CMSC 23310 Final Project: Distributed Hash Table

Josh Feingold Henry Nicholas Riordan Jake Whitaker

Spring 2014

1 Project Overview

For this project we have decided on implementing a distributed hash table for our keystore. We will focus on producing a key-value store that fits in the BASE model.[2] Our design of the distributed hash table comes from Chord[4], Pastry [3], and Dynamo [1]. Overall our design will follow that of Dynamo the closest as we liked some of the simple design choices. Our DHT like the others is based on a circular keyspace on which we place our nodes. These nodes will be in control of all keys between it and its predecessor.

For fault tolerance we will use replicas in order to be able to remain available and consistent after a fail stop. We will use 2 replicas of all keys in addition to the main keystore. In order to detect failures we use a heartbeat that is sent every [**INSERT HEARTBEAT TIME HERE**] and if we do not receive a heartbeat in [**INSERT HEARTBEAT/ NODE TIMEOUT HERE**] then we remove the node from our routing table and adjust our replicas as needed. If we get a heartbeat from a node we do not have in our routing table, we add them and merge their keystore with our own, also adjusting what keys we own and replicate.

2 Implementation

Our implementation follows closely to that of Dynamo. We have a complete routing table and 2 replicas of the key-value pairs.

2.1 Get

Our get function is very simple. When a get request comes into a node the following occurs. The node checks to see if the key belongs to itself. If it does it will retrieve the key and send the get response message to the broker. If it does not own the key, the node will forward the get request to the correct node, using the complete routing table. To ensure that the get message was received by the second node, that node will send the get response back to the first node who then responds to the broker.

2.2 Set

The set function first checks if the key being set is in our section of the keyspace. If not we forward the message to the correct node, who after completing this function verifies completion by sending a message to the original node. If the key is in our section of the keyspace we set the value in our keystore and then send two replication messages, one to our successor and one to our successor's successor. This creates 2 replicas in case the first node fails. After completion we either send a setResponse to the broker or the node who forwarded the request.

2.3 Heartbeats

2.4 Replication

A node only replicates keys it is sent on a replicate message. It stores these keys in the same keystore as its main keys so that if its predecessor fails, it will automatically be able to handle the requests for that keyspace. A node will only delete keys from its keystore if it sees a new node enter that reduces the keyspace that this node would replicate. This check occurs when the routing table is changed due to a heartbeat from an unknown node.

2.5 Merging

3 Example Scripts and Discussion

Listing 1: Example Script 1

```
start test
send {"destination": ["test"], "foo": "bar", "type": "foo"}
get test foo
send {"destination": ["test"], "bar": "baz", "type": "foo"}
set foo 42
get foo
```

4 Conclusion

References

[1] Giuseppe DeCandia et al. "Dynamo: Amazon's Highly Available Key-value Store". In: *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles.* SOSP '07. Stevenson, Washington, USA: ACM, 2007, pp. 205–220. ISBN: 978-1-59593-591-5. DOI: 10.1145/1294261.1294281. URL: http://doi.acm.org/10.1145/1294261.1294281.

[2] Armando Fox et al. "Cluster-based Scalable Network Services". In: *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles.* SOSP '97. Saint Malo, France: ACM, 1997, pp. 78–91. ISBN: 0-89791-916-5. DOI: 10.1145/268998.266662. URL: http://doi.acm.org/10.1145/268998.266662.

[3] Antony I. T. Rowstron and Peter Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg.* Middleware '01. London, UK, UK: Springer-Verlag, 2001, pp. 329–350. ISBN: 3-540-42800-3. URL: http://dl.acm.org/citation.cfm?id=646591.697650.

[4] Ion Stoica et al. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications". In: *IEEE/ACM Trans. Netw.* 11.1 (Feb. 2003), pp. 17–32. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.808407. URL: http://dx.doi.org/10.1109/TNET.2002.808407.