Peer-to-Peer Systems
Exercise 5
Published: 18.12.2014
Presentation date: 15.01.2015

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 5.1 - Implementing Chord in Peerfact-Sim.KOM

In this exercise we investigate the structured peer-to-peer overlay *Chord* in detail as it is one of the best known, and most educational overlays. In order to answer the following questions, please download and install the p2p simulator PeerfactSim.KOM from `http://www.tsn.hhu.de/teaching/lectures/2014ws/p2p.html`. You also have to download and install graphviz `http://www.graphviz.org/` and gnuplot `http://www.gnuplot.info/` to visualize the results. (To install graphviz in Debian/Ubuntu you can simply type *sudo apt-get install graphviz* in the terminal).

a) Preparations

The first part of this exercise is to get familiar with *Chord* and *graphviz*, a simple software to visualize the implemented overlay.

- Install graphviz, e.g. by typing in *sudo apt-get install graphviz* in the terminal.

- Run the script *runEduChord.sh* located in the main folder of the simulator and investigate the outputs of the simulation. You should find the folder *graphs/* which contains graphviz files that contain information about the positioning of different nodes in the overlay for specific points in time. Furthermore, this folder contains a script which can be used to produce visualizations of the overlay. Figure 1 shows the different nodes participating in the overlay as well as their successor pointers (black arrows) and predecessor pointers (green arrows).
- The folder *education/paper/* contains a paper in which Chord is described in detail. Consider this paper for the next tasks below, especially Part **IV. CHORD PROTOCOL**.

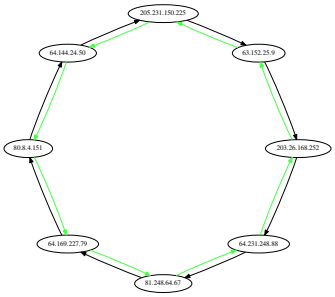

Figure 1: Chord ring visualization.

Solution:

Just do it.

b) Extension of Chord

Next, we focus on Chord's *find_successor()* method which is used to find the successor of a given identifier in the Chord ring, which is the responsible node for the given identifier. For this task focus on the implementation of Chord which can be found in the packages *org.peerfact.impl.overlay.dht.edu.chord.**. Class *ChordNode* is here the most important one since it represents a single Chord instance that can be run on a host.

- Investigate the class *ChordNode* and get familiar with its methods.
- Next, consider class *FindSuccessorOperation* from package *(...).operations*. Can you find this operation in the Chord paper? Which version of the operation has been implemented?
- Extend the existing overlay by introducing a finger table according to the description of Chord.

- Extend the existing *FindSuccessorOperation* so that finger entries are considered as possible next hop of the operation's request messages (again, consider the Chord paper).

- Investigate now the output of your simulation: describe the impact your implementation has on the *FindSuccessorOperation*, especially compare the performance of the operation before, and after your changes.

Solution:

Currently the following version of method *find_ successor(id)* is implemented:

```
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    // forward the query around the circle
    return successor.find_successor(id);
```

In this version, queries are only forwarded to each node's successor. By this means, each node which lies between a requesting node and the responsible node for a given target id has to be traversed in order to find the successor of a given id. Our goal is now to extend the method, so that queries are forwarded to the closest preceding node (out of finger table) of a given target id:

```
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);
```

We need some changes now in order to realize the use of the proposed finger table:

- Add finger table to class ChordNode: array

- On successful join event: add successor to finger table (this is needed to be sure that at least one entry exists in the finger table)

- Add *closest_ preceding_ node()* method which returns the closest preceding finger out of the finger table for a given target id.

- Change *FindSuccessorOperation* so that finger entries are considered as next hops.

- Add *fix_fingers()* method in order to refresh finger entries periodically.

(Consider code for more information).

The effect of the finger table on lookups in Chord can be seen if we consider the number of hops a *find_successor* query has to traverse until the successor of a given target overlay id is found (compare Figures 2 and 3):

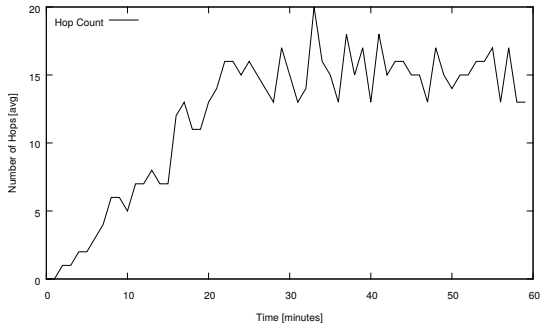

Figure 2: EduChord without finger table: number of Hops

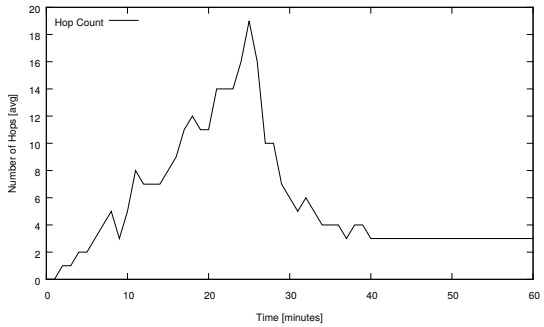

Figure 3: EduChord with finger table: number of Hops

c) Chord under Churn

Finally, we consider churn in the overlay and try to prevent the Chord ring from getting affected of leaving nodes.

- Simulate the simple Chord implementation with churn.

- Consider the Chord paper to find a solution to stabilize the Chord ring during churn: which solution is proposed in the Chord paper?
- Implement the proposed solution and evaluate your changes on the protocol.

Solution:

First, we have to change the parameter *churn* in file *config/education/educhord.xml* to *true*.

The authors of the Chord protocol present a solution to stabilize the Chord ring during churn:

- Instead of using only one successor per node, a list of r successors should be used.

- The stabilize operation should be extended so that nodes send their successor list to requesting nodes.

- Whenever a node detects its successor to be failed, it will replace its successor by another node from the successor list.

Again we extend the current implementation:

- Add successor list to ChordNode (LinkedList).

- Add successor list to GetPredecessorResponse so that nodes retrieve the successors list of their own successor upon stabilize operation

- Add methods to set or update the successor list and to remove successors which are not available any more.

- If timeouts occur during stabilize operation: remove current successor and determine new successor from given succ. list.

The effect of the successor list can be seen best if we investigate the number of failed lookups in EduChord with, and without successor list (Figures 4 and 5):

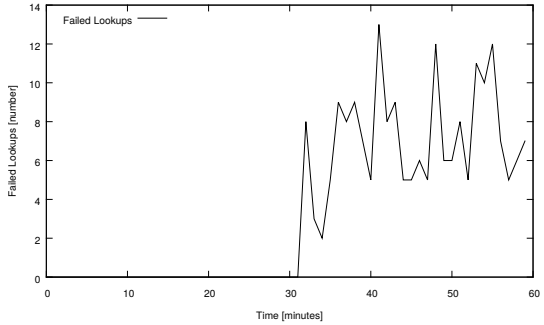

Figure 4: EduChord with successor list: number of failed lookups

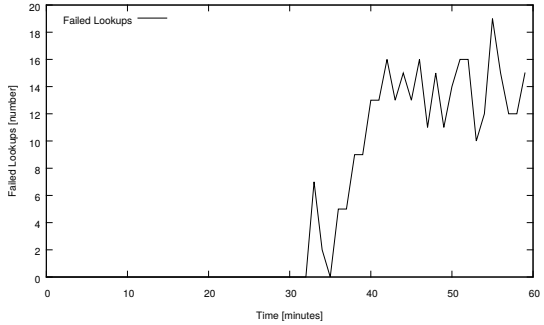

Figure 5: EduChord without successor list: number of failed lookups