes werden die verschiedenen Teilnehmer der Umgebung geprüft: also Client, Server und Loadbalancer. Die Server die erzeugt werden melden sich dann beim LoadBalancer an. Die Clients senden "Anfragen" an den LoadBalancer der dann bestimmt welcher Server mit der Anfrage beauftragt wird. Bei uns ist der Service eine einfach Rechenoperation: der Client sendet eine Zahl und erhält das Doppelte dieser Zahl zurück.

Bei der Angabe des LoadBalancers gibt der User auch gleichzeitig die Art des LoadBalancings an: also entweder –least für die Methode Least Connections oder –weight für Weighted Distribution.

Das Grundgerüst ist also dass zunächst ein LoadBalancer erzeugt wird, dann ein Client dem es in einer Endlosschleife zusteht immer neue Anfragen an den LoadBalancer zu senden und schließlich die Server die sich beim Loadbalancer anmelden und auf Aufträge warten.

Der Loadbalancer erbt von der Klasse Thread (genauso wie Client und Server). Im Konstruktor des Loadbalancers werden wichtige Attribute gesetzt:

```
public LoadBalancer(int port, int mode){
    System.out.println("LoadBalancer laeuft auf PORT: "+port);
    this.lbs = new SocketServer(port);
    this.alg = new Algo(mode);
    this.initiate();
}
```

Die Kommunikation zwischen Client – Loadbalancer – Server funktioniert mit Sockets. Hierzu haben wir zwei Klassen implementiert: SocketClient und SocketServer. Die SocketClient – Klasse arbeitet mit einem Socket-Attribut, bei der SocketServer Klasse sind es logischerweise ein Socket-Attribut und ein ServerSocket-Attribut. In diesen beiden Klassen wird die Kommunikation umgesetzt. Die wichtigsten Methoden dabei sind receive(), send() und accept(),close(). Als Netzwerk-Identifikatoren werden wie üblich IP + Port benutzt.

Wenn der Loadbalancer nun eine Anfrage erhält, gibt es bei uns prinzipiell 4 Arten von "Anfragen" an den Loadbalancer:

Ein Server will sich beim LB registrieren; Ein Server will sich beim LB abmelden; Ein Client stellt eine Anfrage an den LB; Ein Server liefert die Antwort auf eine Anfrage

Auf diese 4 Möglichkeiten wird in einer Endlosschleife im Loadbalancer ständig "gehört". Zusätzlich sind im Loadbalancer Methoden für das Senden von Nachrichten an Server und Client definiert.

Beim Bearbeiten einer solchen Anfrage eines Clients wird zunächst ermittelt an welchen Server die Anfrage denn gesendet werden soll. Hier haben wir uns für die beiden Arten Least Connections und Weighted Distribution entschieden. Der Loadbalancer speichert die verfügbaren Server dann in einem Algo-Objekt welches dann mit der angegebenen Art den optimalen Server zur Bearbeitung der Anfrage bestimmt.

```
private String algoLeastCon() {
    String ip = "";
    ServerProperties s = null;
    double minAnz=999;
    for(Entry<String, ServerProperties> entry : this.server.entrySet()) {
        String key = entry.getKey();
        ServerProperties value = entry.getValue();
        //wenn die derzeitige minAnzahl an Connections groesser als die des servers in liste
        if(minAnz > value.getAnzCon()){
            minAnz = value.getAnzCon();
            ip = key;
            s = value;
        }
    }
    //erhoehen des counters
    s.increaseAnzCon();
    this.server.put(ip, s);
    return ip;

}

private String algoWeightDist() {
    String ip ="";
    ServerProperties s = null;
    double leistung=999;
    for(Entry<String, ServerProperties> entry : this.server.entrySet()) {
        String key = entry.getKey();
        ServerProperties value = entry.getValue();
        //wenn leistungsnummer groesser als die kalkulierte des jeweiligen servers
        //je kleiner die zahl desto besser
        if(leistung > value.calculateLeistung()){
            leistung = value.calculateLeistung();
            ip = key;
            s = value;
        }
    }
    //erhoehen des counters
    s.increaseAnzCon();
    this.server.put(ip, s);
    return ip;
}
```

Bei der Methode Least Connections wird der Server mit den wenigsten aktiven Verbindungen vorgeschlagen. Bei der Methode Weighted Distribution hingegen wird die Anzahl der aktien Verbindungen durch eine Gewichtungskennzahl dividiert. Diese Gewichtungskennzahl wird zufällig beim Erstellen des Servers zwischen 1 und 10 bestimmt (je höher die Zahl desto leistungsfähiger der Server). Der Server mit der kleinsten Zahl die bei dieser Division herauskommt wird schließlich vorgeschlagen.

Um diese Methoden der Lastverteilung nutzen zu können benötigt man natürlich diese Kennzahlen der verschiedenen Server. Dies haben wir mit der Klasse ServerProperties realisiert. Hier der Konstruktor der Klasse:

```
public ServerProperties(){
    this.anzCon = 0;
    this.gewichtung =  (int) ((Math.random()*10)+1);;//VON 1-10
}
```

In dieser Klasse finden sich verschiedene getter- und setter-Methode für die beiden Attribute anzCon und gewichtung. Hier findet sich auch eine Methode zur Berechnung der derzeitigen "Leistung" für das Weighted Distribution Verfahren:

```
public double calculateLeistung() {
    //die anzahl der Verbindugen wird durch die gewichtung des Servers dividiert
    double a = anzCon/gewichtung;
    return a;
}
```

Wurde schließlich ein Server bestimmt wird mittels der push-Methode die Anfrage vom LB an den Server weitergeleitet:

```
this.push(request.split(",")[0], request.split(",")[1]);//SCHICKT ANFRAGE AN SERVER
```

Der Server der die ganze Zeit zuhört erhält und bearbeitet die Anfrage:

```
//server berechnet ergebnis und sendet an lb
System.out.println("SERVER SENDET ERGEBNIS AN LB: "+Integer.parseInt(request.split("\\?")[1])*2);
this.send(this.LBIP.split(":")[0], Integer.parseInt(this.LBIP.split(":")[1]), "Reply "+(Integer.parseInt(request.split("\\?")[1])*2)+" "+request.split("\\?")[0]+"?"+this.ip);
```

Danach sendet er das Ergebnis mittels send() an den LB zurück. Dieser schickt das Ergebnis dann weiter an den Client der angefragt hat:

```
System.out.println("EMPFAENGT ERGEBNIS VOM SERVER "+ip.split("\\?")[1]);
this.pull(request.split(" ")[1],this.alg.getRelationClient(ip));//SCHICKT ANTWORT AN CLIENT
```

Bei der Implementierung hatten wir vorwiegend Probleme mit der Kommunikation und mussten unser SW-Design einige Male umkrempeln.

## 2.3 Testen der Umsetzung

Getestet wurde die Implementierung im TGM Netz. Der LoadBalancer läuft auf Port 9999. Hier die Konsolenausgaben und daraus folgenden Ergebnisse des Testens:

### Testen von Least Connection

```
SERVER 192.168.0.12:3331 HAT SICH REGISTRIERT
SERVER 192.168.0.12:3332 HAT SICH REGISTRIERT
SERVER 192.168.0.12:3333 HAT SICH REGISTRIERT
```

**FIRST TIME**

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT localhost:2331 2
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2332 4
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2333 6
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2334 8
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2335 10
```

**SERVER 192.168.0.12:3331 2**
**SERVER 192.168.0.12:3332 2**
**SERVER 192.168.0.12:3333 1**

**SESSION**

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT localhost:2331 2
```

```
SESSION
```

```
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2332 4
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2333 6
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2334 8
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2335 10
```

**SERVER 192.168.0.12:3331 4**
**SERVER 192.168.0.12:3332 4**
**SERVER 192.168.0.12:3333 2**

**AFTER 20 SECONDS**

```
DELETE SESSION 7faac844-2a8a-4afc-b0e3-
4e8f6a5b0132?192.168.0.12:3331
DELETE SESSION a212f010-0821-4880-b821-
219bd35bec5b?192.168.0.12:3332
DELETE SESSION 9906d333-7d50-4edd-9c4c-
9d34db3d22cd?192.168.0.12:3333
DELETE SESSION 2e19c2d9-a401-40e0-8b31-
a536739dd8a8?192.168.0.12:3332
DELETE SESSION e4b9b82e-d325-41d5-aa24-
717aebdd46ff?192.168.0.12:3331
```

**FIRST TIME**

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT localhost:2331 2
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2332 4
```

```
FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3332
```

ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2333 6

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2334 8

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2335 10

**SERVER 192.168.0.12:3331 5**
**SERVER 192.168.0.12:3332 5**
**SERVER 192.168.0.12:3333 5**

## Testen von Weighted Distribution

SERVER 192.168.0.12:3331 HAT SICH REGISTRIERT
SERVER 192.168.0.12:3332 HAT SICH REGISTRIERT
SERVER 192.168.0.12:3333 HAT SICH REGISTRIERT

**FIRST TIME**

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT localhost:2331 2

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2332 4

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2333 6

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2334 8

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2335 10

**SERVER 192.168.0.12:3331 2**

**SERVER 192.168.0.12:3332 2**
**SERVER 192.168.0.12:3333 1**

**SESSION**

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT localhost:2331 2
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2332 4
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3333
ERGEBNIS 192.168.0.12:3333 → CLIENT 192.168.0.12:2333 6
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2334 8
```

```
SESSION
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2335 10
```

**SERVER 192.168.0.12:3331 4**
**SERVER 192.168.0.12:3332 4**
**SERVER 192.168.0.12:3333 2**

**AFTER 20 SECONDS**

```
DELETE SESSION 7faac844-2a8a-4afc-b0e3-
4e8f6a5b0132?192.168.0.12:3331
DELETE SESSION a212f010-0821-4880-b821-
219bd35bec5b?192.168.0.12:3332
DELETE SESSION 9906d333-7d50-4edd-9c4c-
9d34db3d22cd?192.168.0.12:3333
DELETE SESSION 2e19c2d9-a401-40e0-8b31-
a536739dd8a8?192.168.0.12:3332
DELETE SESSION e4b9b82e-d325-41d5-aa24-
717aebdd46ff?192.168.0.12:3331
```

**FIRST TIME**

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2331 :1 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT localhost:2331 2

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2332 :2 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2332 4

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2333 :3 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2333 6

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2334 :4 → SERVER 192.168.0.12:3331
ERGEBNIS 192.168.0.12:3331 → CLIENT 192.168.0.12:2334 8

FIRST TIME
ANFRAGE CLIENT 192.168.0.12:2335 :5 → SERVER 192.168.0.12:3332
ERGEBNIS 192.168.0.12:3332 → CLIENT 192.168.0.12:2335 10

**SERVER 192.168.0.12:3331 5**
**SERVER 192.168.0.12:3332 8**
**SERVER 192.168.0.12:3333 2**

## 2.4 Installation einer externen LoadBalancing-Software (Balance)

Wir haben uns bei der Softwareauswahl von folgender Liste:
https://www.inlab.de/articles/free-and-open-source-load-balancing-software-and-projects.html
für die Software "Balance" entschieden. Zur Durchführung haben wir drei Server und einen Loadbalancer als virtuelle Maschinen gestartet(Debian). Wir wollen apache-Anfragen loadbalancen also ist auf jedem Server Apache installiert.

Informationen zu der Umgebung:
Server 1: 10.0.105.151
Server 2: 10.0.105.153
Server 3: 10.0.105.159
Loadbalancer: 10.0.105.154

Der nächste Schritt ist das Installieren von "Balance" auf dem Loadbalancer:

apt-get install balance

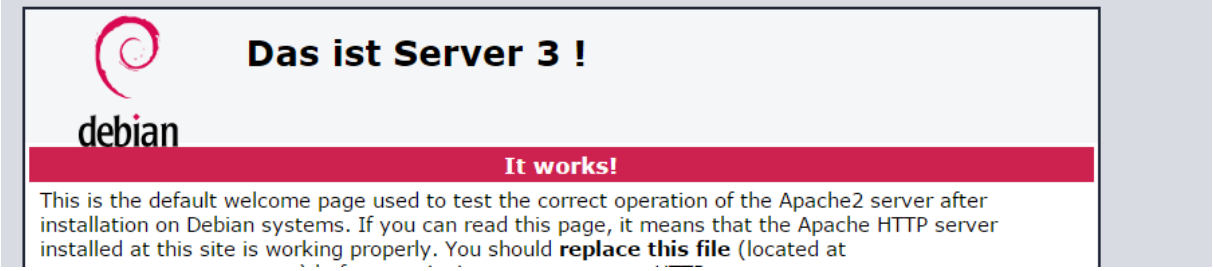
Um den Loadbalancer zu nutzen muss man folgenden Befehl ausführen:
balance port host1 host2 host3

In unserem Fall also:
balance -df 80 10.0.105.151 10.0.105.153 10.0.105.159

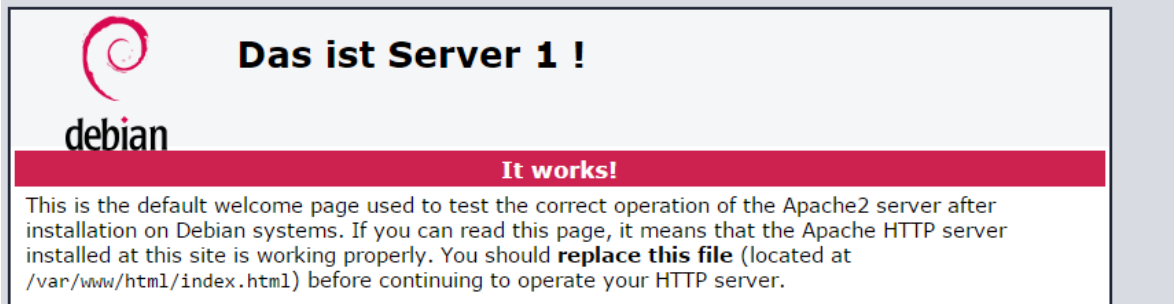
(mit –df schaltet man debugging und tracing informationen ein)

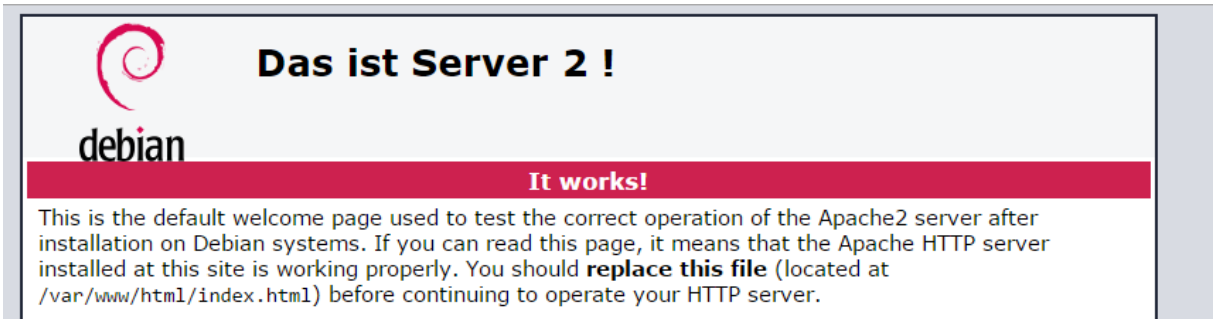
```
root@template:~# balance -df 80 10.0.105.151 10.0.105.153 10.0.105.159
argv[0]=balance
bindhost=NULL
source port 80
file /var/run/balance/balance.80.0.0.0.0 already exists
using AF_UNSPEC
the following channels are active:
  0  0 10.0.105.151:80:0
  0  1 10.0.105.153:80:0
  0  2 10.0.105.159:80:0
```

Zum Ausprobieren lade ich nun drei verschiedene html-Seiten auf die Server 1,2, und 3. In var/www/html/index.html verändere ich den Titel der Seiten zu "Das ist Server x !". Danach restarte ich apache2. Wenn ich nun die URL des Loadbalancers in meinem Browser aufrufe erhalte ich eine default Seite:




Wenn ich diese Seite neu lade erhalte ich dann auch die anderen Server und das zeigt uns, dass eine Lastverteilung stattgefunden hat:






Zudem sehe ich am Loadbalancer die Informationen zu den Verbindungen und Zugriffen:

```
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 2 ... trying group 0 channel 0 ... connect to channel 2 successful
trying group 0 channel 1 ... connect to channel 0 successful
connect to channel 1 successful
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 2 ... connect to channel 2 successful
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 0 ... trying group 0 channel 1 ... trying group 0 channel 2 ... connect to ch
annel 1 successful
connect to channel 0 successful
connect to channel 2 successful
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 0 ... connect to channel 0 successful
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 0 ... trying group 0 channel 2 ... connect to channel 0 successful
trying group 0 channel 1 ... connect to channel 2 successful
connect to channel 1 successful
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 0 ... trying group 0 channel 2 ... connect to channel 0 successful
connect to channel 2 successful
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 0 ... connect to channel 0 successful
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
connect from ::6817:2601:897f:0 clilen=16
trying group 0 channel 2 ... trying group 0 channel 0 ... trying group 0 channel 1 ... connect to ch
annel 2 successful
connect to channel 0 successful
connect to channel 1 successful
```