15214 HW9 Design Documentation

Carlos Gil (cgil)

Nick Zukoski (nmz)

To partition the data, we decided that each user would have all of their data stored on only one server. To implement this, every time a client needs to talk to the backend, it chooses a random server to send its requests to. So when a new user is registered, its home server is selected as whatever server the client happens to have sent the registration request to. Once the user is registered on this server, the server sends out a broadcast to all the other servers notifying them that the new user has been registered and which server they live on.

If a server receives a request for data that lives on another server, it asks the correct server for the data and then routes it back to the client. Each server maintains a duplicate copy of a lookup table from user emails to server IDs that they live on.

The cache is a simple LRU cache that maps request strings to their string responses. Each server has their own cache that caches responses. So if a request comes to Server 1 for data about a user stored on server 2, server 1 will forward the request to server 2 and then relay the response back to the client. At the end of the interaction, both server 1 and server 2 will have identical entries in their caches as they do not differentiate between requests sent from other servers and from clients.

The pros of partitioning in this way is that as the number of users scales, there is no bottleneck in routing the response to the correct server as every server is at most one step away. One con is that the data is very distributed and a single request, say for the first 10 statuses to display in the newsfeed, may have to query 10 different servers. There is very little data redundancy, so if one users page became very popular it could swamp the server it was on (even with a cache).