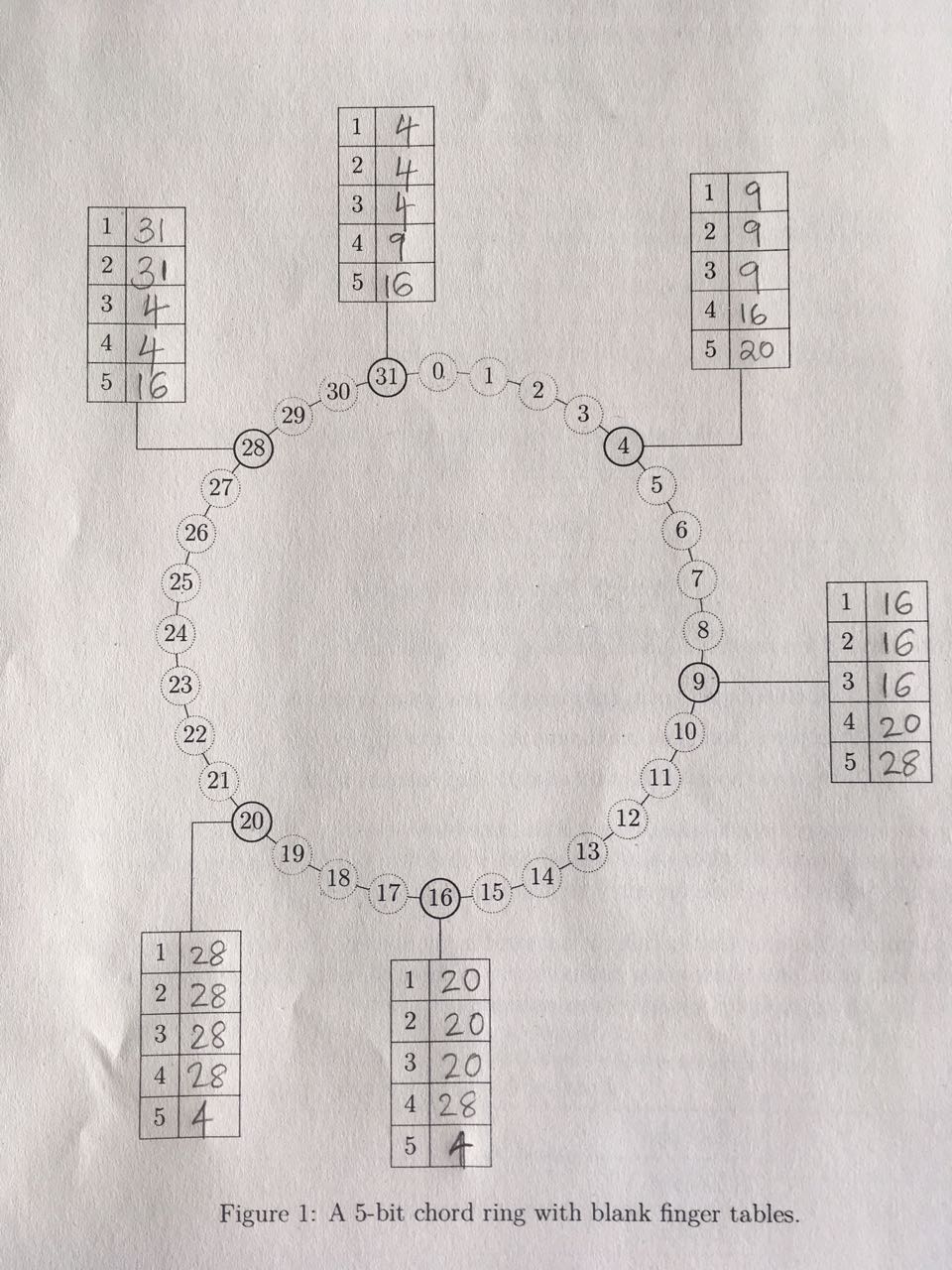
Assignment 2

Team G3T2

* Vishnudatha Kanjur (3250569)
* Julian Hörz (3316908)

Solutions:

1. Parameter Passing and RMI
   1. Its assumed that in the pseudo code accesses array a of size 5 with indexes from 0 to 4
      1. Call-by-value : [13, 6, 5,12, 56]
      2. Call-by-reference : [13, 6,120,12, 56]
      3. Amount of data transferred during the execution is as follows:
         1. Line 1: 35Bytes(Call to remote method getAllVideoNames())+ 30Bytes \* 650 (650 videos names of 30 Bytes) = 19535Bytes
         2. Line 2: No exchange (local variable initialization)
         3. Line 3: (35 Bytes(remote method call)+30Bytes(Pass string vName to Server)+30Bytes(Pass string containing video name from Server)+400Bytes(Pass string containing video description from Server)+800MBytes(Pass string containing video data from Server))\*650 times = 520000322Bytes
         4. Line 4: No exchange (local call of function)
      4. Amount of data transferred during the execution is as follows:
         1. Line 1: 35Bytes(Call to remote method getAllVideoNames())+ 30Bytes \* 650 (650 videos names of 30 Bytes) = 19535Bytes
         2. Line 2: No exchange (local variable initialization)
         3. Line 3: (10 Bytes(remote reference IVideo)+35 Bytes(remote method call)+30Bytes(Pass string vName to Server))\*650 times = 48750Bytes
         4. Line 4: ((35 Bytes(remote method call)+ 30Bytes(Pass string containing video name from Server) +35 Bytes(remote method call) +400Bytes(Pass string containing video description from Server)) \*650 times = 325000Bytes
2. Chord System a)

b) k=30 and n=4

As per the lookup algorithm at slide 23 of chapter 4, the sequence will be 4->20->28->31.

The function call will be as follows:

4.find\_successsor(30)

-> 4. closest\_preceeding\_node.find\_successsor(30)

->20.find\_successor(30)

->20. closest\_preceeding\_node.find\_successsor(30)

->28.find\_successor(30)

-> 31

c) FT table for 23 will be as follows:

|  |  |
| --- | --- |
| 1 | 28 |
| 2 | 28 |
| 3 | 28 |
| 4 | 31 |
| 5 | 9 |

9,16,20,28 needs to be changed while adding new node at 23.

d) As proximity routing is used, node 31 and 16 are near to each other (both located at Stuttgart), it should be used to resolve the key k=17 and then go to node 20. The new sequence to resolve k=17 will be 31->16->20 using the proximity algorithm.

e)

1. Name Services
   1. Without caching:

Iterative lookup



In the above figure, -> indicates the request from client 1.

-> indicates the request from client 2.

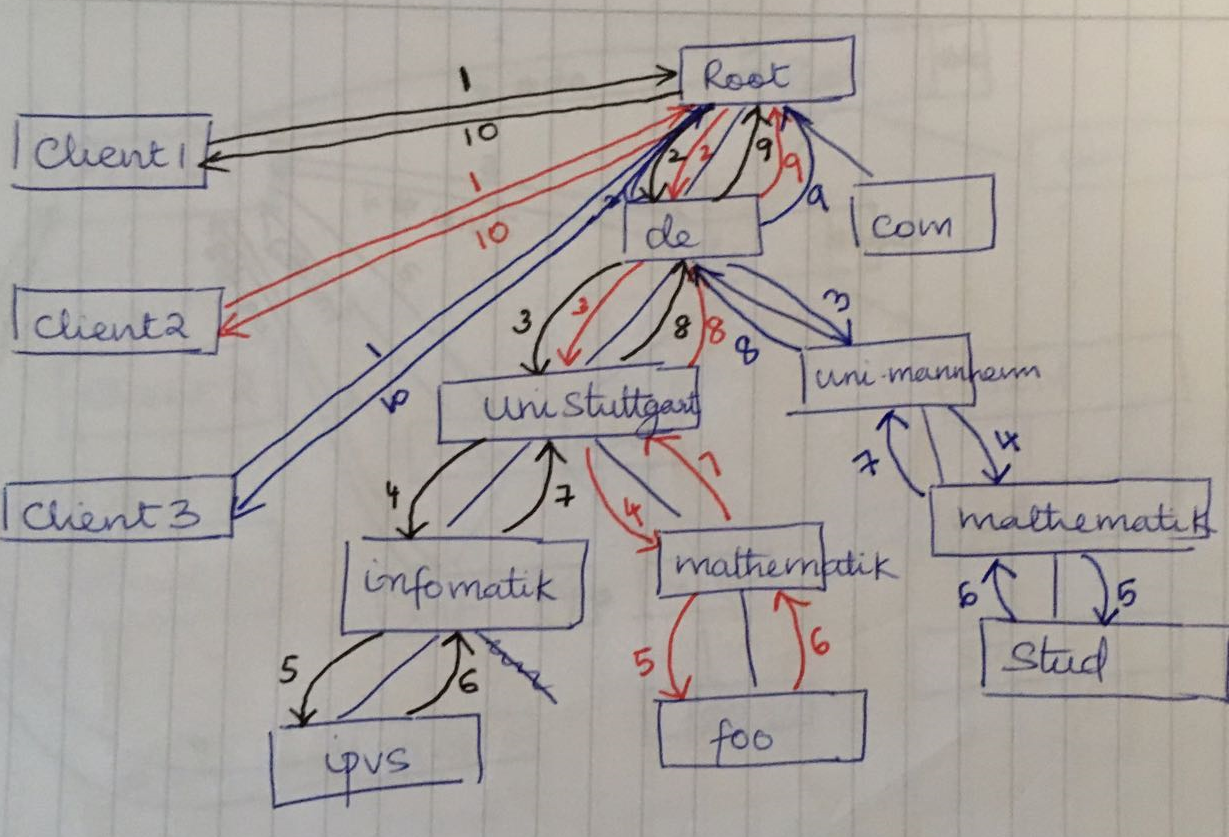
-> indicates the request from client 3.

Recursive lookup

In the below figure, -> indicates the request from client 1.

-> indicates the request from client 2.

-> indicates the request from client 3.



|  |  |  |
| --- | --- | --- |
|  | Iterative lookup | Recursive lookup |
| 1. Lookup | 10 | 10 |
| 2. Lookup | 10 | 10 |
| 3. Lookup | 10 | 10 |

* 1. With server side caching

Iterative lookup:

Lookup 1 is the same as without caching as we assume that initially cache is empty.

Lookup 2, firstly it requests the root server which only has the address of its child server de in the cache. Then client queries de server which has in the cache the address of uni-stuttgart server. Then the client queries uni.stuttgart server and then the uni-Stuttgart server has the address of its child server mathematik(not from cache). Client again queries the foo to resolve the address and gets response. So number of requests = 10 with each having communication time of 40ms.

Lookup 3, firstly it requests the root server which only has the address of its child server de in the cache. Then client queries de server which has the address of uni-mannheim server(not from cache). Then the client queries uni-mannheim server and then the uni-mannheim server has the address of its child server mathematik(not from cache). Client again queries the stud to resolve the address and gets response. So number of requests = 10 with each having communication time of 40ms.

Recursive lookup:

Lookup 1 is the same as without caching as we assume that initially cache is empty. Number of requests is 10, out of which 2(initial request and final response) are 40 ms time and rest 8(intermediate recursive calls) are 5ms time.

Lookup 2, firstly it requests the root server(40ms) which has the addresses of its child server de and uni-stuttgart. Then it recursively calls uni-stuttgart server(5ms) which doesn’t have the address of mathematic server in cache but as mathematik is the child server, it recursively queries it(5ms). Mathematik server queries recursively the foo server(5ms) to resolve the address and gets response to mathematik server(5ms). And the response is sent as per the order of requests. Cache will also be updated accordingly for further requests. So number of requests = 8 with the following response time.

Client2-> root - 40ms

Root->uni-stuttgart -5ms

Uni-stuttgart-> mathematik -5ms

Mathematik-> foo -5ms

Foo->mathematik -5ms

Mathematik->uni-stuttgart -5ms

Uni-stuttgart->root -5ms

Root->Client2 -40ms

Lookup 3, firstly it requests the root server(40ms) which has the addresses of its child server de and uni-stuttgart. So root server queries the de server(5ms) to obtain the address of uni-mannheim. Then it recursively calls uni-mannheim server(5ms) which doesn’t have the address of mathematik server in cache but as mathematik is the child server, it recursively queries it(5ms). Mathematik server queries recursively the stud server(5ms) to resolve the address and gets response to mathematik server(5ms). And the response is sent as per order of the requests. Cache will also be updated accordingly for further requests. So number of requests = 10 with the following response time.

Client3-> root - 40ms

Root->de -5ms

De->uni-mannheim -5ms

uni-mannheim -> mathematik -5ms

Mathematik-> stud -5ms

stud->mathematik -5ms

Mathematik->uni-mannheim -5ms

Uni-mannheim->de -5ms

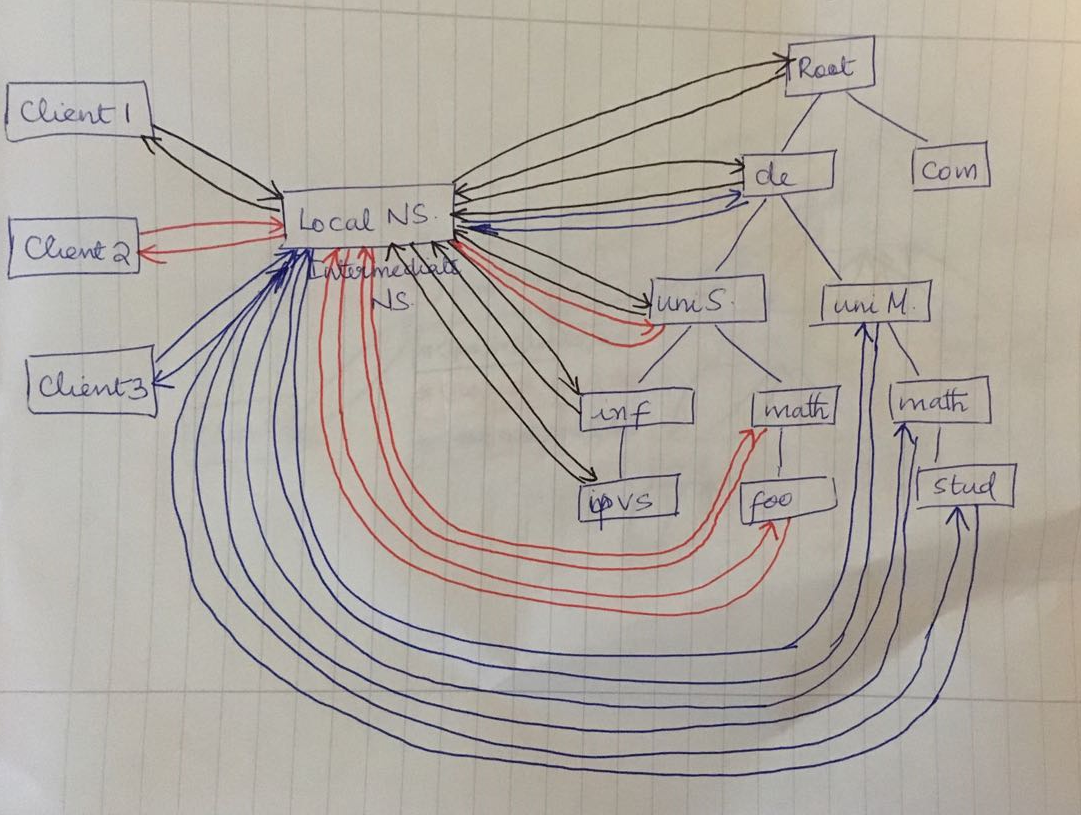
De->root -5ms

Root->Client2 -40ms

|  |  |  |
| --- | --- | --- |
|  | Iterative lookup | Recursive lookup |
| 1. Lookup | 10\*40ms=400ms | 2\*40ms+8\*5ms=120ms |
| 2. Lookup | 10\*40ms=400ms | 2\*40ms+6\*5ms=110ms |
| 3. Lookup | 10\*40ms=400ms | 2\*40ms+8\*5ms=120ms |

* 1. With local name server for clients

Iterative lookup



In the above figure, -> indicates the request from client 1.

-> indicates the request from client 2.

-> indicates the request from client 3.

Here in the figure, cache at intermediate name server is not shown. But after processing the lookup1, cache will be #root,#de,#uni-stuttgart,#infomatik,#ipvs. It uses, #uni-stuttgart for the lookup2 and iteratively process the request further. So after lookup2, cache will be #root,#de,#uni-stuttgart,#infomatik,#ipvs,#mathematik,#foo. For lookup3, it uses #de and iteratively process the request further. So after lookup3, cache will be #root,#de,#uni-stuttgart,#infomatik,#ipvs,#mathematik,#foo,#uni-mannheim,#mathematik,#stud.

Recursive lookup

In the below figure, -> indicates the request from client 1.

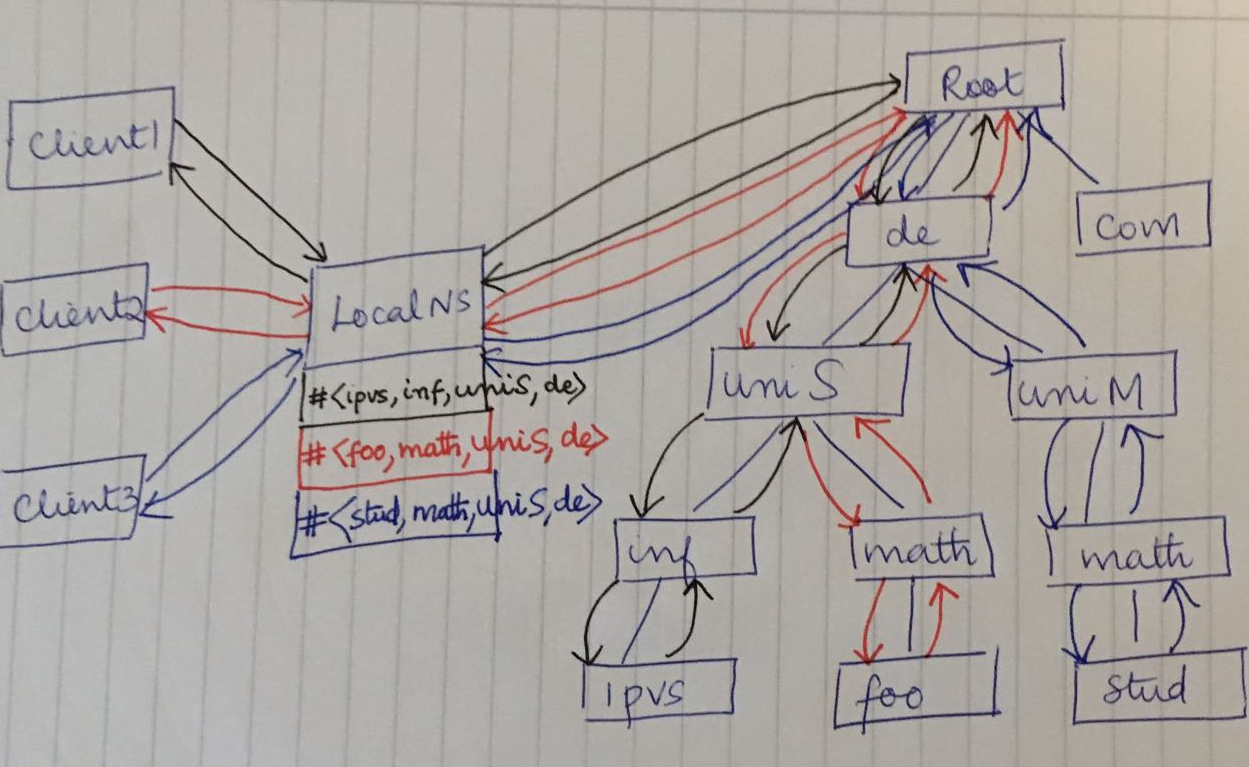
-> indicates the request from client 2.

-> indicates the request from client 3.

After lookup1, cache will be #<ipvs,infomatik,uni-stuttgart,de>. So for lookup2, intermediate server has to query from the root again.

After lookup2, cache will be #<ipvs,infomatik,uni-stuttgart,de>,#<foo,mathematik,uni-stuttgart,de>. So for lookup3, intermediate server has to query from the root again.

After lookup3, cache will be #<ipvs,infomatik,uni-stuttgart,de>,#<foo,mathematik,uni-stuttgart,de>,#<stud,mathematik,uni-mannheim,de>



|  |  |  |
| --- | --- | --- |
|  | Iterative lookup | Recursive lookup |
| 1. Lookup | 2\*5ms+10\*40ms=410ms | 2\*40ms+10\*5ms=130ms |
| 2. Lookup | 2\*5ms+6\*40ms=250ms | 2\*40ms+10\*5ms=130ms |
| 3. Lookup | 2\*5ms+8\*40ms=330ms | 2\*40ms+10\*5ms=130ms |