CSE 138 - Assignment 3

University of California, Santa Cruz

Fall 2020

November 18th, 2020

Rashmi Chennagiri
rchennag@ucsc.edu

Nazib Sorathiya
nssorath@ucsc.edu

Michael Covarrubias
mdcovarr@ucsc.edu

Proxy

If a node is not able to cater to the client's request (for example, if it does not have the required key, or if it has to insert/update the key in another node), it builds a proxy URL and forwards the request to the correct node.

1. Check if the node that was directly contacted by the client contains the specified key.
2. If Yes, then the main node will perform the UPDATE, GET, DELETE operations respectively.
3. If No, the main node will search for the key in the rest of the nodes in the view. If the key is found in one of the nodes, then it will send the UPDATE, GET, DELETE to the node that has the key respectively and get the response from the node to pass to the client. (We are using a proxy URL to send requests internally. When a proxy URL is received, it is understood that it is not the request received from the client and the request has to be executed on that node itself.)
4. If the key is not present in the rest of the nodes, then the main node will insert the key into the node with the lowest number of key-count to distribute the keys uniformly in all the nodes.

The proxy node does not forward any requests.
All the validations on the request json are done in the main node before forwarding it to the proxy node.

PUT Partitioning

When inserting a key, we check the key-count in all the other nodes.
We add the new incoming key into the node with the lowest key count.
This ensures that the key count across all the nodes is balanced even if no explicit request for resharding is received.

Change View Algorithm

The view-change algorithm gets triggered when a client makes a request to a node in the view. Thus,

1. Given that the current VIEW=’10.10.0.4:13800,10.10.0.5:13800’

```
VIEW='10.10.0.4:13800,10.10.0.5:13800'


          +-----------------+              +-----------------+
          |     shard1      |              |     shard2      |
          |     (node1)     |              |     (node2)     |
          | 10.10.0.4:13800 |              | 10.10.0.5:13800 |
          |  ------         |              |  ------         |
          |  - key1         |              |  - key2         |
          |  - key3         |              |  - key4         |
          |  - key5         |              |                 |
          +-----------------+              +-----------------+
```

2. Say view is triggered with a new view, from client to **node1** VIEW=’10.10.0.4:13800,10.10.0.5:13800,10.10.0.6:13800’. Then **node1** needs to notify all other nodes in the current view about the initiation of the view change
3. This is done via a proxy **PUT** request.

```
+-----------------+                                 +-----------------+
|     shard1      |                                 |     shard2      |
|     (node1)     |                                 |     (node2)     |
| 10.10.0.4:13800 |    PUT /proxy/view-change       | 10.10.0.5:13800 |
|  ------         | ----------------------------->  |  ------         |
|  - key1         |                                 |  - key2         |
|  - key3         |                                 |  - key4         |
|  - key5         |                                 |                 |
+-----------------+                                 +-----------------+
```

4. Once all nodes in the view know about the view change. All nodes start to re hash their own keys. With the new information of the **new** view. For example, node1 will re hash it’s keys key1, key3, key5. This is done to determine which keys to keep and which to send to other nodes in the new view.

```
new_distribution = {
  "10.10.0.4:13800": {"key1": "value1"},
  "10.10.0.5:13800": {"key3": "value3"},
  "10.10.0.6:13800": {"key5"}: "value5"}
}
```

For example, this states that node1 will keep key1, but will distribute key3 to node2, and key5 to node3.

5. Node to node distribution of the keys is done via a proxy path '/proxy/receive-dict'. For example, sending **{"key3": "value5"}** to node2 will be sent to path **'http://10.10.0.5:13800/proxy/receive-dict'**

```
+---------+                                       +--------+                +--------+
| Node 1 |                                        | Node2 |                 | Node3 |
+---------+                                       +--------+                +--------+
     |                                                 |                          |
     |                                                 |                          |
     |     PUT /proxy/receive-dict                     |                          |
     |       {"key3": "value3"}                        |                          |
     | ---------------request----------->              |                          |
     |                                                 |                          |
     |                                                 |                          |
     |     PUT /proxy/receive-dict                     |                          |
     |       {"key5": "value5"}                        |                          |
     | --------------------request---------------------------------> |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     | <--------------response-----------              |                          |
     |                                                 |                          |
     |                                                 |                          |
     | <------------------------response---------------------------  |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     |                                                 |                          |
     v                                                 v                          v
```

6. Once the redistribution of keys between nodes is done. The node that received the view change request from the client, responds to the client. In this example it’s node1

```
+---------+                                  +--------+
| client |                                   | Node1 |
+---------+                                  +--------+
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     | <--------------response-----------        |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     v                                           v
```

Appendix

References

- https://pypi.org/project/uhashring/ (Python packaged used for consistent hashing).
- Aleck Zhang (For the idea of adding ASCII art).