Exercise 2 **Exercises Peer-to-Peer-Systems and Security (SS2012)**

Dr. Heiko Niedermayer
Lehrstuhl für Netzarchitekturen und Netzdienste
Technische Universität München

Thursday 21.5. 2012
Hand-in: Thursday 31.5. in lecture or by mail to niedermayer – at – net.in.tum.de
Exercise: Monday 4.6. in lecture

Rules: There will be four exercise sheets. You have to hand-in 70 % of the assignments, attend at least 3 exercise courses and present a solution in the exercise course to get the 0.3 bonus. Up to 3 persons may hand in one sheet.

Task 1 CoolSpots Munich III
This time you should solve the task to organize the CoolSpots Munich network with a Distributed Hash Table. Use spot_ID = h(GPS coordinate of spot) to store the items. As item descriptions are rather short, the data is stored on the DHT nodes and not only on the node that contributed the item.

a) How can you find an item that is directly at your GPS coordinate?
b) Items at my exact GPS coordinates are not too useful, propose a change to the item storage and lookup mechanism to efficiently find items close to a given GPS coordinate?
c) Now, assume a multi-dimensional ID space. What is changing?

Solution:
a)
A user stores a spot in the DHT with a put command.
Spot s = new Spot(GPS Coordinate, SpotName, Author, Rating, Description, Date, …);
DHT.put(s.getKey(), s)

The put command operates on a KBR and routes the message to the node holding the key of the data item. It may be replicated by the DHT to k close neighbors to circumvent failures.
The DHT.get operates in simular fashion, but the node gets a return message with either the list corresponding spots or a failure.

The spots are stored at the hash of the coordinate. Since the lower bits of the coordinate are not relevant (due to noise and due to irrelevance of a high resolution that is smaller than even a small spot) Therefore, we apply an accurracy function that removes the bitlength to a reasonable accuracy. h(accurracyfunction(GPS.x,GPS.y))

b)
Use the get message of the DHT.
key = h(accuracyfunction(GPS.x,GPS.y));
Spot s = DHT.get(key)

c)
It is necessary to perform get request to all coordinates in the given proximity of the GPS coordinate.
➔ DHT.get for coordinate, for coordinate (x + 1 tick), for coordinate (x - 1 tick), for coordinate (y + 1 tick), for coordinate (y - 1 tick), for coordinate (x + 1 tick, y + 1 tick), … and many more coordinates that are within the given distance of the coordinate. ➔ Problem: depending on the resolution a large number of requests have to be made (even for low resolution, this may be 20 or even 100s of requests)

Options:
- exptected: basically adapt the resolution to a reasonably low value that is a good compromise, the requesting node can then filter the spots that are too far
- *an alternative that is not DHT-compatible as it would involve changes in the DHT implementation(!):* nodes not only reply with the requested coordinate, but with others they also have that are within the given proximity

Task 2 Key-based-Routing-API

Please, briefly describe your idea first, before you write pseudocode. Use a pseudocode that avoids unnecessary details. Assume that you have a structure Peer-to-Peer system that implements the Key-based Routing-API. All messages are processed recursively (e.g. no iterative lookup):

a) *Ping and Pong:* a node A pings (sends "Ping") a node B via the KBR network by sending message to its ID. Node B replies to the ID of node A with a "Pong". Give the code for the send and receive operation.

b) *Counter:* implement a counter that counts the number of hops (in the overlay) that the ping message takes.

Solution:

a)

```
void ping(Key x){
        route(x,"ping "+myKey,null);
        }
In  the event routine
void deliver(Key x, Message m)
        {
        if (m is a ping)
                {
                route(m.getID(),"pong"+myKey,null);
                }
        if (m is a pong)
                {
                deliverPong(x,m); // notify application or maintenance if of interest
                }
        }
```

b)

```
In the forwarding event routing
void forward(Key x, Message m, NodeHandle nextHopNode)
        {
        if  (m is ping)
                {
                messagecounter=0
                if (m.containsCounter())
                        messagecounter= extractCounterFromMessage(m);
                messagecounter ++;
                m = m.withoutCounter() | " counter=" | messagecounter;
                }
        }
```

Task 3 Consistent Hashing – Distribution of Interval sizes

In this task we want to compute the distribution of the interval size in systems on the basis of Consistent Hashing like Chord. Let us assume the ID space to be real-valued in the interval [0,1). Without loss of generality we can put our node on the position 0 in the ID space.

a) Nodes are positioned randomly on the basis of uniform random numbers. What the cumulative distribution function (CDF) L of the corresponding uniform distribution.
b) Now calculate the CDF for the minimum of n-1 independent experiments with the distribution from a). Hint: The CDF for the minimum von random variables with CDFs L_1, L_2, L_3, … is given by the forumla $L\_\min = 1 - \prod_i (1 - L_i)$.
c) Now differentiate L_min to get the probability density function l_min.
d) Plot the probability density function l_min.

Solution:

a)

Uniform distribution in interval [0,1).

U(x) = x für 0<=x<=1

b)

Apply formula for the minimum

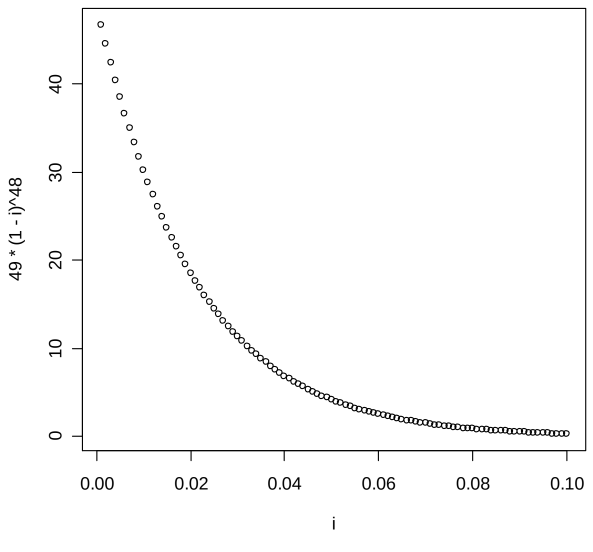
$L(x) = 1 - (1 - U(x))^{n-1} = 1 - (1 - x)^{n-1}$ für 0 <= x <= 1

c)

Differentiate L(x)

$$l(x) = \frac{dL}{dx} = -(1-x)^{n-2}(n-1)(-1) = (n-1)(1-x)^{n-2}$$

d) Plot for n=50



Task 4 A flexible Chord

The finger table entries in the classic Chord algorithm always point to the first node in the corresponding finger interval. This does not allow the freedom to select a finger among multiple peers. Yet, there are proposals to allow Chord to link to any node in the interval of the finger. If you remember the proof for the complexity of the lookup of O(logn) in Chord, we needed that the distance is halved per step.

Show that despite of that change, Chord still achieves O(logn) hops with high probability.

Solution:

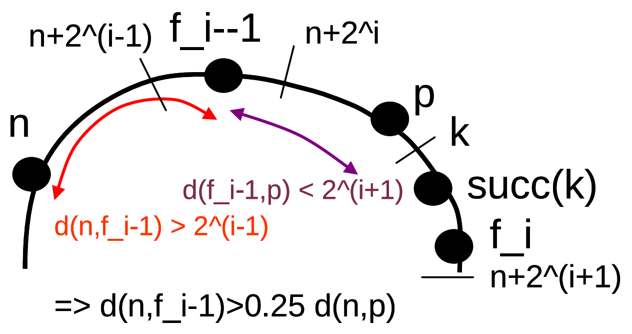
Nodes position themselves randomly. The finger table follows the strategy above. The ID is arbirtrary.

The basic difference to Chord is not that not necessarily the first node in the interval i is taken. This means that you mean search for an ID with a responsible node in the interval that is smaller than the node in your finger table for the interval. This is new compard to the old case.

So, we may only utilize the link to the preceeding interval i-1. This may also happen in classic Chord when the first node is the successor(ID) and responsible for it. In that case, its predecessor is not in this interval. In Chord we always need to find the predecessor first.

Assumption: n is large and relevant intervals contain nodes.

d(n,f_i-1) > 2^(i-1)
d(f_i-1,p) < 2^i+2^(i-1)
=> 1/3 d(f_i-1,p) = 2^(i-1)
=> d(n,f_i-1) > 1/3 d(f_i-1,p) = 1/3 (d(n,p)-d(n,f_i-1)) da d(n,p) = d(n,f_i-1) + d(f_i-1,p)
=> 4/3 d(n,f_i-1) > 1/3 d(n,p)
=> d(n,f_i-1) > ¼ d(n,p)
=> per step we progress 0.25 of the way, so up to 0.75 remain.
=> in 3 steps with reach the case when only 0.42 of the way remain and 0.58 were progressed.
=> in O(1) steps we reach the case of progressing more than 50 % of the way which we needed in the proof for the classic Chord.



Task 5 Distance and links

The more links each node in a DHT has, the shorter the distance. Yet, one can gain more or gain less depending on how good a strategy is.

Assume first, that a node sits on a ring-like ID space which we simplify to the interval [0,1). Each node links to its successor and predecessor. Each node has 100 long distance links. As long-distance links each node links to nodes i=1..100 in the distance i/100.

a) Does this approach achieve logarithmic distance?

b) What happens if you lose contact to your successor?

Assume now that you apply a different strategy and add links in distances ¼, ½, ¾, 13/16, 14/16, 15/16, 61/64, 62/64, 63/64, ... The basic idea is to divide the first interval and then the most distant intervals into quarters.

c) How does this approach affect the distance? (Hint: What reduction is achieved within one hop.)

Solution:

a)

No, the long distance links only help to approach the target to a distance of 1/100. This means O(n/100) = O(n) nodes remain in the last interval.

b)

Since all the long distance links target links further away, min 1/100 distance, they do not necessarily help to know the successor of the former successor. As a consequence, the search for the new successor is not easy and may not succeed in time.

c)

With the argument from a) long distance links do not help you to approach the target closer than ¼. Therefore, the situation is even more extreme and again O(n).

However, if you route in backward direction, then you improve your performance over e.g. Chord as your quarter your distance on each step. O(log4(n))