# Loadbalancer

*Loadbalancer gebruikmakend van het round-robin algoritme*

**Middleware specialisatiethema**
**Rob Juurlink IID7**
**2003 / 2004**

# INHOUDSOPGAVE


**1. Loadbalancer.....1**
**1.1. Opdrachtomschrijving.....1**
1.1.1. Inleiding.....1
1.1.2. Werkwijze.....1
**1.2. Wat is een loadbalancer.....2**
1.2.1. Poor man's load balancer .....2
1.2.2. Round Robin load balancer.....3
*Weighted Round Robin.....3*
*Least Connection.....3*
*Weighted Least Connection .....3*
*Adaptive weighted method.....3*
**1.3. Sessies.....4**
1.3.1. Sessie replicatie.....4
1.3.2. Sessie binding.....4
**1.4. loadgenerator.....5**
1.4.1. Siege.....5
*Performance test.....5*
**1.5. Ontwerp loadbalancer.....6**
1.5.1. Non-blocking.....6
1.5.2. Classes.....7
*LoadBalancer.....7*
*LoadBalancerConnection.....7*
*Opmerkingen.....7*
1.5.3. Classediagram.....8
**1.6. Testen.....9**
1.6.1. De opstelling.....9
1.6.2. Load genereren.....10
1.6.3. Testresultaat.....11
**2. Conclusie.....12**
**3. Referenties.....13**
**4. Bijlagen.....14**
**4.1. LoadBalancer.java.....14**
**4.2. LoadBalancerConnection.java.....16**

# 1. LOADBALANCER

## 1.1. OPDRACHTOMSCHRIJVING

### 1.1.1. Inleiding

Situaties veranderen: meer gebruikers, meer berichten, meer clients voor dezelfde server. Alles moet op termijn meer, groter, sneller, stabieler.

Dit levert dus vaak problemen op, aangezien de infrastructuur die beschikbaar is hier niet op berekend is, en aanpassing c.q. uitbreiding vaak niet mogelijk is vanwege structurele architectuur problemen.

Een pragmatische oplossing is dan het multipliceren van een server (of service) en een loadbalancer er voor laten zorgen dat de load zodanig verdeeld wordt over de servers, dat de gewenste performance gehaald wordt.

### 1.1.2. Werkwijze

Onderzoek m.b.v. het internet welke typen loadbalancers  er zijn en wat de belangrijkste kenmerken ervan zijn. Onderzoek de verschillende soorten algoritmes voor balancing en de situaties waarin die gebruikt worden.

Installeer een netwerk bestaande uit een client-systeem, een loadbalancer, en drie servers. Op de drie servers dient een Apache webserver te draaien, op het client systeem een stukje software dat kan functioneren als load-generator voor de webservers. De loadbalancer zit tussen het client systeem en de servers in, en dient de request van het clientsysteem te verdelen over de servers.

Overweeg de consequenties van session management en de wijze waarop de belasting van de servers gemeten kan worden.

De load-generator en loadbalancer software moeten nog geschreven worden. Gebruik een (grafische) tool voor performance meting om aan te tonen hoe de load verdeeld wordt. Op het internet zijn vele van deze tools te vinden.

# 1.2. WAT IS EEN LOADBALANCER

## 1.2.1. Poor man's load balancer

Dit is de meest simpele vorm van loadbalancing die eigenlijk geen loadbalancing genoemd mag worden. Door aan een domeinnaam meerdere IP-adressen te koppelen, wordt er bij elke DNS-aanvraag een willekeurig IP-adres uit een van de beschikbare adressen teruggegeven.

Een nadeel van deze methode is dat de belasting niet gelijkmatig verdeeld wordt. De reden hiervan is de manier waarop DNS werkt. Gebruikers krijgen het IP-adres dat bij een domeinnaam hoort via de internet provider via welke ze toegang tot het internet hebben. De DNS-server van de internetprovider kijkt eerst in z'n geheugen of deze naam misschien al aanwezig is. Is de naam al aanwezig en is de naam nog nog niet verlopen, dan wordt het IP-adres uit het geheugen opgegeven.

Is de "`time to live`" van de domeinnaam verlopen, of staat deze nog niet in het geheugen, dan voert de DNS-server van de provider een nieuwe aanvraag uit.

Opgesomd zijn de nadelen van deze aanpak:

- Alle servers moeten via een publiek IP-adres via het internet te bereiken zijn.
- Als een IP-adres uit de lijst van IP's die gekoppeld zijn aan een domeinnaam verwijderd wordt, duurt het nog maximaal tijd "`time to live`" voordat dit IP-adres niet meer gebruikt wordt. Hetzelfde geldt voor het toevoegen van een IP-adres. Zolang er een geldig IP-adres in de buffer van de DNS-server van de internetprovider aanwezig is, wordt een nieuw IP-adres nog niet doorgevoerd. Deze tijd kan wel oplopen tot een dag.
- Als veel gebruikers via dezelfde provider toegang hebben tot het internet, wordt de belasting niet gelijkmatig verdeeld.
- Het is niet mogelijk een gewicht mee te geven aan een bepaalde server.

## 1.2.2. Round Robin load balancer

Round Robin is het simpelweg één voor één de verschillende servers een verbinding doorgeven. Van dit type balancer bestaan een aantal subtypen die rekening kunnen houden met de capaciteit van de verschillende servers, het aantal connecties en een combinatie van beide.

### Weighted Round Robin

Als Round Robin, maar de servers met een hoger gewicht krijgen meer verbindingen te verwerken. Zo kan van tevoren aan worden gegeven hoe zwaar een server belast mag worden.

### Least Connection

Als er grote verschillen zijn in de tijdsduur van het verwerken van een request kan het zijn dat Round Robin niet meer goed werkt. Least Connection geeft de server die het minste aantal verbindingen heeft open staan het meeste werk.

### Weighted Least Connection

Een combinatie van Weighted Round Robin en Least Connection. Er is een mogelijkheid om aan te geven dat een bepaalde server zwaarder belast kan worden dan een andere en er wordt rekening gehouden met het aantal actieve verbindingen.

### Adaptive weighted method

De servers bepalen zelf hoe zwaar ze belast kunnen worden. Dit kan afhankelijk zijn van verschillende factoren zoals het aantal openstaande verbindingen, beschikbare bandbreedte, hoeveelheid vrij geheugen, het aantal draaiende processen en gebruik van de CPU.

# 1.3. SESSIES

Als de te benaderen webapplicatie / website gebruik maakt van sessies, is het balanceren van de belasting niet het simpel verdelen van de binnenkomende request over de aanwezige applicatieservers / webservers.

Bij gebruik van sessies weet de server tussen verschillende aanroepen dat het te maken heeft met dezelfde gebruiker. Er kan tijdens een sessie data opgeslagen worden aan de kant van de server, bijvoorbeeld de inhoud van een winkelwagentje tijdens het winkelen op een online webshop. Deze data is op dat moment fysiek alleen aanwezig op de server die de request afhandelt. Zou een volgende aanroep door de loadbalancer doorgestuurd worden naar een andere server, dan zouden deze sessie gegevens niet meer beschikbaar zijn.

Bovenstaand probleem kan op verschillende manieren opgelost worden.

## 1.3.1. Sessie replicatie

Bij sessie replicatie wordt de sessie beschikbaar gesteld aan alle servers die zich achter de loadbalancer bevinden. Dit beschikbaar stellen kan op verschillende manieren uitgevoerd zijn.

Een mogelijkheid is om de sessie data altijd in een database op te slaan. De gegevens zijn dan via de database bereikbaar voor de andere servers. Een nadeel van deze methode is dat het te langzaam is. Op het moment dat de data nog niet is weggeschreven kan er op een andere server alweer een request voor dezelfde sessie binnenkomen. Het opslaan van de data in een bestand op een gedeeld bestandssysteem heeft dezelfde nadelen.

Een betere oplossing is om sessies in het geheugen te repliceren. Er zijn bestaande protocollen die de data kunnen uitsturen naar de overige servers in het cluster. De data is daardoor altijd ook op de andere servers in het geheugen aanwezig.

## 1.3.2. Sessie binding

Nadat een sessie op een server opgezet is, regelt de loadbalancer dat de connectie altijd naar dezelfde server gaat. Deze manier van sessie binding wordt ook wel “sticky sessions” genoemd. Als de server overlijdt, wordt de request doorgestuurd naar een andere server in het cluster. Wordt er gebruik gemaakt dan sessie replicatie, dan kan de oude sessie weer opgepakt worden. Is er geen sessiereplicatie toegepast, dan is de oude sessie verloren.

# 1.4. LOADGENERATOR

De loadgenerator genereert http requests bij de Loadbalancer. Een loadgenerator doet in principe hetzelfde als wat honderden clients zouden doen. Bij veel verkeer zou een enkele webserver er behoorlijk onder lijden, waardoor de prestaties zouden verminderen of dat zelfs de webserver er mee ophoudt. De Loadbalancer voorkomt dit.

## 1.4.1. Siege

Voor het genereren van de load, wordt het stress test programma Siege[1] versie 2.55 gebruikt.

Uit de Siege handleiding:

> *Siege is an http/https regression testing and benchmarking utility. It was designed to let web developers measure the performance of their code under duress, to see how it will stand up to load on the internet. It lets the user hit a web server with a configurable number of concurrent simulated users. Those users place the webserver "under siege." The duration of the siege is measured in transactions, the sum of simulated users and the number of times each simulated user repeats the process of hitting the server. Thus 20 concurrent users 50 times is 1000 transactions, the length of the test. Performance measures include elapsed time of the test, the amount of data transferred ( including headers ), the response time of the server, its transaction rate, its throughput, its concurrency and the number of times it returned OK.*

De test wordt gestart door commando's aan het commando `siege` mee te geven.

### Performance test

De test wordt gestart door het volgende commando uit te voeren:

```
siege --url=http://10.0.20.2 --concurrent=10 --delay=1 -r10


The server is now under siege..      done.
Transactions:                   1000 hits
Availability:                 100.00 %
Elapsed time:                   7.09 secs
Data transferred:              41000 bytes
Response time:                  0.03 secs
Transaction rate:             141.04 trans/sec
Throughput:                  5782.79 bytes/sec
Concurrency:                    4.90
Successful transactions:        1000
Failed transactions:               0
```

1 Siege is een GNU/Linux programma. Te downloaden vanaf http://www.joedog.org/siege/

# 1.5. ONTWERP LOADBALANCER

De loadbalancer moet nadat er bepaald is naar welke server de verbinding moet gaan, een in- en uitgaande verbinding opzetten. Het opzetten van een verbinding is mogelijk door gebruik te maken van classes in Java die overweg kunnen TCP verbindingen.

We gebruiken hiervoor de nieuwe `bibliotheek java.nio.channels` (new I/O) die sinds Java versie 1.4 standaard aanwezig is. Deze nieuwe bibliotheek heeft in vergelijking met de “oude” classes uitgebreidere en betere mogelijkheden voor het werken met streams.

Omdat we verbindingen opzetten die in twee richtingen werken, zijn de nieuwe methoden `getChannel` erg handig.

## 1.5.1. Non-blocking

Het grootste voordeel van het gebruik van de nieuwe bibliotheek is de mogelijkheid tot niet-blokkerende in- en uitvoer van data. De gangbare Engelse term hiervoor is `non-blocking`.

Niet blokkerende in- en uitvoer heeft een veel betere performance en het voorkomt dat het bij een foute verbinding gaat lijken alsof het programma vastgelopen is. Dit betekent geen gedoe met Threads meer om een verbindingsfout af te kunnen vangen.

Om met niet blokkerende in- en uitvoer te kunnen werken, moet er gebruikt gemaakt worden van het Channel object in plaats van het Stream object.

## 1.5.2. Classes

Het brainstormen heeft een ontwerp met twee classes opgeleverd. De loadbalancer gaat gebruik maken van het `round-robin` algoritme. Hieronder volgt een beschrijving van de twee classes:

### LoadBalancer

Deze class bevat de main methode. Na de start van de code, creëert deze class een serversocket en bindt deze aan poort 8080 (poorten < 1024 zijn niet te benaderen voor een niet root processen).

Elke keer als er een aanvraag(request) vanaf een webbrowser binnenkomt. bepaalt de methode `getServer()` volgens het `round-robin` algoritme welke server de aanvraag verder gaat afhandelen.

Als is bepaald welke server de aanvraag verder gaat afhandelen, wordt de verbinding tussen loadbalancer-browser en loadbalancer-webserver gecreëerd en opgezet. Daarna wordt een nieuw object van LoadBalancerConnection gecreëerd en opgestart in een Thread. Dit object handelt de verbinding verder af.

### LoadBalancerConnection

Een object van dit type dat draait in een `Thread` en verzorgt de vebinding tussen de webbrowser(client) en die naar een van de servers.

De opdracht vanaf de client wordt ingelezen en doorgestuurd naar de webserver. Daarna wordt net zo lang in stapjes van 1024 bytes data gelezen van de server en doorgestuurd naar de webbrowser(client).

Als er niet meer data vanaf de server binnenkomt, wordt de verbinding verbroken en zal het object verdwijnen.

### Opmerkingen

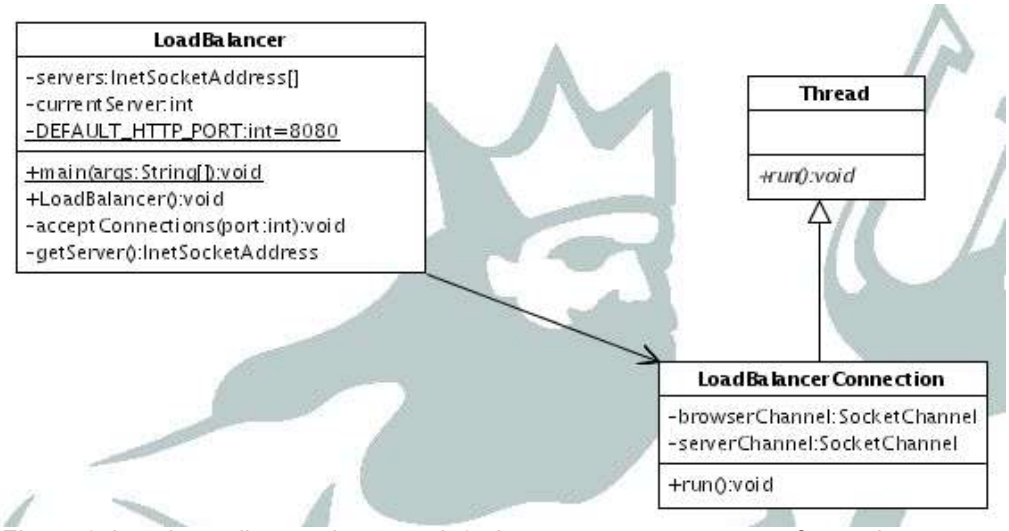
In dit ontwerp worden van eventueel opgetreden fouten een melding gemaakt en verder genegeerd.

Ook heeft de `keep-alive` vraag van de browser geen effect. De verbinding wordt altijd meteen verbroken nadat de server klaar is met het verzenden van de data.

De loadbalancer controleert niet of de webservers op de opgegeven adressen nog actief zijn. Als een van de webserver uitvalt, zal het resultaat zijn dat er af en toe een `Connection Refused` fout optreedt.

### 1.5.3. Classediagram

Het ontwerp is in een classediagram afgebeeld, zie figuur 1.




*Figuur 1, het classediagram bestaat uit 2 classes, waarvan er een erft van de bestaande Thread class.*

# 1.6. TESTEN

Om de loadbalancer te testen moeten er een aantal webserver opgezet worden. Deze webservers moeten rechtstreeks bereikbaar zijn voor de machine waarop de loadbalancer draait.

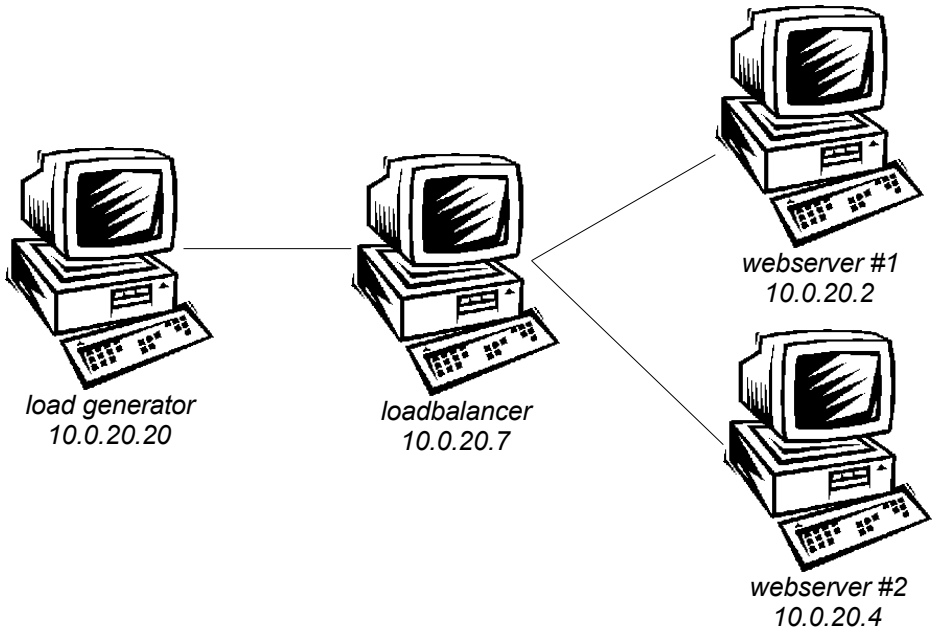
Er wordt hier verder niet uitgelegd hoe een webserver geïnstalleerd moet worden. Dit valt buiten dit verslag.

Tijdens het testen is de drempel voor het loggen op `SEVERE` gezet. We willen niet hebben dat bij het gebruik de performance gehinderd wordt door het loggen.

## 1.6.1. De opstelling

De opstelling is niet zoals in de opdracht staat beschreven een opstelling van 3 webservers en een loadbalancer. In plaats daarvan zijn er twee webservers aanwezig, omdat er niet meer computers in m'n prive netwerk aanwezig zijn.

De opstelling is schematisch weergegeven in het figuur hieronder.



## 1.6.2. Load genereren

Op beide webserver is een bestandje `info.php` gecreëerd dat op dezelfde locatie staat. Dit bestand levert een pagina op met een grootte van 35058 bytes. Zie figuur x voor een fragment van deze webpagina.


*Figuur 2, een deel van de info.php website.*

Het testen van de loadbalancer wordt gestart nadat onderstaande commando is uitgevoerd:

```
siege --url=http://10.0.20.7:8080/info.php --concurrent=10 --benchmark
      --time=10S
```

| ***Parameters*** | ***Omschrijving*** |
|---|---|
| concurrent=10 | Het aantal gesimuleerde gelijktijdige gebruikers. |
| benchmark | Voer de test uit zonder vertraging tussen de oproepen door. |
| time=3600S | De totale tijd de test moet duren. Opgegeven in seconden. |
| url=http://10.0.20.7:8080/info.php | De URL inclusief poortnummer van de loadbalancer. |

### 1.6.3. Testresultaat

Het eerste resultaat toont het maximaal aantal hits op 1 server in 10 seconden. Om straks eerlijk te kunnen vergelijken zit in deze configuratie de loadbalancer er wel tussen, maar worden de aanvragen slechts doorgestuurd naar één webserver.

De loadbalancer vertraagt het aantal aanvragen ongeveer met een factor vijf. Dit komt omdat de loadbalancer niet geoptimaliseerd is.

In totaal heeft deze configuratie 218 aanvragen kunnen verwerken. Beide machines waarop een webserver draait zijn praktisch aan elkaar gelijk.

```
siege --url=http://10.0.20.2:8080/info.php --concurrent=10 --benchmark --time=10S
** Siege 2.55
** Preparing 10 concurrent users for battle.
The server is now under siege...
Lifting the server siege..      done.
Transactions:                    218 hits
Availability:                 100.00 %
Elapsed time:                  10.49 secs
Data transferred:            7397522 bytes
Response time:                  0.47 secs
Transaction rate:              20.78 trans/sec
Throughput:                 705197.54 bytes/sec
Concurrency:                    9.74
Successful transactions:         218
Failed transactions:               0
```

Nu nogmaals, maar dan met beide webservers in de loadbalancer geconfigureerd:

```
siege --url=http://10.0.20.2:8080/info.php --concurrent=10 --benchmark --time=10S
**************************************************
siege: could not open /home/rob/.siegerc
run 'siege.config' to generate a new .siegerc file
**************************************************
** Siege 2.55
** Preparing 10 concurrent users for battle.
The server is now under siege...
Lifting the server siege..      done.
Transactions:                    225 hits
Availability:                 100.00 %
Elapsed time:                  10.31 secs
Data transferred:            8284959 bytes
Response time:                  0.44 secs
Transaction rate:              21.82 trans/sec
Throughput:                 803584.74 bytes/sec
Concurrency:                    9.57
Successful transactions:         225
Failed transactions:               0
```

Na de tweede test is het aantal hits in 10 seconden in vergelijking tot de vorige test bijna gelijk. Dit komt omdat de loadbalancer de vertragende factor is. De webserver verwerkt de aanvragen erg snel.

Wat wel is te zien in de logfiles van beide webservers is dat de aanvragen gelijkmatig verdeeld worden over beide webservers. Het aantal aanvragen is in dezelfde tijd per server gehalveerd, waardoor de server minder belast is geweest.

# 2. CONCLUSIE

Beide webservers worden minder belast doordat de aanvragen gelijkmatig over beide webservers verdeeld worden. Het aantal af te handelen aanvragen is bij gelijke belasting daardoor per server gehalveerd.

Het doel van de loadbalancer is dus geslaagd. De load per webserver is gedaald!

Een minpuntje is dat de loadbalancer in vergelijking tot de webserver erg langzaam werkt. Dit komt waarschijnlijk deels doordat de loadbalancer op een tragere machine draait, de loadbalancer is Java geschreven is en deels omdat de loadbalancer niet geoptimaliseerd is.

# 3. REFERENTIES

[1] **THESERVERSITE**, *In memory Session Replication*, februari 2004, http://www.theserverside.com/articles/article.jsp?l=Tomcat

[2] **SUN MICROSYSTEMS**, *New I/O APIs*, februari 2004, http://java.sun.com/j2se/1.4.2/docs/guide/nio/index.html

# 4. BIJLAGEN

## 4.1. LoadBalancer.java

```
1   /*
2    * LoadBalancer.java
3    */
4
5   import java.nio.channels.*;
6   import java.net.*;
7   import java.util.*;
8   import java.util.logging.*;
9
10  /**
11   * De loadbalancer.
12   * Start luisteren op een opgegeven poort. De verbindingen die op deze poort
13   * binnenkomen worden doorgestuurd naar een van de aanwezige servers.
14   */
15  public class LoadBalancer {
16
17      // Beschikbare servers instellen
18      private InetSocketAddress[] servers = {
19          new InetSocketAddress("10.0.20.2",80),
20          new InetSocketAddress("10.0.20.4",80),
21      };
22
23      private int currentServer = 0;
24
25      private final Logger logger = Logger.getLogger(this.getClass().getName());
26      private static final int DEFAULT_HTTP_PORT = 8080;
27
28      /** Start luisteren op standaard poort. */
29      public LoadBalancer() throws Exception {
30          acceptConnections(DEFAULT_HTTP_PORT);
31      }
32
33
34      /**
35       * Start het luisteren op de ingestelde poort.
36       * @param port Luisteren op poortnummer.
37       */
38      private void acceptConnections(int port) throws Exception {
39
40          // Selector for incoming requests
41          Selector acceptSelector = Selector.open();
42
43          // Create a new server socket and set to non blocking mode.
44          ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
45          serverSocketChannel.configureBlocking(false);
46
47          // Bind the server socket to the local host and port
48          InetSocketAddress isa = new InetSocketAddress(InetAddress.getLocalHost(), port);
49          serverSocketChannel.socket().bind(isa);
50
51          // Registreer acceptatie op de server socket.
52          // Accepteer verbindingen.
53          serverSocketChannel.register(acceptSelector, SelectionKey.OP_ACCEPT);
54
55          // De select method komt terug als een hierboven geregistreerde
56          // operatie optreedt.
57          while(true) {
58
59              logger.info("Start select (wachten op verbinding)");
```

```
60              acceptSelector.select();
61
62              // Someone is ready for I/O, get the ready keys
63              logger.info("Er is verbinding gemaakt. De selected keys uitlezen.");
64              Set keys = acceptSelector.selectedKeys();
65
66              // Loop de keys uit de collection door en verwerk de requests.
67              for (Iterator i = keys.iterator(); i.hasNext();) {
68
69                  SelectionKey selectionKey = (SelectionKey) i.next();
70                  i.remove();
71
72                  //Maak verbinding
73                  if (selectionKey.isAcceptable()) {
74
75                      // The key indexes into the selector so you
76                      // can retrieve the socket that's ready for I/O
77                      ServerSocketChannel browserSocketChannel =
                            (ServerSocketChannel) selectionKey.channel();
78                      SocketChannel browserChannel = browserSocketChannel.accept();
79
80                      logger.info("Browser(client) heeft verbinding gemaakt." +
                            browserChannel.socket().getInetAddress().toString());
81
82                      // Bepaal naar welke server we gaan verbinden
83                      InetSocketAddress serverAddress = getServer();
84                      SocketChannel serverChannel = SocketChannel.open();
85
86                      // Na een connect is het socket bruikbaar
87                      serverChannel.connect(serverAddress);
88
89                      logger.info("Verbinding naam webserver opgezet: " +
                            serverChannel.socket().getInetAddress().toString());
90
91                      // Er zijn nu twee channels.
92                      // In een nieuwe thread de verbinding af laten handelen
93                      Thread connection = new LoadBalancerConnection(browserChannel,
                            serverChannel);
94                      connection.start();
95
96                      logger.info("Verbinding is apparte thread is gestart.");
97
98                  }
99              }
100         }
101     }
102
103
104     /**
105      * Een server kiezen uit een van de beschikbare.
106      * Er wordt gebruik gemaakt van het roudrobin algoritme.
107      * @return De hostname en het poortnummer.
108      */
109     private InetSocketAddress getServer() {
110         InetSocketAddress server = servers[currentServer];
111         currentServer++;
112         currentServer = (currentServer % (servers.length));
113
114         logger.info("Server die request gaat afhandelen: " + server.toString());
115
116         return server;
117     }
118
119
```

# 4.2. LoadBalancerConnection.java

```
1   import java.lang.Thread;
2   import java.nio.*;
3   import java.nio.channels.*;
4   import java.nio.charset.*;
5   import java.util.logging.*;
6
7   /**
8    * Een connectie tussen browser en server afhandelen.
9    * Nadat de verbinding verbroken is, sterft de Thread af.
10   */
11  public class LoadBalancerConnection extends Thread {
12
13      private final Logger logger = Logger.getLogger(this.getClass().getName());
14
15      // Een channel voor communicatie tussen de browser en de loadbalancer
16      // en een channel vor de communicatie tussen loadbalancer en webserver.
17      // Een Channel is een kanaal vor twee richtingen
18      private SocketChannel browserChannel;
19      private SocketChannel serverChannel;
20
21      // Charset and decoder for US-ASCII
22      private static Charset charset = Charset.forName("US-ASCII");
23      private static CharsetDecoder decoder = charset.newDecoder();
24
25      /**
26       * Een loadbalancer connectie instellen.
27       * Beide kanalen instellen.
28       * @param theBrowserChannel Kanaal tussen browser van de client en loadbalancer.
29       * @param theServerChannel Kanaal tussen loadbalancer en webserver.
30       */
31      public LoadBalancerConnection(SocketChannel theBrowserChannel,
            SocketChannel theServerChannel) {

32          browserChannel = theBrowserChannel;
33          serverChannel = theServerChannel;
34      }
35
36      /**
37       * De code die uitgevoerd wordt in de Thread.
38       * Start het doorzenden naar de browser en webserver.
39       */
40      public void run() {
41
42          try {
43
44              logger.info("Thread gestart.");
45
46              // De data lezen in stukjes van 1Kb
47              ByteBuffer buffer;
48              buffer = ByteBuffer.allocateDirect(1024);
49              buffer.clear();
50
51              // Lees het commando van de client (browser).
52              // We gaan er hier vanuit dat de request niet langer dan 1Kb is.
53              // De request kan langer zijn bij het posten van een formulier.
54              browserChannel.read(buffer);
55
56              // De gelezen inhoud printen
57              buffer.flip();
58              CharBuffer cb = decoder.decode(buffer);
59
60              logger.info("Opdracht client ingelezen: '" + cb + "'");
61
62              // Stuur de data door naar de server
63              buffer.rewind();
```

```
64              serverChannel.write(buffer);
65
66              logger.info("Opdracht doorgezonden naar de webserver");
67
68              // Zolang er data is (de gebruikte buffer is vol), deze doorsturen.
69              // Geen data meer, dan de lus afbreken.
70              while (true) {
71
72                  // Wacht op de reactie van de server
73                  buffer.clear();
74                  serverChannel.read(buffer);
75
76                  // De gelezen inhoud printen
77                  buffer.flip();
78                  CharBuffer cb2 = decoder.decode(buffer);
79                  logger.info("Antwoord terug van de webserver: '" + cb2 + "'");
80
81                  // Geef de reactie door aan de client
82                  buffer.rewind();
83                  browserChannel.write(buffer);
84
85                  logger.info("Antwoord tergug naar de browser(client) gestuurd.");
86
87                  // Als de buffer niet helemaal gevuld is, nemen we aan dat er niet meer data
88                  // aanwezig was en breken we uit de while lus.
89                  if (buffer.position() != buffer.capacity()) {
90                      logger.info("Buffer was nog niet vol, einde verbinding...");
91                      break;
92                  }
93              }
94
95              // Sluiten van de verbindingen.
96              serverChannel.close();
97              browserChannel.close();
98
99          }
100         catch (Exception e) {
101             logger.severe("Fout bij verbinding: " + e.toString());
102         }
103         finally {
104             // Zorgen dat de verbindingen altijd gesloten worden.
105             try {
106                 if (serverChannel!=null) serverChannel.close();
107             }
108             catch (Exception e) {}
109             try {
110                 if (browserChannel!=null) browserChannel.close();
111             }
112             catch (Exception e) {}
113         }
114     }
115 }
```

```
120     /**
121      * Startmethode.
122      * Parameters worden genegeerd.
123      */
124     public static void main(String[] args) {
125 
126         // De rootlogger instellen.
127         // Alleen tijdens het testen de logger aanzetten.
128         Logger logger;
129         logger = Logger.getLogger("");
130         logger.setLevel(Level.INFO);
131         logger.setUseParentHandlers(false);
132         logger.info("De logger is ingesteld.");
133 
134         try {
135             LoadBalancer loadBalancer = new LoadBalancer();
136         }
137         catch(Exception e) {
138             logger.severe("Kan de server niet starten! error: " + e.toString());
139         }
140     }
141 }
```