total:26/31

Distributed systems I
Winter Term 2019/20

G2T1 – Assignment 2 (theoretical part)

Felix Bühler	Clemens Lieb	Steffen Wonner	Fabian Bühler
2973410	3130838	2862123	2953320

16. November 2019

9

1 - Parameter Passing and RMI

a)

1 (i) `[1, 2, 3, 4, 5]`

1 (ii) `[5, 4, 3, 4, 5]`

1 (iii) `[5, 4, 3, 2, 1]` *or* `[1, 2, 3, 4, 5]` depending on the order of restore operations. The first option appears when y is restored before x. The second option appears when the restoration order is reversed.

b)

The result printed to the system console is:

```
$ java StringServer
1: B
1 2: BRS
2 3: ABSBRS
```

We can deduce that all objects that extend UnicastRemoteObject are sent as reference, all other objects are sent as value. In the first and last two lines the parameter is a StringBuffer object which does not implement UnicastRemoteObject and is therefore sent as value. In the third line the parameter is a StringServer object which does implement UnicastRemoteObject and because of this is sent as reference.

c)

1 The first line shows that even "complex java objectsäre passed by value so long as they are not remote objects. This implies that even very large non-remote objects would be serialized and passed by value (e.g. Arrays with hundreds of thousands of elements). Such a behaviour incurs large overheads, especially if only parts of the passed value are accessed by the remote procedure.

Felix Bühler
Clemens Lieb
Steffen Wonner
Fabian Bühler

1 The lines two and three demonstrate the reference-passing behaviour within java's RMI implementation. Notably, remote objects are passed by reference. Calls to the passed remote objects are routed back to the JVM the actual object resides in. This can lead to large overheads if a lot of calls are made to the referenced object.

10

2 - Chord System

a)

4

Node 4:

1	11
2	11
3	11
4	16
5	23

Node 11:

1	16
2	16
3	16
4	19
5	31

Node 16:

1	19
2	19
3	23
4	26
5	4

Node 19:

1	23
2	23
3	23
4	31
5	4

Node 23:

1	26
2	26
3	31
4	31
5	11

Node 26:

1	31
2	31
3	31
4	4
5	11

Node 31:

1	4
2	4
3	4
4	11
5	16

b)

1

1. node 4 → node 16
2. node 16 → node 19
3. node 19 → node 23

c)

2

Node 24:

1	26
2	26
3	31
4	4
5	11

The finger table of node 16 will change eventually while the one for node 23 will change immediately.

d)

2 If both nodes join simultaneously they are between the same two nodes (4 and 11). If they now try to each update their predecessor and successor then we have a race condition while updating a linked list. This can leave the list (or in this case ring) in an inconsistent state that makes routing impossible!

Felix Bühler
Clemens Lieb
Steffen Wonner
Fabian Bühler

e)

0 This code updates the FT entries of p for the former successor's predecessor which has the id q.

```
for i ← 0 to ⌊log2(q − p)⌋ do
    FT_p[i] ← q
end for
```

the assignment explicitly asks for 9 lines of code

This code assumes that only q has been inserted between p and the former successor. The obvious issue arising from that (namely that multiple insertions are not correctly handled), can be easily remedied by repeating the consistency check after updating the finger table. As presented the code is already eventually consistent, though.

1

f)

As outlined in the forum, it's assumed that the node ids within the system are at least vaguely evenly distributed under the given hash function. Given that assumption and the assumption that there are at least three nodes in the system, it's reasonable to assume that no node is responsible for an adress space larger than $2^m - 1$. Carrying that assumption onwards, any entity will be stored on two nodes, because the smallest $p \leq ID_1(e)$ will resolve to different nodes in the separate chord rings. why?

This property holds true under the assumption that no node is responsible for an adress space larger than the largest lookup shortcut, which is the offset for the second chord ring. If that property is not given, the insert is not guaranteed to store an entity's copy on two different nodes. That failure can be seen when considering $ID_1(n_1) + 2 = ID_1(n_2) + 1 = ID_1(n_3)$ for three nodes $\{n_1, n_2, n_3\}$. It's likely that an inserted entity on a chord-ring structure with 8 entries will be inserted into n_3, violating the preferred property.

7

3 - Name Services

a)

2

	Iterative resolution	Recursive resolution
1. Lookup	10	10
2. Lookup	10	10
3. Lookup	8	8

b)

2

	Iterative resolution	Recursive resolution
1. Lookup	10	10 (8 fast, 2 slow)
2. Lookup	10	8 (6 fast, 2 slow)
3. Lookup	8	8 (6 fast, 2 slow)

c)

i.

Felix Bühler
Clemens Lieb
Steffen Wonner
Fabian Bühler

1
```
$ dig NS uni-stuttgart.de
...
;; ANSWER SECTION:
uni-stuttgart.de.       2191    IN      NS      minnehaha.rhrk.uni-kl.de.
uni-stuttgart.de.       2191    IN      NS      dns3.belwue.de.
uni-stuttgart.de.       2191    IN      NS      dns1.belwue.de.
uni-stuttgart.de.       2191    IN      NS      dns0.uni-stuttgart.de.
uni-stuttgart.de.       2191    IN      NS      dns1.uni-stuttgart.edu.
...
```

There are 5 server responsible for the DNS entry of 'uni-stuttgart.de':

- minnehaha.rhrk.uni-kl.de
- dns3.belwue.de
- dns1.belwue.de
- dns0.uni-stuttgart.de
- dns1.uni-stuttgart.edu

ii.

2
```
$ dig -x 129.69.216.249
...
;; ANSWER SECTION:
249.216.69.129.in-addr.arpa. 86397 IN   PTR     ipvslogin.informatik.uni-
    stuttgart.de.
...
```

PTR is usefull for resolving an IP address to a domain or hostname (reverse DNS lookup). The given IP resolves to: 'ipvslogin.informatik.uni-stuttgart.de'

0 **iii.**

not a manual iterative address resolution

```
$ dig +trace www.uni-stuttgart.de
...
.                       347153  IN      NS      h.root-servers.net.
.                       347153  IN      NS      l.root-servers.net.
.                       347153  IN      NS      d.root-servers.net.
.                       347153  IN      NS      m.root-servers.net.
.                       347153  IN      NS      e.root-servers.net.
.                       347153  IN      NS      k.root-servers.net.
.                       347153  IN      NS      i.root-servers.net.
.                       347153  IN      NS      b.root-servers.net.
.                       347153  IN      NS      c.root-servers.net.
.                       347153  IN      NS      a.root-servers.net.
.                       347153  IN      NS      f.root-servers.net.
.                       347153  IN      NS      g.root-servers.net.
.                       347153  IN      NS      j.root-servers.net.
...
;; Received 553 bytes from 192.168.0.1#53(192.168.0.1) in 28 ms

de.                     172800  IN      NS      z.nic.de.
de.                     172800  IN      NS      f.nic.de.
de.                     172800  IN      NS      s.de.net.
de.                     172800  IN      NS      a.nic.de.
de.                     172800  IN      NS      l.de.net.
```

Felix Bühler
Clemens Lieb
Steffen Wonner
Fabian Bühler

```
de.                     172800  IN      NS      n.de.net.
...
;; Received 802 bytes from 2001:500:12::d0d#53(g.root-servers.net) in 22 ms

uni-stuttgart.de.       86400   IN      NS      dns0.uni-stuttgart.de.
uni-stuttgart.de.       86400   IN      NS      dns1.belwue.de.
uni-stuttgart.de.       86400   IN      NS      dns1.uni-stuttgart.edu.
uni-stuttgart.de.       86400   IN      NS      dns3.belwue.de.
uni-stuttgart.de.       86400   IN      NS      minnehaha.rhrk.uni-kl.de.
...
;; Received 619 bytes from 2001:67c:1011:1::53#53(n.de.net) in 25 ms

www.uni-stuttgart.de.   3600    IN      A       129.69.8.19
...
uni-stuttgart.de.       3600    IN      NS      dns0.uni-stuttgart.de.
uni-stuttgart.de.       3600    IN      NS      minnehaha.rhrk.uni-kl.de.
uni-stuttgart.de.       3600    IN      NS      dns1.uni-stuttgart.edu.
uni-stuttgart.de.       3600    IN      NS      dns3.belwue.de.
uni-stuttgart.de.       3600    IN      NS      dns1.belwue.de.
...
;; Received 1365 bytes from 129.143.2.10#53(dns1.belwue.de) in 25 ms
```

The path the lookup took was:

1. 192.168.0.1
2. g.root-servers.net
3. n.de.net
4. dns1.belwue.de

what is the IP address of the "leaf" domain?