Distributed key-value storage system

Vetle Hofsøy-Woie[1] & Ragnar Helgaas[2]
vho023@uit.no rhe066@uit.no

[12]Department of Computer Science, The Arctic University of Norway

September 27, 2021

I. Introduction

Decentralized distributed systems have several benefits over other systems especially removing the risk of a single point of failure. However, one of the main drawbacks of creating a distributed data lookup system is the time needed to determine which node stores the data. This report describes and tests an implementation of the Chord protocol[1] and discusses how the implementation can be improved.

II. Background

Chord is a peer-to-peer lookup protocol initially proposed by Stoica *et al.*[1]. The Chord protocol assigns each node and key an identifier of m-bits, ordered as a circle from 0 to $2^m - 1$. A key belongs to a specific node if the key's identifier lies between the node's identifier and the next clockwise node identifier on the circle. This scheme results in each node needing knowledge about its successor and predecessor on the identifier circle and not all other nodes on the network[1].

III. Design

This report describes an implementation of the Chord protocol. The implementation uses a RESTful API to communicate between the user and the nodes and intercommunication between the nodes. During the service startup, each node calculates the identifier for every node in the system; it then places itself accordingly on the identifier circle and disregards all other nodes except its successor and predecessor. When a lookup occurs on a node, the node will check whether the lookup key resides between its identifier and its predecessor's identifier. If it does, the node has the data. If not, it forwards the requests to its successor. For this scheme to work, it is essential that the hashing algorithm used is sufficiently random and that the m-bit identifier is large enough that the likelihood of two nodes getting the same identifier is negligible. In this design, the implementation calculates the identifiers using the SHA-1 hash function and a 35-bit identifier.

IV. Implementation

The implementation discussed in this report was written in the Python programming language. Communication of data between nodes is done using a RESTful API. The communication between the nodes is done using the Python request library. This implementation does not utilize threading to handle requests, and the requests are made using the Python request library. Each node calculates the identifier for every node in the system. Using equation 1, as mentioned in section III the node disregards all other nodes other than its own neighbors after this computation.

$$Id = Hash(IP) \bmod 2^m \tag{1}$$

The key provided for either storage or search is referred to as an external key for this implementation. Each external key is associated with an id in the same way as the IP addresses of the nodes. This id is referred to as a local key. When a user initiates a lookup after an external key, it is matched with the hashed local key and returned. In addition, nodes within the distributed system store values in dictionaries using the local keys.

V. Evaluation

Because of the linked list type indexing of nodes, a lookup should have a worst-case complexity of $O(N)$ and an average complexity of $O(\frac{N}{2})$, where N is the number of nodes. Based on this data, the average lookup time is expected to increase linearly as the number of nodes increases. The following experiment tests the system to try to ascertain this hypothesis.

A. Experiment

1. Several systems were created with an increasing amount of nodes, starting at one and ending at 16 nodes. For each of the systems, the test measured the throughput of PUT's and GET's per second. The test ran one hundred PUT, GET pairs one hundred times to get the average wait time when communicating with the system.The result of this experiment, and its standard error, can be seen in figure 1

VI. Discussion

Figure 1 supports the hypothesis as mentioned above, as it is clear that the response time of the system grows linearly as the number of nodes in the system increases. Because of this linear growth in response time, the actual throughput of the system quickly decreases as the number of nodes increases. The evolution of throughput in relation to number of nodes is illustrated in figure 2.

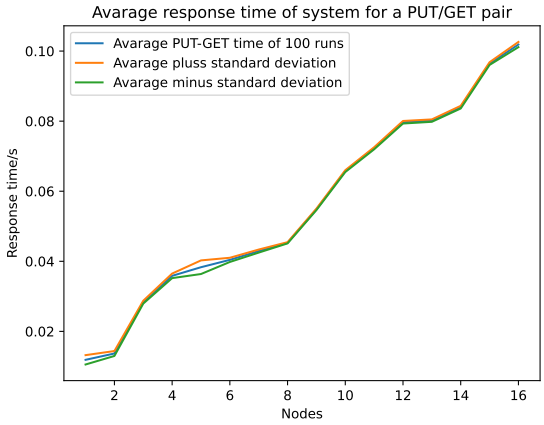

Fig. 1. Response time of system based on number of nodes

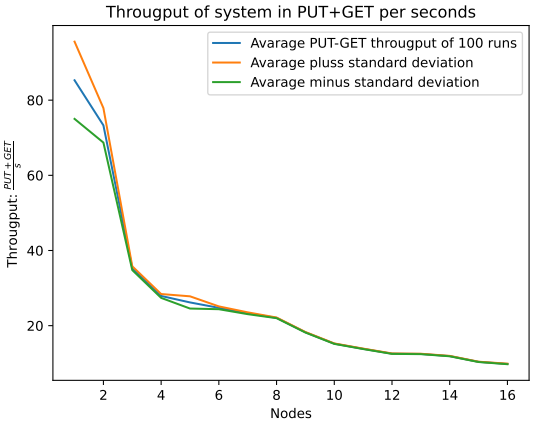

Fig. 2. Throughput of system based on number of nodes

Stoica *et al.* proposes another lookup scheme in their paper on the subject, which increases the number of nodes each node needs to communicate with but decreases the average lookup time to $O(\frac{1}{2}\log N)$[1]. This scheme is more scalable in the number of nodes, but each node must contain information about $O(\log N)$ other nodes instead of only two nodes for the linear scheme.

A. Communication protocol

The implementation discussed in this report uses a RESTful API over an HTTP connection as the intercommunication protocol. The HTTP protocol is probably unnecessary as some simple TCP or UDP protocols also could be used. By implementing a RESTful API towards the end-user and some more specialized protocol between the nodes in the network, some increase in throughput could be expected.

B. Future work

The current implementation does not support a dynamic amount of nodes, where the main benefit would be strengthening the overall robustness of the system. For example, if a node notified the system that it was to leave, a protocol concerning the distribution of the node's information ensures the future prosperity of the system. Dynamic nodes would also improve hardiness in case of a node going dark by having backup nodes within the system that can take its place. The backup of data could be done by having each node back up its successor's data or distributing the backup data across the whole system. Further studies could research whether this backup would be worth the extra complexity the system would need. To increase the system's robustness, even more, ACID transactions could be implemented on the system. However, the complexity needed to implement this might be so high that it would be better to utilize some distributed database instead.

VII. Conclusion

This report described and tested the throughput of an implementation based on the Chord protocol by Stoica *et al.* Thoughts on how the implementation could be improved has also been put forth. At last some discussions around the writers thoughts on future work was presented.

References

[1] Ion Stocia; Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, "CHORD: A Scalable Peer-to-peer Lookup Service for Internet Applications," *IEEE/ACM Trans. Netw., vol. 11, no. 1, pp. 17–32*, pp. 1–12, Feb 2003.