JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Load Balancing – DEZSYS 10

Systemtechnik
5BHITT 2015/16

Christoph Hackenberger
Rene Hollander

Version 1.0

Note:

Betreuer: T. Micheler

Begonnen am 12. Februar 2016

Beendet am 3. 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 (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

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.

1.4 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

Bewertung: 16 Punkte
- 2 Load Balancing Methoden (jeweils 6 Punkte)
- Session Persistenz (2 Punkte)
- Einsatz Load Balancing Software (2 Punkte)

1.5 Quellen

[1] "Praktische Arbeit 2 zur Vorlesung 'Verteilte Systeme' ETH Zürich, SS 2002", Prof.Dr.B.Plattner, übernommen von Prof.Dr.F.Mattern (http://www.tik.ee.ethz.ch/tik/education/lectures/VS/SS02/Praktikum/aufgabe2.pdf)
[2] http://www.tik.ee.ethz.ch/education/lectures/VS/SS02/Praktikum/loesung2.zip

2 Load Balancing Software – Nginx

Für die Zusatzaufgabe der Open-Source Load Balancer Applikation wurde nginx verwendet.
Zur Demonstration wurde ein Round Robin zwischen google.at und tgm.ac.at eingerichtet.

2.1 Nginx konfigurien

Nach Anleitung [1]

Zuerst Nginx installieren

apt-get install nginx

Danach die Datei /etc/nginx/sites-available/default wie folgt bearbeiten:

```
upstream backend {
        server tgm.ac.at;
        server google.at;
}

server {
   location / {
      proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
   }
}
```

Bei der Konfiguration der Location sollte man nicht vergessen den richtigen Hostname beim Request zu setzen ansonsten wird hier backend eingesetzt. [2]

Zum Schluss noch den Service neustarten

service nginx restart

Nun wird man wenn man die Adresse unseres Maschine im Browser aufrufen entweder auf google.at oder tgm.ac.at weitergeleitet.

3 Load Balancer für Pi Berechnung

Für die Aufgabe wurde ein Load Balancer entwickelt welcher mittels verschiedener Algorithmen ein Load Balancing für Services, welche Pi berechnen, implementiert.

3.1 Kommunikation

Die Kommunikation zwischen Service und LoadBalancer wurde mit SocketIO gelöst. SocketIO ist ein Framework für eventbasierte bidirektionale Kommunikation.

Dafür wird am LoadBalancer ein Server erstellt:

```
this.io = new SocketIO(port);
```

Sobald sich ein Client verbindet wird auf das register oder disconnect event gewartet:

```
socket.on('register', (data) => {
    socket.data = data;
    this.strategy.register(socket);
    debug("Service " + data.name + " registered")
});
```

```
socket.on('disconnect', () => {
    if (!socket.data) return;
    this.strategy.unregister(socket);
    debug("Service " + socket.data.name + " unregistered")
});
```

Durch diese beiden Events wird der ausgewählten Load Balancing Strategy mitgeteilt ob sich ein Service verbunden oder die Verbindung getrennt hat.

Beim Start eines Service baut er eine Verbindung zum LoadBalancer auf und sendet sobald er verbunden ist das register Event um seinen Namen mitzuteilen:

```
this.socket = SocketIO.connect(address);
this.socket.on('connect', () => {
    debug('Connected to loadbalancer');
    this.socket.emit('register', {name, options});
});
```

Auch lauscht der Service auf das execute Event um eine berechnung auszuführen. Sobald das execute Event empfangen wird, wird die Berechnung in einem separaten Thread gestartet. Wenn die Berechnung fertig ist, wird das Ergebniss über den callback zurück an den Load Balancer geschickt.

```
this.socket.on('execute', (data, callback) => this.onExecute(this, data,
callback));
onExecute(that, data, callback) {
    if (!data) data = {};
    data.service = that.name;
    if (!data.digits) data.digits = 10;
    debug('Got request for ' + data.digits + ' digits of pi');
    pi.calculate(data.digits, (estimate) => {
        debug('Finished calculating ' + data.digits + ' digits of pi');
        callback({service: that.name, pi: estimate})
    });
}
```

3.2 Webservice

Um den Load Balancer zu testen wird am Load Balancer ein RESTful Webservice zur verfügung gestellt: Über die Endpoints /pi/ und /pi/1000 kann Pi auf eine bestimmte Zahl an Nachkommastellen berechnet werden. Der Webservice wurde mit Koa.js erstellt:

```
let koa = new Koa();
let router = new Router();
router.get('/pi/', function *(next) {
    debug('Got new request');
    this.body = yield loadBalancer.execute();
    yield next;
});
router.get('/pi/:digits', function *(next) {
    debug('Got new request for ' + this.params.digits + ' digits');
    this.body = yield loadBalancer.execute({digits: this.params.digits});
    yield next;
});
koa.use(router.routes());
koa.use(router.allowedMethods());
koa.listen(args.webport);
```

3.3 Load Balancing Algorithmen

Round Robin

Als erster und einfachster Algorithmus wurde ein Round Robin implementiert. Hierbei wird einfach immer der nächste Service aus einer Liste genommen.

```
execute(data, callback) {
    this.sockets[this.current].emit('execute', data, callback);
    this.current++;
    this.current %= this.sockets.length;
}
```

Least Connections

Als zweiter Algorithmus wurde Least Connections gewählt. Hier wird beim Registrieren neuer Services ein neues Objekt erstellt, welches den Socket und die Anzahl der aktiven Verbindungen enthält, und in ein Array gespeichert. Wenn ein Service gewählt werden soll, wird das Array nach der Anzahl der aktiven Verbindungen der Services sortiert und das erste Service gewählt.

```
register(socket) {
    this.sockets.push({s: socket, active: 0});
}
execute(data, resolve, reject) {
    let serv = this.sockets.sort(function(a,b) {
        return a.active - b.active;
    })[0];
    serv.s.emit('execute', data, (data) => {
        serv.active--;
        resolve(data)
    });
    serv.active++;
}
```

Weighted Distribution

Als dritter und letzter Algorithmus wurde Weighted Distribution implementiert. Hier wird einfach beim Registrieren des Service auch ein Weight übergeben. Ansonsten funktioniert der Algorithmus ähnlich wie Least Connections, allerdings wird das Array aus dem der Service gewählt absteigend sortiert.

```
register(socket) {
    if (!socket.data.options.weight) socket.data.options.weight = 1;
    this.sockets.push({s: socket, weight: socket.data.options.weight});
}
execute(data, resolve, reject) {
    if (this.sockets == 0) {
        reject(new Error("Not a single service has registered by now"));
    }
    let serv = this.sockets.sort(function (a, b) {
        return b.weight - a.weight;
    })[0];
    serv.s.emit('execute', data, (data) => {
        serv.weight++;
        resolve(data)
    });
    serv.weight--;
}
```

3.4 Berechnung von Pi

Zur berechnung von Pi wird der Chudnovsky-Algorithmus verwendet. Er wurde in C++ mit der gmp Library implementiert. Da bei NodeJS native Dependencies erst bei der Installation kompiliert werden, gibt es einen Fallback in Javascript der Math.PI nach einer Zeit abhängig der Digits zurückliefert:

```
setTimeout(() => callback(Math.PI + ""), digits / 10)
```

3.5 Ausführen

Start a loadbalancer with:

 # Without logging

node loadbalancer.js --balancerport 5001 --webport 5000 --algorithm wdistrib

 # With logging

DEBUG=loadbalancer node loadbalancer.js --balancerport 5001 --webport 5000 --algorithm wdistrib

Start a service with:

 # Without logging

node service.js --hostname localhost --port 5001 --name service1 --weight 3

 # With logging

DEBUG=service node service.js --hostname localhost --port 5001 --name service1 --weight 3

4 GitHub Repo

https://github.com/ReneHollander/dezsys10-loadbalancing

5 Quellen

[1] How to Set Up Nginx Load Balancing, DigitialOcean Community, verfügbar unter: https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-load-balancing [abgerufen am 3.3.2016]

[2] Make nginx to pass hostname of the upstream when reverseproxying, serverfault, verfügbar unter: http://serverfault.com/questions/598202/make-nginx-to-pass-hostname-of-the-upstream-when-reverseproxying [abgerufen am 3.3.2016]