Distributed Key Value Store

Submitted by: Shubhi Mittal, Shraddha Phadnis

Design Specification:

Implement an in-memory distributed key-value (KV) store. The KV store should be able to Store data across multiple nodes i.e. at any given time, a single node might not have all the data. The following are the components to be designed.

1. Database server: A server process communicates with client through public Restful APIs.The KV store is essentially hosted on a data server.
2. Client: A client is essentially any restful client that communicates with the servers indirectly through a proxy. For simplicity, we use http curl requests to simulate client.
3. Proxy: A proxy server acts as a mediator between client and database servers. Proxy server is responsible for routing client's request across multiple data servers and manage communication across data servers.. The proxy server also acts as a load-balancer and ensures a uniform workload distribution among various servers.

Features**:**

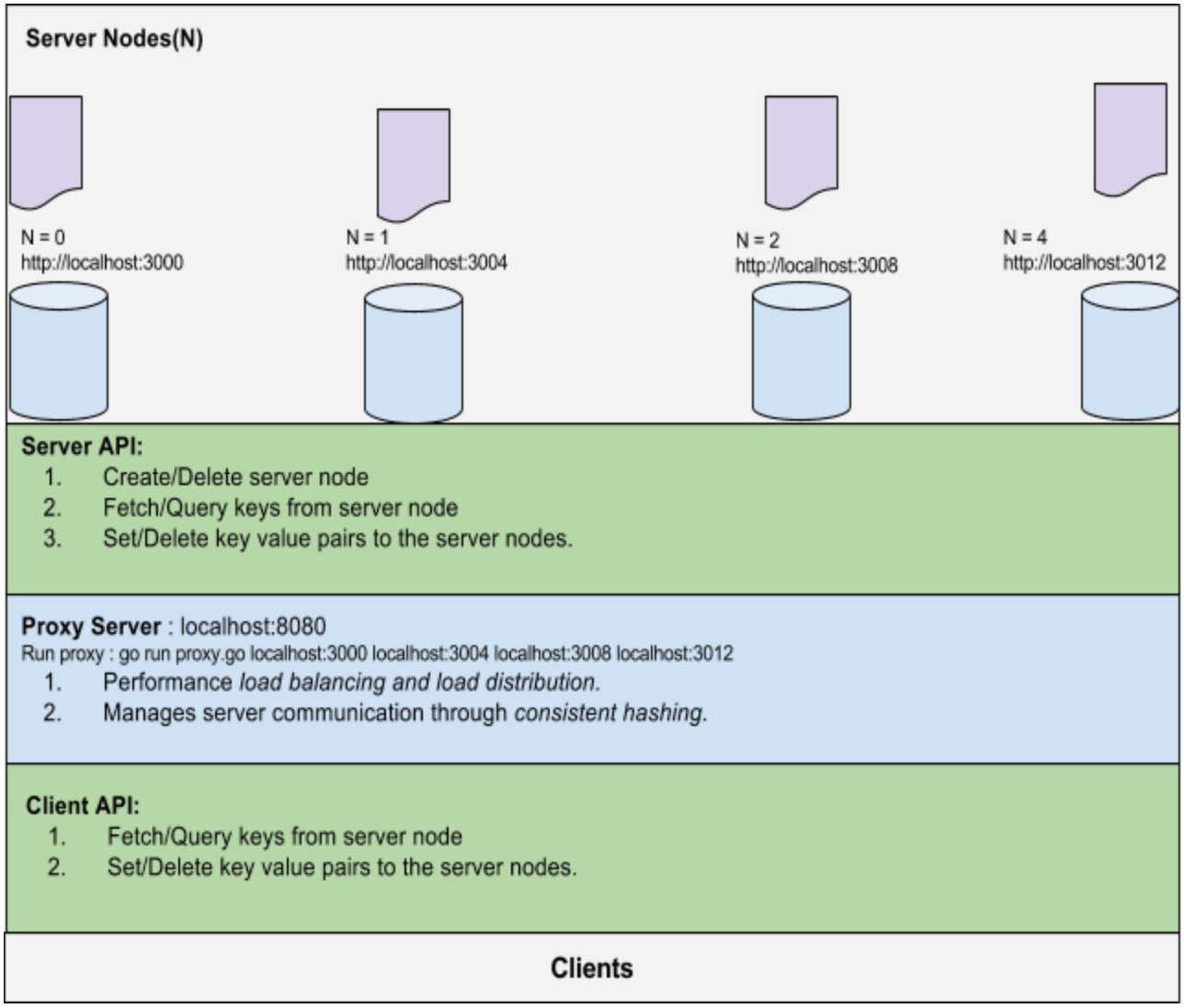
1. Adding/Deleting database server node.
2. Creating/Deleting key value pairs in the database.
3. Fetching/Querying key value pairs in the database.

API Design for communication between Client<->Proxy and Proxy<->Server:
https://piazza.com/class/j6zju0uhbwv6fc?cid=108

Distributed Key Value Store

Submitted by: Shubhi Mittal, Shraddha Phadnis

Application Architecture:



As shown above, the application has a layered architecture with each layer having two way communication with its adjacent layer. The top most layer consists of server nodes. Each server node is an individual process with a live database object (in memory key value store). In addition to the in-memory database object. Each server also consists of a persistent json file which gets updated on each successful write api call to the server. The usage of a persistent key value store becomes important in case of node failure which is explained in detail in Fault tolerance section. Proxy server communicates with the server nodes using server api's. At proxy, through consistent hashing, load is distributed amongst servers. Client is abstracted from the knowledge of serves and how load is distributed amongst servers. Client has direct interaction with proxy and at any point proxy resolves the client's request to the best of its knowledge.

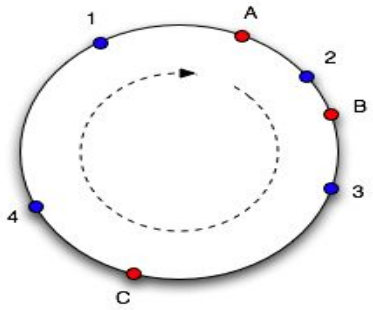
Distributed Key Value Store

Submitted by: Shubhi Mittal, Shraddha Phadnis

Consistent Hashing:

The basic idea behind the consistent hashing algorithm is to hash both objects and caches using the same hash function, The reason to do this is to map the cache to an interval, which will contain a number of object hashes. If the cache is removed then its interval is taken over by a cache with an adjacent interval. All the other caches remain unchanged.

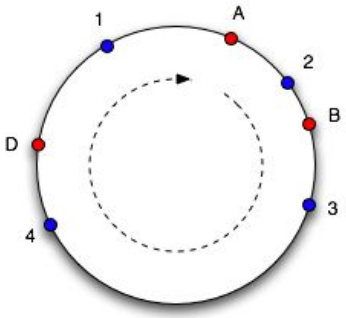
Cache in our design can be considered to be as server nodes and objects are the Key-value pairs.



Load Balancing:

For example there are 3 caches and {1,2,3,4} objects. Then, objects are filled in a clockwise order such that Object 1 is in cache ‘A’, Object 2 in is cache 2 and so on.

If any of the nodes are removed. For example if cache C is removed only elements in cache c are distributed accordingly(In our example object 3 will be placed in cache ‘A’) and rest of the objects are remained unchanged. If additional node is added at the position marked, it will include all the elements before and after that bucket. As in the below example if cache D is added at the position marking then 3, 4 are included in cache D



Distributed Key Value Store

Submitted by: Shubhi Mittal, Shraddha Phadnis

Fault tolerance:

Fault tolerance refers to the ability of the system to recover from any type of failure. In this application design fault tolerance is achieved at two levels:

1. Fault tolerance at the server nodes: In order to keep the data available in case of failures, there is a persistent json file which is written each time a set request is received for a specific server node. The set request is an atomic transaction which is only successful once all the key-value pairs in the request are updated in the in-memory database and also written to the persistent storage. In order to avoid the cost of writing in persistent storage, we will write the data to the file only one time/set api call. When a node fails, it can be restarted using this json file as a backup image and processing can continue.
2. Fault tolerance at the proxy server: Since proxy is independent of servers, A failure at proxy does not results in any loss of data. Hence proxy can simply be restarted to continue operations.