Report 5: Chordy, a distributed hash table

Alexandre Tamborrino

October 3, 2012

1 Introduction

In this seminar we have implemented a peer-to-peer distributed hash table (DHT) following the Chord protocol. DHT are important because they allow horizontal scalability of in-memory stored data and fault-tolerance.

2 Main problems and solutions

First, we have only implemented the DHT's *ring* without the ability of storing and retrieving data.

Considering the stabilize operation, each node sends a *request* message to its successor telling it that it wants to know its predecessor. One specific case is when there is only one node in the ring : at the beginning, the successor of the single node is itself and it has no predecessor. Then, it will begin the *stabilizing* procedure: it sends a *request* message to its successor (itself), and then it answers to itself that it has no predecessor, therefore it sends to itself to take itself as a predecessor (*notifying*). Thus, when there is only one node in the system, it has itself as predecessor and successor (in the code, Xkey is equal to Skey).

Moreover, the frequency of this stabilizing operation is every 100 ms for each node. The pros of a frequent stabilizing procedure are that the system sees very rapidly if another node has entered the system (or if a node has crashed), and then it is able to very rapidly re-build a correct ring. The cons are obviously that it induces a network traffic overload. On the other hand, if there is no stabilizing procedure, the ring can not be re-build after additions or deletions of nodes, so it is important to find the right frequency for this operation.

Secondly, we have added a storage mechanism to the DHT implementation. The tricky part is to correctly split the data store when a new node is added to the system. More specifically, if the node N has a new predecessor P, N should keep the keys-values between (Pkey,Nkey] and give the rest to P. To do this, I have changed the arity of the function split/2 to split/3 as follows: *split(Nkey, Pkey, Store)*. Here is the code:

```
split(LocalKey, Pkey, Storage) ->
  {Keep, Give} = lists:foldl(fun({Key,Value},{AccSplit1,
      AccSplit2}) ->
                      case key:between(Key, Pkey, LocalKey) of
                          true ->
                              %% keep for local node
                              {[{Key,Value}|AccSplit1],AccSplit2};
                          false ->
                              {AccSplit1,[{Key,Value}|AccSplit2]}
                      end
                    end, {[],[]}, Storage).
```

3 Evaluation

In order to test the performance of the DHT, I have implemented a test module that allows to launch either 1 or 4 nodes (with randomly generated keys) and to launch either 1 or 4 test machines. If there is only one test machine, this machine adds 4000 randomly generated keys, and then do a lookup for all of these 4000 keys. If there are 4 test machines, each of these adds 1000 keys and then do a lookup for all of these 1000 keys (all the test machines run concurrently). As my computer has 8 cores, we could consider that all these nodes and test machines are running in parallel as if they were distributed. However, there is no network latency here, so these tests may differ from real distributed tests. We will measure the lookup time for each configuration. In the case of multiple test machines, the lookup times of all machines are added together. The results are given Table 1. The lookup time is an average and is given in second (not very accurate, but we are only interested here by the difference between the configurations).

	1 node	4 nodes
1 test machine	0.003	0.003
4 test machines on the same node	0.08	0.09
4 test machines each on a different node	X	0.002

Table 1: Average lookup time (seconds)

Thus, we see that the distributed nature of the DHT really improves the performance when the clients (test machines) act concurrently on the system but not all on the same node.

4 Conclusions

During this seminar I have understood the basics of the P2P Chord DHT and I have made some tests to highlight how the distributed nature of this DHT affects performance.