Sam Pastoriza
Chris Lambert
Joel Shapiro
MusicVines

Scaling in Riak

To account for scaling, Riak uses both replication and partitioning. Replication is the act of duplicating data across multiple servers, which Riak does by default. Since Riak automatically distributes data across multiple servers, it claims to achieve a near-linear performance increase as storage capacity rises. Riak will automatically redistribute data as nodes are added or removed to balance data across nodes. Riak eliminates manual sharding and hot spots through replication of data. Despite sharding the data across multiple servers, a Riak database network is masterless, meaning that any of the primary nodes can talk directly with the users, eliminating the need for a router similar to MongoDB. Furthermore, the use of a masterless system allows database maintainers to add and remove nodes really easily. Riak claims to support the ability to add and remove capacity on demand without requiring a manual restructuring of the cluster. They use an intelligent data hashing system in order to automatically distribute data evenly across nodes and redistribute it when adding or even removing a node. Surprisingly, they can reallocate this distribution of data in a non-blocking manner, meaning that the database can stay fully functional while internally restructuring itself.

The enterprise version of Riak supports “multi-datacenter replication”, which allows for data replication across servers hosted in multiple places. This allows for companies to gain benefits of locality, allowing them to serve the same information to farther reaches of the world faster. Again, this system of replication is masterless, but this time there is a single cluster which is the primary one which manages the requests of the secondary data clusters. Obviously our group will not be implementing this type of data replication as our data is only available in side of the rose-hulman firewall, so there are no issues of latency.

Partitioning is how we decide to divide a set of keys among different servers. We can choose a specific server to have a set range of keys, and then have the rest of the remaining servers host the rest of the different non-overlapping keys. The total capacity can increase as the number of partitions increase since there are more nodes on more servers, but each server has to do less work. However, if a node goes down, the partition goes down with it. This is why Riak uses the combination of replication and partitioning for scaling and availability. By portioning the database in this manner, Riak allows its larger databases to be run on a larger number of less powerful machines rather than requiring each machine become more and more powerful. This allows databases to scale horizontally as the dataset grows without having to pay an ever increasing amount of money on individually more powerful machines as the database will spread the work over multiple machines more efficiently.

From version 2.0, Riak supports strong consistency and high availability. This should break the CAP theorem, but Riak only sets a bucket type property as strongly consistent. This means that either a request is successfully replicated to a majority of partitions or it will fail. However, if too many nodes go down, you have lost high availability and the write will fail. To resolve this, you need to try again after repairing the node.

Riak defaults to high availability, but since we currently only storing usernames and hashed passwords in Riak, it is critical that we have strong consistency. We will divide the usernames, since they are unique, into two partitions on two different nodes, and then replicated those two nodes, so there will be a total of 4 nodes. This means there are two pairs of nodes that have two sets of the same data. Riak uses a variable called n_val to describe the number of duplicated nodes to create. The default is three, but we will use n_val = 2, since this will divide the number of usernames equally.