Distributed Computing, Peer-to-Peer and GRIDS - ID2210

Amir H. Payberah Fatemeh Rahimian
amir@sics.se fatemeh@sics.se

April 26, 2010

Contents

1 Project Outline

The aim of this project is to implement and evaluate an algorithm for *Distributed Hash Tables (DHT)* from various aspects.

The project has two main parts:

1. In the first part of the project you will be required to implement Chord [1], which is introduced in the lecture.
2. In the second part of the project you are expected to run extensive simulations and write a report on the findings. For this part you are given the implementation of the Chord.

2 Metrics and Network Settings

This section shows the definition of metrics that should be evaluated in this assignment and also different network settings. The following metrics are important for us:

1. **Generated traffic:** In Chord, the maintenance traffic is messages send by a peer when it calls the successor stabilization and fixes the finger list. Intuitively, the more often a peer refreshes its routing table during the simulation time, the more the bandwidth consumption.
2. **Lookup hops:** To measure the latency of reaching from any peer in the system to any other peer, we measure the number of hops that a lookup message passes through to find the result.
3. **Successful and correct lookups:** The number of *successful* and *correct* lookups. A lookup is called successful, if the peer receives its reply before timeout, otherwise it is a *failed* lookup. If the successful lookup returns the correct result, it is called a correct lookup.

The mentioned metrics have to be evaluated under various setting defined by the following parameters:

1. **Network size:** Different number of nodes in the system.
2. **Churn level:** Having different levels of churn. Churn in a system refers to joins and failures of peers. Churn level is defined by (i) the ratio of the join events to the failure events, and (ii) their inter-arrival time.

3 Your Tasks

In this section you find the detail information about each task.

3.1 Task One

Your first task is to implement Chord in Kompics [2]. Section 5 contains the pseudo code of Chord (Algorithms 1 and 2). This pseudo code is presented in RPC style, while you have to implement them in message-passing model. In the message-passing model, peer p instead of calling a remote procedure of a peer q directly, should send a message to q, and q after receiving that message should process it and send another message back to p. For details of how to implement an RPC call in message-passing model, please consult the given sample code in the Lab0. Along with this document, you are given one Java package for Chord.

Here are some helpful hints to use in the implementation:

- Attached to this document is the skeleton of the Chord implementation. You can use this skeleton and fill in the methods, but feel free to implement it from scratch.
- In your defined scenarios, at the beginning only one node joins, and after a while the rest of nodes join. See the given sample scenario in the Lab0.
- After a node has set its successor, i.e, has joined the overlay, it should send a `BootstrapCompleted` to bootstrap server.
- To implement Chord, assume the peers have access to the failure detector for their successors and predecessors. The failure detector helps the peers to detect the failure of peers of interest. Note that peers can not use the failure detector for their finger list and successor list.
- The given provides the `belongTo` methods in `RingKey` class, which can be helpful in implementation of Chord. This method checks whether `id` is in the range of `from` and `to` or not.
- The given source code generates `peer.dot` as an output. You can use the following command to convert it to a PS file. The `neato` is part of the *graphviz* [3] application.

```
neato -Tps peer.dot -o peer.ps
```

How to verify your implementation: The Chord implementation should satisfy the following properties in the absence of churn: (i) the constructed ring should be correct, which means that the successor and predecessor of each peer should point to the correct peer, (ii) the finger list and the successor list should be filled with the proper peers in the overlay, and (iii) your implemented Chord should work in churn scenario.

A few local variables are defined in the provided `Peer` classes that are used to store the properties of each peer, e.g. the variable `succ` points to the successor of a peer and `pred` stores the peer predecessor. To verify these properties, the provided code contains a class called `Snapshot` that periodically generates a report about the peers status. This class checks the properties via the `succ`, `pred`,

`fingers` and `succList` variables of `Peer` and reports whether they are correct or not. You can modify the `Snapshot` class if you want to check additional properties of the constructed DHT.

3.2 Task Two

The main idea of task 2 is to analyze the behavior of Chord in different network configurations and scenarios. A typical experiment comprises of three sequential parts, (i) network creation, (ii) network stabilization and (iii) taking measurements. In the first part, the overlay is constructed by letting nodes join the system. This is done until the desired network size is achieved. In the second part, the overlay is allowed to stabilize. During this period, peers continuously perform overlay maintenance. The delay for this part should be enough so that the peers have created their routing tables. Now, the network is ready for the actual experiment. In the third part, you have to trigger events as per requirements of the experiment and gather statistics that have to be reported.

In all the experiments consider the followings:

- Assume the ID space is from 0 to $2^{13} - 1$.
- Assume the successor list stabilization and the finger list stabilization are synchronized.

In this task you are supposed to answer to the following questions:

- **Question 1:** How does the rate of stabilization influence the total generated traffic in a static network?
  (Hints: Assume there are 200 nodes in the system. Change the stabilization interval from 500 milliseconds to 2500 milliseconds, while increasing it at each step by 500 milliseconds. Draw a plot such that the X-axis shows the stabilization interval and the Y-axis shows the total generated traffic.)
- **Question 2:** How does the rate of stabilization influence the ratio of successful lookups and the ratio of correct lookups, in a dynamic network?
  (Hints: Do 100 lookups after 50 peers join the system and are stabilized. At the same time the system generates 100 operations, such that 70 operations are join and 30 operations are fail, and the time between the generation of two consecutive operations is extracted from an exponential distribution with a mean of 500 milliseconds. Change the stabilization interval from 500 milliseconds to 2500 milliseconds, while increasing it at each step by 500 milliseconds. Draw a plot such that the X-axis shows the stabilization interval and the Y-axis shows the ratio of successful lookups and the ratio of correct lookups.)
- **Question 3:** How does the rate of stabilization influence the lookup hop-count in a dynamic network?
  (Hints: Do 100 lookups after 50 peers join the system and are stabilized. At the same time the system generates 100 operations, such that 70 operations are join and 30 operations are fail, and the time between the generation of two consecutive operations is extracted from an exponential distribution with a mean of 500 milliseconds. Change the stabilization

interval from 500 milliseconds to 2500 milliseconds, while increasing it at each step by 500 milliseconds. Draw a plot such that the X-axis shows the stabilization interval and the Y-axis shows the lookup hop-count.)

- **Question 4:** How does the level of dynamism influence the ratio of successful lookups and the ratio of correct lookups?
  (Hints: Assume the successor list stabilization and the finger list stabilization are synchronized and equals 1000 milliseconds. Do 100 lookups after 50 peers join the system and are stabilized. At the same time, the system generates 100 operations, such that 70 operations are join and 30 operations are fail. Change the mean of the interval time from 200 milliseconds to 1000 milliseconds, while increasing it at each step by 200 milliseconds. Draw a plot such that the X-axis shows the time between the generation of two consecutive operations and the Y-axis shows the ratio of successful lookups and the ratio of correct lookups.)

- **Question 5:** How does the level of dynamism influence the lookup hop-count?
  Amuse the successor list stabilization and the finger list stabilization are synchronized and equals 1000 milliseconds. Do 100 lookups after 50 peers join the system and are stabilized. At the same time, the system generates 100 operations, such that 70 operations are join and 30 operations are fail. Change the mean of the interval time from 200 milliseconds to 1000 milliseconds, while increasing it at each step by 200 milliseconds. Draw a plot such that the X-axis shows the time between the generation of two consecutive operations and the Y-axis shows the lookup hop-count.)

- **Question 6:** How does the network size influence the lookup hop-count?
  (Hints: Assume the successor list stabilization and the finger list stabilization are synchronized and equals 1000 milliseconds. The number of nodes is 2^i, where i changes from 4 to 9. Do 1000 lookups after all peers join the system. Draw a plot such that the X-axis shows the number of nodes in the system and the Y-axis shows the lookup hop-count.)

4 Things To Deliver

The assignment can be done individually or in group of two. You should hand in the following materials in a zip file. Name the file according to the following convention: `LabX_FirstnameFamilyname1_FirstnameFamilyname2.zip`. This file should include:

1. The source code of the implemented Chord.
2. A report that contains the drawn plots and a brief explanation for each one. It is recommended to use Gnuplot [4] to draw the plots, but you are free to use any other applications.

5 Chord Pseudo Code

In this section you can find the pseudo code of Chord. These algorithms are shown in RPC style, whereas you should implement them in message-passing model. Algorithms 1 and 2 show the RPC style pseudo code of Chord.

Algorithm 1 Chord Pseudo Code

```
1: // create a new Chord ring.
2: n.create() {
3:   predecessor = nil;
4:   successor = n;
5: }
6: //------------------------------------------------------------
7: // join a Chord ring containing node u.
8: n.join(u) {
9:   predecessor = nil;
10:  successor = u.find_successor(n);
11: }
12: //------------------------------------------------------------
13: // called periodically.  verifies n's immediate successor, and
14: // and tells the successor about n.
15: n.stabilize() {
16:  x = successor.predecessor;
17:  if (x ∈ (n, successor))
18:    successor = x;
19:  successor.notify(n);
20: }
21: //------------------------------------------------------------
22: // n thinks it might be our predecessor.
23: n.notify(u) {
24:  if (predecessor is nil or u ∈ (predecessor, n))
25:    predecessor = u;
26: }
```

Algorithm 2 Chord Pseudo Code - Cont.

```
1: // called periodically.  refreshes finger table entries.
2: // next stores the index of the next finger to fix.
3: n.fix_fingers() {
4:  next = next + 1;
5:  if (next > m)
6:   next = 1;
7:  finger[next] = find_successor(n + 2^(next−1));
8: }
9: //-------------------------------------------------------------
10: // called periodically.  checks whether predecessor has failed.
11: n.check_predecessor() {
12:  if (predecessor has failed)
13:   predecessor = nil;
14: }
15: //-------------------------------------------------------------
16: // ask node n to find the successor of id
17: n.find_successor(id) {
18:  if (id ∈ (n, successor])
19:   return successor;
20:  else {
21:   m = closest_preceding_node(id);
22:   return m.find_successor(id);
23:  }
24: }
25: //-------------------------------------------------------------
26: // search the local table for the highest predecessor of id
27: n.closest_preceding_node(id) {
28:  for i = m downto 1 {
29:   if (finger[i] ∈ (n, id)) {
30:    return finger[i];
31:   }
32:  return n;
33: }
```

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17.32, 2003.

[2] Kompics - `http://www.kompics.se`

[3] Graphviz - `http://www.graphviz.org`

[4] Gnuplot - `http://www.gnuplot.info`