Peer-to-Peer Systems
Exercise 3
Published: 19.11.2014
Presentation date: 27.11.2014

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems – Exercise Winter Term 2014/2015

General Remarks

Welcome to the exercise for the lecture Peer-to-Peer Systems.

Please follow the general remarks regarding the organization of the exercise.

- The lecture's website is to be found here: `http://tsn.hhu.de/teaching/lectures/2014ws/p2p.html`
- For further inquiries, please contact the lecturer under the following email address: `graffi@cs.uni-duesseldorf.de`

Problem 3.1 - FreePastry

In the following we will have a closer look on one of the most commonly used DHTs in research called FreePastry. In order to answer the following questions, please read the short paper about FreePastry which can be obtained at `http://www.freepastry.org/PAST/overview.pdf`.

a) Routing table

How does the routing table of a FreePastry node look like? How large is the routing table of a FreePastry node?

Solution:

A node's routing table is organized into $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The $2^b - 1$ entries at row n of the routing table each refer to a node whose nodeId shares the present node's nodeId in the first n digits, but whose $n + 1$th digit has one of the $2^b - 1$ possible values other than the $n + 1$ digit in the present node's id.

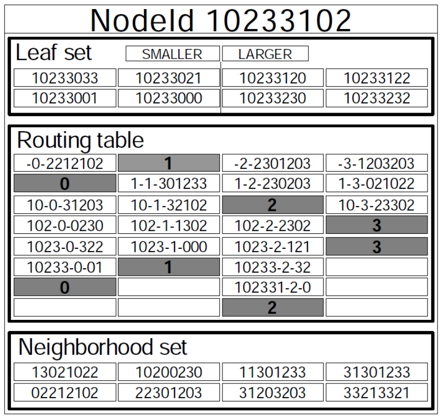
b) Routing

How is routing done in FreePastry? Please describe briefly the routing scheme.

Solution:

In each routing step, the current node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current nodeId. If no such node is found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the current nodeId.

c) ID ranges



NodeId 10233102

Leaf set

	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Abbildung 1: Example of a routing table for node 10233102.

Have a look at figure 1. For which ID range is the node 10233102 in FreePastry responsible? In the case that two nodes have the same distance to an ID, the node with the smaller ID shall be responsible.

Solution:

A node is responsible for all keys in the ID space that are closer to that node according to the distance metric then any other given node. Node 10233102 has as closest neighbors in the leafset the nodes 10233033 and 10233120. In order to calculate the arithmetical mean between 10233102 and 10233033, 10233120 respectively, these ID's are converted to decimal numbers.

$$10233102_4 = 19410_{10}$$

$$10233033_4 = 19407_{10}$$

$$10233120_4 = 19416_{10}$$
$$\frac{1}{2} * (19407 + 19410) = 19408.5$$
$$\frac{1}{2} * (19410 + 19416) = 19413$$

Thus the ID range for which 10233102 is resonsible for is from 10233101($= 19409_{10}$) to 10233111($= 19413_{10}$).

d) FreePastry and Chord

Now compare FreePastry with Chord. What are the main similarities and differences between these two DHTs?

Solution:

Similarities:

- Both DHTs use a ring structure for organizing the ID space
- Both DHTs route a given message to the node which is responsible for the ID of the message within $O(log(N))$ routing steps.

Differences:

- In contrast to Chord, which maintains only $log_2(N)$ routing entries FreePastry maintains $\lceil log_{2^b} N \rceil * 2^b - 1 + l$ entries in the routing table
- FreePastry uses prefixrouting whereas Chord routes messages according to a distance function.
- FreePastry uses a symetric distance function, whereas Chord routing is based on an asymmetric distance metric.

e) Robustness

Which of the both DHTs (Chord or Freepastry) do you consider to be more robust against churn? Please explain your answer.

Solution:

Due to its bigger routing table Freepastry is much more robust against churn then Chord. There exist several papers which showed that the basic Chord approach is not robust in case of failing nodes. In contrast to that FreePastry shows a much more robust behavior in case of failing nodes (remains stable with churn rates of up to 70%).

Problem 3.2 - Kademlia

Kademlia is one of the most used p2p overlays, let us have a look at it.

a) XOR metric

Prove that the XOR metric of Kademlia fulfills the triangle inequality, i.e. that following holds for all IDs x, y, z:

$$d(x, y) + d(y, z) \geq d(x, z).$$

Solution:

Generally, for binary numbers A, B following counts:

$$A = (A \wedge B) + (A \wedge \neg B),$$

where $\wedge$ is the bitwise AND and $\neg$ the bitwise NOT. $+$ is the normal summation of binary numbers. In this special equation, however, $+$ is identical to the bitwise OR ($\vee$), as one bit never occurs in the left as well as the right term in the summation. Thus, there is no carry over (Übertrag) during the summation.
Applying this observation on A and B in the equation, following follows:

$$\begin{aligned} A + B &= (A \wedge B) + (A \wedge \neg B) + (B \wedge A) + (B \wedge \neg A) \\ &= (A \wedge \neg B) + (B \wedge \neg A) + 2 \cdot (A \wedge B) \\ &= (A \oplus B) + 2 \cdot (A \wedge B) \\ &\geq A \oplus B, \end{aligned}$$

where $\oplus$ is the bitwise EXCLUSIVE-OR (XOR).
When setting $A = x \oplus y$ and $B = y \oplus z$, we obtain the desired triangle inequality:

$$\begin{aligned} (x \oplus y) + (y \oplus z) &\geq (x \oplus y) \oplus (y \oplus z) \\ &= x \oplus (y \oplus y) \oplus z \\ &= x \oplus 0 \oplus z \\ &= x \oplus z \end{aligned}$$

b) Example Kademlia network

Consider a Kademlia network with IDs ranging from 0 to $2^6 - 1$, i.e. being 6 bits long. Following peers participate in the network:

101100	110110	010101	000001	001110	011001

A) Determine the shortest unique prefixes of the IDs. I.e. note the unique initial bit combinations of each node ID.

B) Draw the ID tree from the view of peer 010101, use the prefixes only and mark the other subtrees seen from the peer.

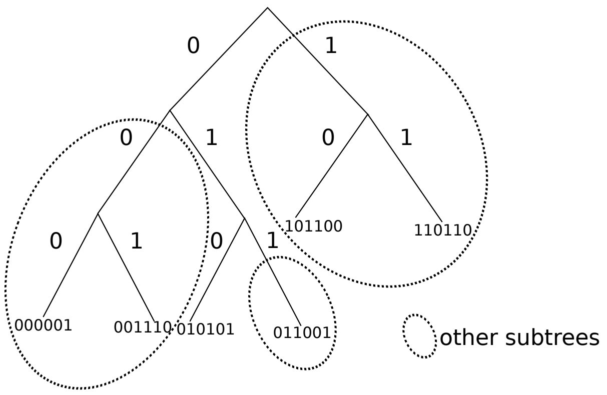
C) Assume that the network is further populated, and peer 010101 knows all other peers. In which k-buckets are the nodes listed previously?

Solution:

A) Shortest prefixes of the IDs

101100 **11**0110 **010**101 **000**001 **001**110 **011**001

B) The ID tree with the other subtrees seen from peer 010101



C) The k-buckets of node 010101: 101100 und 110110 in k-bucket number 5 (XOR-Distanz in $[2^5, 2^6)$)
000001 und 001110 in k-bucket number 4 (XOR-Distanz in $[2^4, 2^5)$)
011001 in k-bucket number 3 (XOR-Distanz in $[2^3, 2^4)$)

c) Populating the buckets

Assume a Kademlia network with ID size of 8 bits. The bucket size is $k = 4$. The k-buckets of the peer with ID 11001010 are as follows:
k-Bucket 7: 01001111, 00110011, 01010101, 00000010
k-Bucket 6: 10110011, 10111000, 10001000
k-Bucket 5: 11101010, 11101110, 11100011, 11110000
k-Bucket 4: 11010011, 11010110
k-Bucket 3: 11000111
k-Bucket 2:
k-Bucket 1:
k-Bucket 0:

A) Messages from following nodes arrive in this given order: 01101001, 10111000, 11110001, 10101010, 11100011, 11111111 How do the buckets, the orderings in the buckets and the waiting lists change?

B) Now the node detects that peer 11101110 cannot be reached anymore, what is the reaction?

C) Which addresses would the peer reply to a lookup looking for ID 11010010?

Solution:

A) Following messages arrive: 01101001, 10111000, 11110001, 10101010, 11100011, 11111111

- Check head (01001111) of bucket 7 if online (yes), push 01001111 to tail of bucket 7, 01101001 $\rightarrow$ in waiting list 7,

- 10111000 $\rightarrow$ pushed to the tail of k-bucket 6

- Check head (11101010) of bucket 5 if online (yes), push 11101010 to tail of bucket 5, 11110001 $\rightarrow$ in waiting list 5

- 10101010 $\rightarrow$ added to the tail of k-bucket 6

- 11100011 $\rightarrow$ pushed to the tail of k-bucket 5

- Check head (11101110) of bucket 5 if online (yes), push 11101110 to tail of bucket 5, 11111111 $\rightarrow$ added to the waiting list of k-bucket 5

B) When 11101110 cannot be reached anymore, 11101110 is deleted from the bucket. The node 11110001 from the waiting list is added to the tail of k-bucket 5.

C) In order to reply to a lookup looking for ID 11010010, the k closest nodes must be picked. However, k-bucket 4 is not full. Only 2 nodes exist: 11010011 and 11010110. Thus the list needs to be extended with 2 more nodes to reach $k = 4$ entries. The 2 closest entries for the searched ID are: 11000111 and 11110000. A list of the contact addresses and peer IDs of these for entries is replied.