Chord implementation for Peersim

Andrea Presa
presa.andrea@gmail.com

1. Configuration file

This document explains how to write the configuration file that allows to execute the chord protocol in the Peersim network simulator. This is the content of the example file config-chord.cfg:

```
# random.seed 1234567890
simulation.endtime 10^6
simulation.logtime 10^6

simulation.experiments 1

network.size 5000

protocol.tr UniformRandomTransport
{
      mindelay 0
      maxdelay 0
}
```

These lines are standard declarations in Peersim event-driven protocols: the first one defines the seed used to generate pseudo-random numbers (if uncommented all the simulations will behave the same, useful for debugging), the second and the third define the time taken by the simulation and the logging of results, the fourth the number of experiments, the fifth the number of nodes in the network. The last commands create a transport protocol that will be necessary to send messages between nodes.

```
protocol.my ChordProtocol
{
      transport tr
}
```

Here is where we define the core protocol that simulates the behaviour of a chord application in every node. Messages will be exchanged by the transport protocol defined above.

```
control.traffic TrafficGenerator
{
      protocol my
      step 100
}
```

We need now to istantiate a control subclass that will generate lookup messages (source and destination are randomly choosen) every predefined step of time.

```
init.create CreateNw
{
      protocol my
      idLength 128
      succListSize 12
}
```

Prior to the simulation we have to create the perfect chord network. **`CreateNw`** class will initialize every node by generating the Chord identifiers (randomly chose within the number of bytes specified in **`idLength`**), linking successors (the **`succListSize`** is the size of the successors list) and predecessors, creating the corresponding finger tables (with **`idLength`** number of entries) and other parameters.

```
control.observer MessageCounterObserver
{
      protocol my
      step 90000
}
```

The last part the basic execution of the protocol needs is the observer, that computes values like the average number of hops in lookup processing and so on. These results are printed at every specified step time.

```
control.dnet DynamicNetwork
{
      add 20
      add -25
      minsize 3000
      maxsize 7000
      step 100000
      init.0 ChordInitializer
      {
            protocol my
      }
}
```

To test the protocol in churn condition (nodes are removed and new ones are added) we use the peersim-provided control, Dynamic Network, that can be customized by declaring the number of nodes to add (or remove), the bounds on the network dimension and the step time.

In order to allow our Chord protocol to accept new nodes inside the network, every node must be initialized by the init lines. A node that is already part of the chord ring is randomly selected and will help the new one in finding successors and finger table entries.

The execution of the protocol in perfect condition (no churn, reliable transport) has lead to the results that were predictable: the maximum number of hops that a lookup message has to pass before reaching the correct destination is bounded by the logarithm in base 2 of the number of nodes within the network. Of course this number must be approximated to an integer (so 12,29 becomes a limit of 13 hops). The average number of hops is also predictably low (bear in mind that lookup from a node to itself can be generated). These parameters were fixed: 128 of ID length and finger table entries, 12 as successor list size, 10^6 as time of simulation with about 9000 messages produced for every experiment.

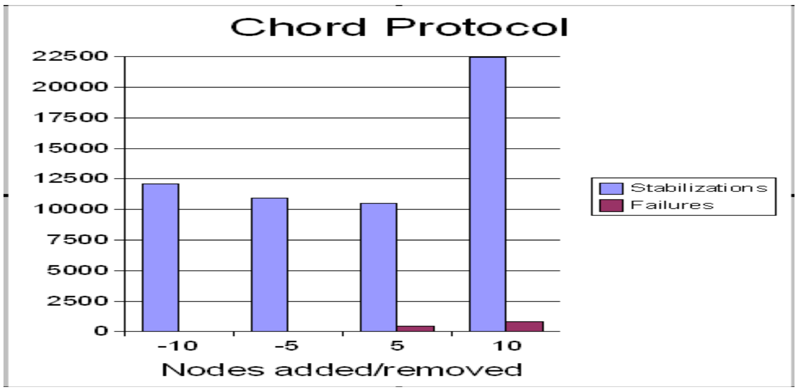
2. Run the code

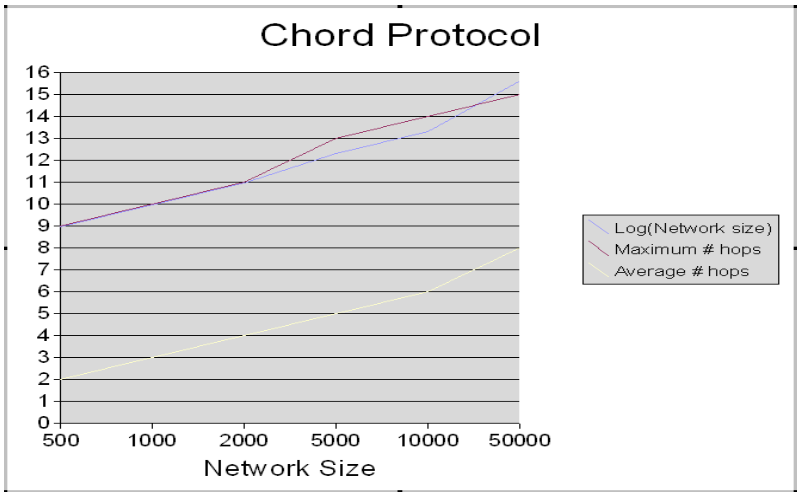
To execute the configuration file, run the following command:

```
java -cp peersim-1.0.3.jar:djep-1.0.0.jar:jep-2.3.0.jar peersim.Simulator
config-chord.txt
```

3. Results

To correctly judge the behavior in spite of dynamism of the network (node failures, node insertions) we must consider more parameters, in particular the number of calls to the stabilization method and the failure in reaching the destination (in which case the message is thrown away). The fixed parameters were: 128 as idLength, 12 as successor list size, 5000 as the the size of the initial network, 3000 and 7000 as minimum and maximum bounds for the network size, execution time as the experiment before and the step of the dynamic control set at 10 seconds. As you may notice, the removing process is not very critical for the ring: the number of failures is bounded (so low it doesn't appear on the graph) and the maximum number of hops is more than acceptable (15 instead of 13). Bear in mind that the stabilization is the number of times that the nodes call the method stabilize(), which it's executed every time successor lists or finger tables have to be updated (can happen many times per interaction), so it is predictably high considering that the network is reduced or enhanced of a fifth of its original dimension during the simulation. When the nodes are joining we notice that the number of failures increases: this is due to the fact that a failure in the remove-only condition has to be declared when the chord id to which deliver the message is between a node and its successor, so that we can assume that the message was originated for a node that has been lost in the meanwhile. The problem is that to deal with a consistent number of joins the initializer cannot determine the absolutely correct successor of a node (the performance would be too slow) but only a node that is not too far away, and then at every interaction the new node will adjust its successor pointer to a closer node in the ring. Unfortunately in the process many messages can come and if they are directed to a node between the new one and its successor they will be dumped. This explains why the number of failures increases with the number of added nodes. In a longer simulations however, the successor lists will be adjusted until no more failure appears. The average and the maximum number of hops are the same as the ones in absence of churn.


Chord Protocol
Nodes added/removed
Stabilizations
Failures


Chord Protocol
Network Size
Log(Network size)
Maximum # hops
Average # hops