Report 5: Chordy

Bernardo González Riede

October 12, 2017

1 Introduction

Distributed Hast tables are about retrieving a value associated with a given key. The purpose is to distribute load over several servers, which can be geographically far away from each other. It's suppose to handle high churn rates while still providing the service. The tolerable churn rate depends on the implementation, mostly regarding the amount of replications. This determines the amount of *sequential* nodes which can fail simultaneously with the DHT continuing to work. The network can be handled either *structured* or *unstructured*. Without going much into the details, *structured* offers simpler lookups with the cost of having to maintain the network wich introduces overhead. *Unstructured* on the other hand doesn't have an overhead, but the lookups are more complex, thus more expensive.

DHT only handles inserts and lookups of values associated with a key. The application which uses the DHT manages the data.

2 Main problems and solutions

2.1 Questions in the report

What does *From* possibly being greater than *To* mean?
DHT is based on a ring architecture. To close it, the *last* and *first* node have to be connected. Therefore *From* may be greater than *To*.

What are the pros and cons of a more frequent stabilizing procedure? What is delayed if we don't do stabilizing that often?
The acceptance of new nodes as Successor. A new node joining gets established as *predecessor*, but unless *stabilize/1* is executed, this nodes doesn't get used as a *successor*.

What would happen if we didn't schedule the stabilize procedure? Would things still work?
It could work to an arbitrary degree. Not executing *stabilize/1* means no adding of joined nodes as *successor*. The exception is when a node is added, since it will acquire a successor in the process. This means that the first

node will point to itself and the line of predecessor will be correct. Therefore the "ring" isn't closed.

Which part should be kept and which part should be handed over?
All the pairs with keys being less than or equal to the new predecessors ID.

What will happen if a node is falsely detected of being dead?
Assuming it's predecessor and successor detect it, they will close the ring between them. Should the presumably dead node return, it will eventually, thanks to the schedule*stabilize* integrate itself in the ring.

2.2 Problems

Throughout the coding several difficulties were encountered. Most of them were because of syntax issues. Therefore it's unnecessary to include them in this report. Logic issues which appeared were because of differences in the proposed code and the written one. e.g. passing the *Next* node as the las argument rather than some intermediary.

3 Performance

For testing purposes, a *client.erl* file was created which works differently than *test.erl*. The main difference is that it doesn't wait for a confirmation. If a collection of values is inserted, it sends all request and processes each confirmation which includes the inserted key. This results in a possible mixing of inserting and acknowledging of keys. Additionally the keys are inserted in a sequential run, not random. This gives less room for randomness to influence the behavior. Although not measured, assumptions can be made about the behavior of the system while several nodes being present.

- When dealing with larger hash tables, the local lookup function is probably the most cpu intensive task. Since the nodes work in one process, i.e. the same process is in charge of routing messages or looking up a function, a node won't forward a lookup request while it's locally looking up a key.
- If, on the other hand, the network latency is the major impact, then the increased amount of nodes won't decrease the lookup rate proportionally.

4 Conclusion

Distributed Hash Tables don't seem to focus on reducing response time, rather than on distributing workloads. This isn't to say that it doesn't reduce average response time, which it does, but it's more of an desirable side effect.