A Distributed Key-Value Database using Chord

Amit Barve
North Carolina State University
aabarve@ncsu.edu

Shaurya Garg
North Carolina State University
sgarg7@ncsu.edu

ABSTRACT

Key-value based databases are used extensively by many web applications and services. Many of such applications like Facebook, GitHub are so large that it is impractical to use centralized databases for persistent storage. Distributed database seem to be a good choice for such applications but having a distributed database has its own challenges. Data consistency, High Availability, Temporary/Permanent failures of nodes are some of the critical problems that need to be solved before such databases can be used in production. Many approaches (such as Hadoop Distributed File System) suggest having a central manager node which will be responsible for the complete cluster and will resolve any issues. However, such a database is prone to single point of failures and can not be considered as a fully distributed database. We will implement a completely distributed Peer-to-Peer key value database based on Chord's ring topology protocol. Chord protocol achieves complete decentralization with the help of a Distributed Hash Table. It supports joining of a new node, can maintain multiple replicas and handles node failures/departures efficiently. A consistent and simple to use API is provided for utilizing this database in any application.

Keywords

Distributed systems, Key-value Database, Peer-to-Peer, Chord

1. INTRODUCTION

Key-Value database is a type of NoSQL (Not Only SQL) database which is good for storing non-relational data. Key-Value stores are at the heart of the applications which require persistent storage to store unstructured data. Generally, a Key-Value store provides a very simple interface using 'Get', 'Put' and 'Delete' operations. A 'Get' operation retrieves the value stored for a given Key, 'Put' operations puts a new pair of Key and Value and 'Delete' removes the specific key from the persistent storage. However, when applications start scaling up it is not feasible to store all the data in a single node. Having a single node will result in performance bottle-neck as well as will be a single point of failure. Thus it becomes necessary to split the total system load into multiple servers which give the benefit of combining storage from multiple nodes and removing the performance bottle-neck as multiple nodes can handle the requests. Replicating the data on different nodes will provide high availability and remove the single point of failure. These multiple nodes should work transparently (the application and the end user should not be aware of multiple nodes) while still maintaining all the properties of a normal Key-Value store such as data consistency and high availability.

A lot of research has been done on the topic of building a distributed Key-Value store and various algorithms have been proposed. Peer-to-Peer based systems, which work in completely decentralized fashion are generally a good choice for such a Distributed Key-Value store. There are many reasons why Peer-to-Peer systems perform well. Firstly, these systems do not have a central manager node. Each node has the same capabilities and responsibilities thus making it more robust. Having a central node which manages the other nodes or maintains cluster meta-data(for an e.g. centralized list of data-node mapping) introduces a single point of failure in the system. Secondly, many algorithms have been proposed which can make use of Distributed Hash Tables (DHT) which are highly scalable to create an overlay network of all Peer nodes and route requests efficiently. Finally, Peer-to-Peer systems provide the ability to combine heterogeneous commodity hardware to store data at different geographical locations.

This project will be focused on one such distributed lookup protocol called Chord[4]. Chord is a routing protocol and supports only one operation: given a key, it maps the key onto a node. Chord uses a 128-bit SHA-1 hash function to generate a unique chord-ID for each node. All these Chord nodes are assumed to be arranged in a circular order and each data lookup follows the same circular path. A Chord node needs 'routing' information about $\log(n)$ other nodes. Because this information is distributed, a node resolves the hash function by communicating with other nodes. The cost of a Chord lookup grows as the log of the number of nodes, so it is scalable. The nodes store routing information consisting of pointers to a set of nodes at exponentially increasing distances. Chord uses consistent hashing to assign keys which tends to balance the load since each node receives roughly the same number of keys resulting in little movement of keys when nodes join or leave the system. Chord is robust and adapts efficiently as nodes join and leave the system.

2. RELATED WORK

There are few other Peer-to-Peer protocols which make use of consistent hashing. Pastry[1] uses Distributed Hash Tables to perform application-level routing and object location in a potentially very large overlay network. In Pastry, a unique Id is assigned to every node and object location is resolved based on that ID. The algorithm finds the desti-

nation of the message by doing a prefix match of the message key with the node keys present in the routing table. This algorithm ensures $\log(n)$ hops under the ideal scenario. Kademlia[8] is another such algorithm which can be used for topology control in Peer-to-Peer systems. Kademlia assigns the UUID values to each entity and uses XOR function to calculate the distance between two nodes. Node lookups are done by using routing tables which are similar to the routing tables used in Pastry. However, node lookups take place asynchronously and multiple lookup requests are used to accelerate the lookup procedure.

Many Key-Value stores have been built on top of different topology control protocols such as Chord, Pastry or Kademlia. One of the famous distributed Key-Value storage platforms is Amazon Dynamo[3]. It was developed at Amazon to suit the needs and scale of different services. Dynamo is specifically designed to have tight control over different parameters such as availability, consistency, cost-effectiveness and performance. Like many other Peer-to-Peer systems Dynamo also uses consistent hashing[6] for cluster topology, it uses gossip-based membership protocol, provides a vector clock based eventual consistency model and deals with temporary failures using Hinted hand-off mechanism. A special feature of Dynamo is that it uses zero-hop DHT. Unlike Chord or Pastry, it uses enough information in each node to route requests directly. OceanStore[5] is another persistent storage system designed to be highly available, highly distributed and provide continuous access to the persistent data. OceanSotre assigns a unique GUID to each entity in the system but unlike other databases, OceanStore does not directly impose ring topology. Instead, locations of objects are stored in a distributed data structure. Data location and routing is then done in a two-phase process with the help of a Plaxton et.al[2] object location algorithm. OceanStore also supports a consistency model based on object versioning which allows cleaner consistency maintenance in case of failures.

The other popular distributed database is Cassandra[7] which also uses ring topology like Chord. Cassandra uses Hinted hand-off to provide eventual consistency and high write availability. Cassandra uses a unique Accrual Failure Detection in a Gossip based setting for membership. Accrual Failure Detectors are very good in both their accuracy and their speed and they also adjust well to network conditions and server load conditions.

3. PROPOSED APPROACHES

Proposed Distributed Key-Value database will implement and make use of the functionality proposed by Chord. The first step would be to implement a consistent hashing scheme to assign a unique ID to all entities in the system. An ID for a particular node will be generated by taking SHA-1 of its IP-address and similarly, ID for an object will be generated by taking SHA-1 of the key associated with that object. Secondly, creating an object structure(Finger Table) for storing the finger table which stores the mapping of node key to the IP address and port number. A newly joined node will copy the finger table of its neighbour and later a stabilization protocol running in the background will keep on updating the finger tables as the network evolves. Every node runs stabilize periodically (this is how newly joined or failed nodes will be noticed by the network). A new node will join by calling join method on existing node (details of which will be externally provided). All the keys that were associated with this node and were being stored on some other node will now be shifted to this node.

For routing, each node will forward the requests to a node looked up from the Finger table entries which is the largest key less than or equal to the search key. This guarantees a node lookup operation to have an upper bound of $O(\log N)$. Each node also maintains a successor list and a predecessor list which helps in node join and failure operations. Implementation of the Chord layer will be done by using JAVA RMI mechanism.

Once the Chord layer is implemented we will build an application API layer on top of it. This API will consume the functionality provided by Chord for persisting the objects, retrieving them and replicating them. Application layer will expose a simple API consisting of 'Get', 'Put' and 'Delete' which can be used by the application developers for manipulation of the objects. Application layer will be responsible for maintaining replicas on neighbor nodes of the actual key-ID successor node. In case a node fails, all replicas of the objects stored on that node will have to be replicated somewhere else, however, we plan to implement this after everything else has been implemented and tested. The database will provide the strong consistency model where each replica needs to be updated for a successful 'Put' or 'Delete' operation. The persistent storage on each node will leverage the underlying file system for storing the objects. Finally, a failure to deliver a message to a node will be considered as a failure of that node and will trigger stabilization algorithm, no separate membership protocol is used.

4. REFERENCES

[1] P. D. Antony Rowstron. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *18th IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[2] R. R. C. Plaxton and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. *ACM SPAA*, pages 311–320, 1997.

[3] M. J. G. K. A. L. A. P. S. S. P. V. W. V. Giuseppe DeCandia, Deniz Hastorun. Dynamo: amazon's highly available key-value store. *SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, 2007.

[4] D. K. M. F. K. H. B. Ion Stoica, Robert Morris. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM '01 Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, 2001.

[5] Y. C. S. C. P. E. D. G. R. G. S. R. H. W. W. W. C. W. John Kubiatowicz, David Bindel and B. Zhao. Oceanstore: An architecture for global-scale persistent storage,. *9th international Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[6] L. L. L. P. KARGER, LEHMAN. Consistent hashing and random trees: Distributed caching protocols for

relieving hot spots on the world wide web. *29th Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[7] A. L. Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44:35–40, 2010.

[8] K. A. P. to-peer Information System Based on the XOR Metric. Petar maymounkov and david mazieres. *In 1st International Workshop on Peer-to-peer Systems*, 2002.