Technologisches Gewerbe Museum

VSDB

Load Balancing

Author: Siegel Hannah & Soyka Wolfram

January 8, 2015

Contents

Contents

1 Working time

Estimated working time

Task	**Person**	**Time in hours**
UML	Siegel	1
	Soyka	0
Basic Load Balancing Scaffold	Siegel	3
	Soyka	3
Weighted Round Robin	Siegel	1
	Soyka	1
Agent Based Adaptive	Siegel	2
	Soyka	2
Session Persistence	Siegel	2
	Soyka	1
Testing CPU	Siegel	2
	Soyka	0
Testing RAM	Siegel	0
	Soyka	2
Testing I/O (Harddisk)	Siegel	2
	Soyka	1
Testing	Siegel	3
	Soyka	2
Final Documentation	Siegel	2
	Soyka	1
Total	Siegel	16
	Soyka	11
Total Team		**27 hours**

Final working time

Task	**Person**	**Time in hours**
UML	Siegel	2
	Soyka	0
Basic Load Balancing Scaffold	Siegel	1
	Soyka	0
Log4j	Siegel	2
	Soyka	0
Starter	Siegel	1
	Soyka	0
Weighted Round Robin	Siegel	3
	Soyka	0
Agent Based Adaptive	Siegel	2
	Soyka	0
Session Persistence	Siegel	2
	Soyka	0
Testing CPU	Siegel	4
	Soyka	0
Testing RAM	Siegel	1
	Soyka	0
Testing I/O (Harddisk)	Siegel	2
	Soyka	0
Testing	Siegel	-
	Soyka	-
Final Documentation	Siegel	3
	Soyka	0
Total	Siegel	24
	Soyka	0
Total Team		**24 hours**

2 Task Description

The task description can be found under https://elearning.tgm.ac.at/mod/assign/view.php?id=31104. It is german and therefore we did not copy it.

3 Load Balancer

3.1 Load Balancing Algorithms

We decided to do the Weighted Round Robin and the Agent based Adaptive Algorithm.

3.1.1 Weighted Round Robin

The Weighted Round Robin simply loops the servers and distributes the load. The distribution through is done using the weight. So if there are two servers, S1 with the Eight 30 and S2 with the weight 70, the first Server gets 30% and the second gets 70% of the clients requests.
We have done this using an array and an iterator. The Array for the above mentioned example looks like:

```
{"S1","S1","S1","S2","S2","S2","S2","S2","S2","S2"}
```

We have done the Array more like this, so the Load is still more equally distributed:

```
{"S1","S2","S1","S2","S1","S2","S2","S2","S2","S2"}
```

The output of this `allocation` will be seen if our program is started with the Weighted Round Robin Algorithm and Algorithm Logging turned on.

3.1.2 Agent based Adaptive

The Agent based Adaptive Algorithm is a bit more complicated. The Load-Balancer is fetching the information of the Servers of what their current load is. Normally this is done using a file, but because we ware only simulating it, we simply used a `getCurrentWeight()` method.
The weight is calculated.

3.1.3 Session Persistance

We have also implemented a Session Persistence.

4 Doing the Task

At first we have done a UML. Because we need to practice it, we really did it on our own so it has not been perfect. We hereby request a quick review to this UML at one point in time, so we can learn out of our failures.

4.1 UML Design

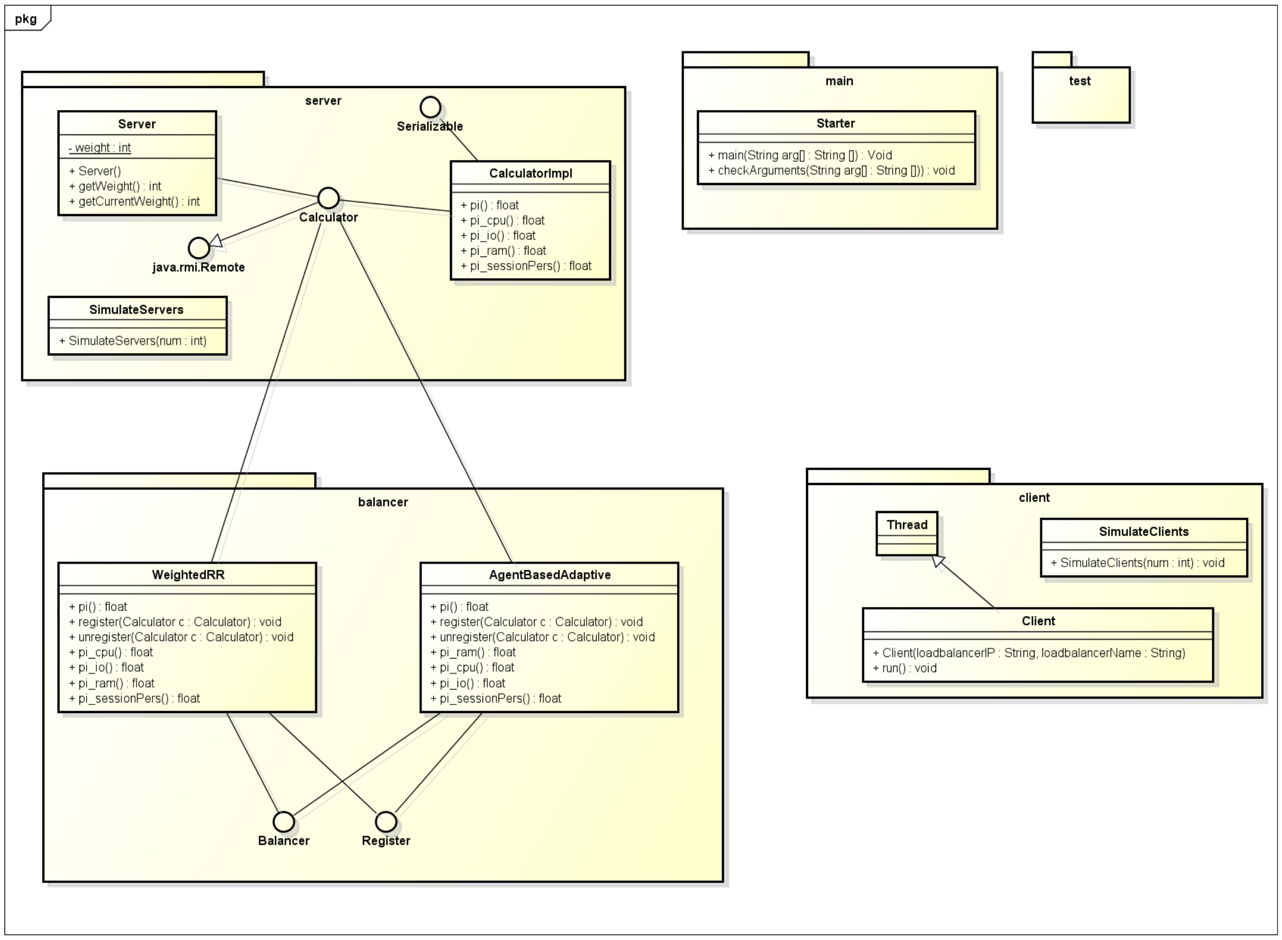

pkg
server
Server
- weight : int
+ Server()
+ getWeight() : int
+ getCurrentWeight() : int
Serializable
CalculatorImpl
+ pi() : float
+ pi_cpu() : float
+ pi_io() : float
+ pi_ram() : float
+ pi_sessionPers() : float
Calculator
java.rmi.Remote
SimulateServers
+ SimulateServers(num : int)
main
Starter
+ main(String arg[] : String []) : Void
+ checkArguments(String arg[] : String []) : void
test
balancer
WeightedRR
+ pi() : float
+ register(Calculator c : Calculator) : void
+ unregister(Calculator c : Calculator) : void
+ pi_cpu() : float
+ pi_io() : float
+ pi_ram() : float
+ pi_sessionPers() : float
AgentBasedAdaptive
+ pi() : float
+ register(Calculator c : Calculator) : void
+ unregister(Calculator c : Calculator) : void
+ pi_ram() : float
+ pi_cpu() : float
+ pi_io() : float
+ pi_sessionPers() : float
Balancer
Register
client
Thread
SimulateClients
+ SimulateClients(num : int) : void
Client
+ Client(loadbalancerIP : String, loadbalancerName : String)
+ run() : void

Figure 1: Class Diagram First Try

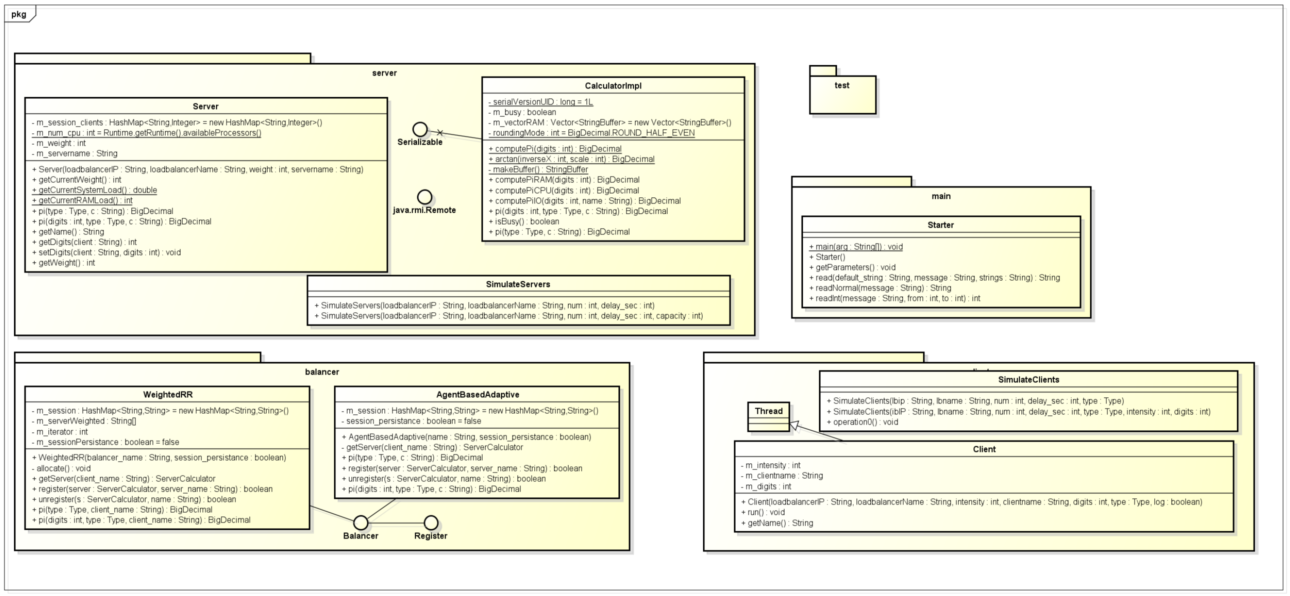

pkg
server
Server
- m_session_clients : HashMap<String,Integer> = new HashMap<String,Integer>()
- m_num_cpu : int = Runtime.getRuntime().availableProcessors()
- m_weight : int
- m_servername : String
+ Server(loadbalancerIP : String, loadbalancerName : String, weight : int, servername : String)
+ getCurrentWeight() : int
+ getCurrentSystemLoad() : double
+ getCurrentRAMLoad() : int
+ pi(type : Type, c : String) : BigDecimal
+ pi(digits : int, type : Type, c : String) : BigDecimal
+ getName() : String
+ getDigits(client : String) : int
+ setDigits(client : String, digits : int) : void
+ getWeight() : int
Serializable
java.rmi.Remote
CalculatorImpl
- serialVersionUID : long = 1L
- m_busy : boolean
- m_vectorRAM : Vector<StringBuffer> = new Vector<StringBuffer>()
- roundingMode : int = BigDecimal.ROUND_HALF_EVEN
+ computePi(digits : int) : BigDecimal
+ arctan(inverseX : int, scale : int) : BigDecimal
- makeBuffer() : StringBuffer
+ computePiRAM(digits : int) : BigDecimal
+ computePiCPU(digits : int) : BigDecimal
+ computePiIO(digits : int, name : String) : BigDecimal
+ pi(digits : int, type : Type, c : String) : BigDecimal
+ isBusy() : boolean
+ pi(type : Type, c : String) : BigDecimal
SimulateServers
+ SimulateServers(loadbalancerIP : String, loadbalancerName : String, num : int, delay_sec : int)
+ SimulateServers(loadbalancerIP : String, loadbalancerName : String, num : int, delay_sec : int, capacity : int)
test
main
Starter
+ main(arg : String[]) : void
+ Starter()
+ getParameters() : void
+ read(default_string : String, message : String, strings : String) : String
+ readNormal(message : String) : String
+ readInt(message : String, from : int, to : int) : int
balancer
WeightedRR
- m_session : HashMap<String,String> = new HashMap<String,String>()
- m_serverWeighted : String[]
- m_iterator : int
- m_sessionPersistance : boolean = false
+ WeightedRR(balancer_name : String, session_persistance : boolean)
- allocate() : void
+ getServer(client_name : String) : ServerCalculator
+ register(server : ServerCalculator, server_name : String) : boolean
+ unregister(s : ServerCalculator, name : String) : boolean
+ pi(type : Type, client_name : String) : BigDecimal
+ pi(digits : int, type : Type, client_name : String) : BigDecimal
AgentBasedAdaptive
- m_session : HashMap<String,String> = new HashMap<String,String>()
- session_persistance : boolean = false
+ AgentBasedAdaptive(name : String, session_persistance : boolean)
- getServer(client_name : String) : ServerCalculator
+ pi(type : Type, c : String) : BigDecimal
+ register(server : ServerCalculator, server_name : String) : boolean
+ unregister(s : ServerCalculator, name : String) : boolean
+ pi(digits : int, type : Type, c : String) : BigDecimal
Balancer
Register
SimulateClients
+ SimulateClients(lbip : String, lbname : String, num : int, delay_sec : int, type : Type)
+ SimulateClients(iblP : String, lbname : String, num : int, delay_sec : int, type : Type, intensity : int, digits : int)
+ operation0() : void
Thread
Client
- m_intensity : int
- m_clientname : String
- m_digits : int
+ Client(loadbalancerIP : String, loadbalancerName : String, intensity : int, clientname : String, digits : int, type : Type, log : boolean)
+ run() : void
+ getName() : String

Figure 2: Final Class Diagram

4.2 Basic Load Balancing Scaffold

4.2.1 Starter class

We have done a `Starter` class, where the components were stared and a dialog for getting the informations from the user.

4.2.2 Interfaces

We have defined some Interfaces:

- Calculator
- Server
- Register
- ServerCalculator (Calculator and Server interface)
- Balancer (Register and Calculator interface)

4.2.3 Basic Load Balancer

The Load Balancer uses RMI:

```
Balancer x;
// Exporting stub
try {
        x = (Balancer) UnicastRemoteObject.exportObject(this, 0);

        // creating the registry
        Registry registry = LocateRegistry.createRegistry(1099);

        // binding Balancer
        registry.bind(balancer_name, x);
} catch (AlreadyBoundException e) {
        Log.error("Load Balancer already bound, can not bound it again",e);
} catch (RemoteException e) {
        Log.error("There was a remote exception while exporting the Object",e);
}
```

This is as usually, we basically have copied it from last year's task.

4.2.4 Client

The Client is a Thread. It has a specific number of digits which it demands from the server. It also has a specific intensity with which the client demands the Services.
There are different Types of

4.2.5 Server

```
// calculator object
m_calc = new CalculatorImpl();

//fetch registry
Registry registry = LocateRegistry.getRegistry(loadbalancerIP,1099);

//fetch balancer
m_balancer = (Balancer) registry.lookup(loadbalancerName);

ServerCalculator x;
//bind server
x = (ServerCalculator) UnicastRemoteObject.exportObject(this, 0);

m_balancer.register(x, servername);
```

4.2.6 Pi-Calculation

The Pi Calculation has been copied from last years task.

4.3 Weighted Round Robin

The Weighted Round Robin Load Balancer has an allocation method:

```
//all the servers
HashMap<String, Integer> availiable_servers = new HashMap<String, Integer>();

//fetching the information from the servers: <Servername, Weigth>
for (Entry entry : m_servers.entrySet()) {
    ServerCalculator obj = (ServerCalculator) entry.getValue();
        availiable_servers.put(entry.getKey().toString(),obj.getWeight());
}

//calculate total weigth (Sum from all the Weights)
...
Log.logAlg("Allocation has found out a total amount of " + number + " units");

//generate servers array
String servers[] = new String[number];
int already_allocated = 0;
while (already_allocated < number) {
                ...
}
//set the allocated values
  while (m_iterator != 0){}
  m_serverWeighted = servers;
```

And he gets the next Server from the getServer method.

```
public ServerCalculator getServer(String client_name) throws RemoteException{
  if ( m_servers.size() <= 0){
        Log.logMax("There is no server which could handle this request!");
        return null;
  }
  else{
        //if session persistance should be used,
        // and the client has already a server as a peer,
        // and this server is still availiable...
                choosen_server = m_servers.get(m_session.get(client_name));
                //if the server has to big of a load (90)
                if(!(choosen_server.getCurrentWeight()<90)){
                        // choose different server
                }

        //else
                //get the next server 'in line'
                choosen_server = (ServerCalculator)
                    m_servers.get(server_weighted[m_iterator]);
                m_iterator++;
                //reset iterartor
                if(m_iterator==server_weighted.length)
                        m_iterator = 0;
        }
  return choosen_server;
}
```

4.4 Agent Based Adaptive

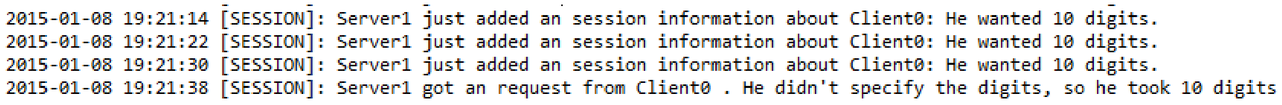
The Agent Based Adaptive has only the getServer method which returns a Server.

```
if ( m_servers.size() <= 0){
  Log.logMax("There is no server which could handle this request!");
  return null;
}
else{
        //if session persistance should be used,
        // and the client has already a server as a peer,
        // and this server is still availiable...
                choosen_server = m_servers.get(m_session.get(client_name));
                //if the server has to big of a load (90)
                if(!(choosen_server.getCurrentWeight()<90)){
                        // choose different server
                }
        //else
          int smallest_load = 101;

          //find the server with the smallest load
          for (Entry entry : m_servers.entrySet()) {
             ...
          }

          // get the server object
          choosen_server = m_servers.get(servers_capacities);

          return choosen_server;
```

4.5 Session Persistence

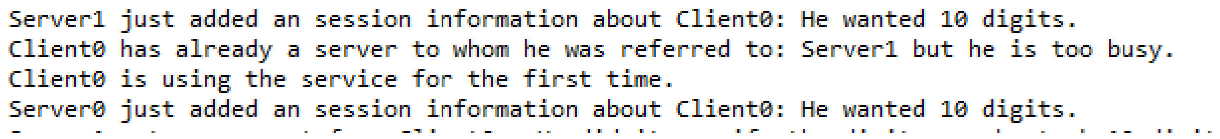
We have implemented the session persistence. We thought, that in order to 'proof' that it works, we will simply let the client make 3 requests and see if it works and then let him make 1 request without any specifications of it's digits.
The server will then know (because the session persistance is implemented and therefore the client always gets the same server which stores the information) how many digits to give back to the client. This is actually working. Only if the servers load is bigger than 90, the Client

will get a new peer, but the information is going to be transferred.



```
2015-01-08 19:21:14 [SESSION]: Server1 just added an session information about Client0: He wanted 10 digits.
2015-01-08 19:21:22 [SESSION]: Server1 just added an session information about Client0: He wanted 10 digits.
2015-01-08 19:21:30 [SESSION]: Server1 just added an session information about Client0: He wanted 10 digits.
2015-01-08 19:21:38 [SESSION]: Server1 got an request from Client0 . He didn't specify the digits, so he took 10 digits
```

Figure 3: Session Persistence - digit information successfully retrieved



```
Server1 just added an session information about Client0: He wanted 10 digits.
Client0 has already a server to whom he was referred to: Server1 but he is too busy.
Client0 is using the service for the first time.
Server0 just added an session information about Client0: He wanted 10 digits.
```

Figure 4: Session Persistence - if the server was too busy

4.6 Testing with CPU, RAM and I/O

We asked the User when starting what kind of type should be done. Either the CPU will be used, a lot of RAM or an I/O should be implemented.

4.6.1 CPU

We simply did a useless Pi computing after returning the right value. That this works can be seen in Figure 5.

```java
public BigDecimal computePiCPU(int digits) {
  //do some useless things
  computePi(digits*100000);

  return computePi(digits);
}
```

4.6.2 RAM

The Ram was overfloated using a String Buffer:

```java
public BigDecimal computePiRAM(int digits) {
  //adding stuff to RAM
  StringBuffer b = makeBuffer();
  m_vectorRAM.addElement(b);
  return computePi(digits);
}
```

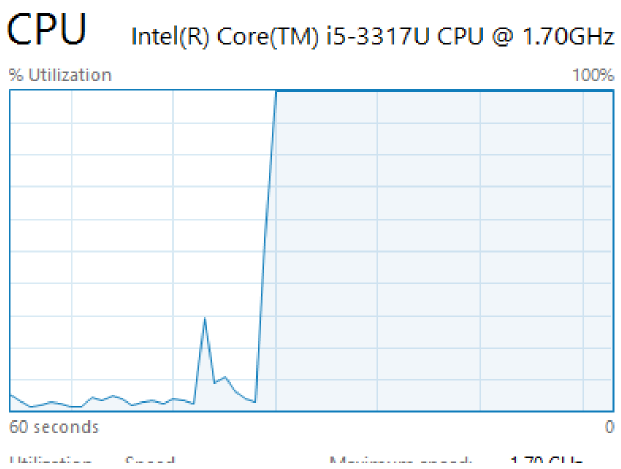


Figure 5: CPU usage

4.6.3 IO

```
public BigDecimal computePiIO(int digits,String name) {
              StringBuffer b = makeBuffer();
              FileOutputStream fo = null;
              PrintWriter pw = null;
              try {
                     File f = new File("C:/temp/myfile"+name+".txt");
                     f.deleteOnExit();
                     fo = new FileOutputStream(f);
                     pw = new PrintWriter(fo,true);
                     pw.println(b.toString());
                     pw.close();
                     fo.close();
              } catch (FileNotFoundException e) {
                     Log.error("Could not find File", e);
              } catch (IOException e) {
                     Log.error("There occured a IO Exception...", e);
              }

              return computePi(digits);
}
```

4.6.4 Implementing CPU, RAM and IO

RAM

The Ram was computed using the following code:

```
public static int getCurrentRAMLoad(){
  int res = 0;
  Runtime runtime = Runtime.getRuntime();
  long ram_availiabe=runtime.totalMemory();
  long ram_used = runtime.totalMemory() - runtime.freeMemory();

  res  = (int)( ((100.0 / (double)ram_availiabe)*(double)ram_used)+0.5);
  return res;
}
```

CPU
As it can be seen in the Problems section, the CPU with Java and Windows was not possible, at least not with normal APIs. We than decided not to implement this, because this took too long.

5 Testing

Testing has not be finished yet. Until now everything worked fine.

5.1 JUnit

We have not yet found the time to implement JUnit Test cases.

6 Comparison between the Load Balancing Algorithms

Both Algorithms work equally fine.
The Weighted Round Robin is hard to test, because we only do have two laptops. For calculating the Current Weight, the CPU of the system is needed, but we were not able to retrieve it. So we simulated it, which is not perfect either.

We think that the Weighted Round Robin Algorithm is great whenever the Connections are short and require only easy calculations. The Agent Based Adaptive Algorithm would be better if there were time consuming Requests, which request more difficult operations.
Other than that, we also have seen that the session persistence takes the scheduling away from the Balancer, because it kind of skips the algorithm if there is one server already assigned to a client.

7 Problems

7.0.1 Log4j

We were not able to include Log4j. There was a new version and we couldn't find a useful hint on the internet. Therefore we simply wrote our own Log-class.

7.0.2 RMI over Network didn't work

The RMI Network communication was not working.
The bug was fixed when the Virtual machine networks were turned off.

Mr. Borko then told us, that the interface was simply missing.

7.0.3 Calculating the CPU Load

We were not able to calculate the cpu load. The most promising answer was found under [1], but this simply returned -1.
Other than that external libraries must be used.