Distributed Systems
Report 5: Chordy

Yerrapragada Akhil

October 9, 2019

1 Introduction

This seminar is intended to explain the process of creating a distributed hash table using Erlang and describe the functionality of it. The key technical aspects implemented in chordy are: node1, node2, key, storage and test.

In node1, we will implement a ring structure to which we add nodes. We stabilize the incoming node to the ring in such a way that it checks if it exactly fits in between a predecessor and a successor. If it doesn't, proceed to next and so on. Post we have a stable ring in place, In node 2, we use a storage to which we add some random generated key value pairs. Here, we use add and lookup functionalities to check if the nodes are in place, also, we use the handle functionality to handle a scenario where a new node enters the ring post storage is implemented. We also have storage module, which is used to create a storage, add elements to the storage and look into them when necessary. The key module contains the implementations required to actively create some key randomly when necessary. Both Key and Storage modules are used in node2 implementation.

2 Main problems and solutions

When adding another node to the ring and probing it, we can observe the increase in time. The more nodes we add to the ring, it is obvious that the time will increase. Also, it doesn't matter that the newly created nodes access the same node because we have a handling functionality available that will make the ring stable. When we add more nodes to the ring post implementing the storage, we need to make sure that the storage is equally spit between the new node and the existing nodes. Therefore, we implement a handover functionality so that the storage is levelled.

3 Evaluation

1) Initial implementation of the ring without storage:

```
Node:501490715 forwarding probe to <0.101.0>  St
probe
Node:723040206 forwarding probe to <0.103.0>  St
12> Node:945816365 forwarding probe to <0.107.0>
12> Node:311326755 forwarding probe to <0.99.0>
12> Node:<0.99.0> Removing probe after 1micros?.
 Nodes visited: 5
```

2) Ring with storage implemented:

```
Node:501490715 forwarding probe to <0.101.0>  Storage: []
probe
Node:723040206 forwarding probe to <0.103.0>  Storage: []
12> Node:945816365 forwarding probe to <0.107.0>  Storage: []
12> Node:311326755 forwarding probe to <0.99.0>  Storage: []
```

3) Ring with keys added to a node.

```
13> N1!status.
NodeID= 443584618   Pred={311326755,<0.107.0>} Succ={501490715,<0.105.0>}  Storage: [{334453409,
gurka},{417500367,gurka},{333324069,gurka},{316985313,gurka},{333638099,gurka},{315478554,gurka}
,{413537478,gurka},{393299010,gurka},{400771145,gurka},{388887546,gurka},{328017639,gurka},{3302
49246,gurka},{400961457,gurka},{421398761,gurka}]
```

4) Changes observed in storage when new node is added to the ring.

```
41> Up and running!41> N1!status.
NodeID= 443584618   Pred={398257748,<0.153.0>} Succ={501490715,<0.105.0>}  Storage: [{417500367,
gurka},{413537478,gurka},{400771145,gurka},{400961457,gurka},{421398761,gurka}]
```

5) Status of new node with it's updated storage.

```
42> N20!status.
NodeID= 398257748   Pred={316730073,<0.140.0>} Succ={443584618,<0.99.0>}  Storage: [{334453409,g
urka},{333324069,gurka},{316985313,gurka},{333638099,gurka},{393299010,gurka},{388887546,gurka},
{328017639,gurka},{330249246,gurka}]
```

4 Conclusions

Overall, a good amount of learning was made on load balancing and consistent hashing concepts. Certain new list API's were learnt, and significant amount of progress was made in learning erlang.