Design Documentation

Distributed Storage Service

Features Implemented -

i. Data Partitioning & Replication using Consistent Hashing
ii. High Availability for writes using vector clocks with reconciliation during read
iii. Temporary failure in replicas handled by sloppy quorum technique and hinted hand-off
iv. Failures of nodes detected by a gossip-based distributed failure detection

We have 4 physical systems, whose IP addresses are the following:

- 172.18.16.38
- 172.18.16.86
- 172.18.16.47
- 172.18.16.123

As of now, we are running “REST-WEB-SERVER” on 172.18.16.47(Any node can be used as REST server). We have used “MD5” hashing algorithm. Now servers’s IP addresses are kept on REST Server where a consistent hash is implemented as key-value pairs using MD5 hashing algorithm. Now as per consistent hashing concept, the object data will be distributed among the nodes based on their MD5 hash. For example, if hashing algorithm is “MOD-128”, a list like the following will be created.

Note: Here we used MOD-128 for demo purpose only. **In our actual system implementation we are using MD-5 which is a 2^128 based hashing scheme.**

Key	**Node**
124 - 38	172.18.16.38
39 - 47	172.18.16.47
48 - 86	172.18.16.86
87 - 123	172.18.16.123

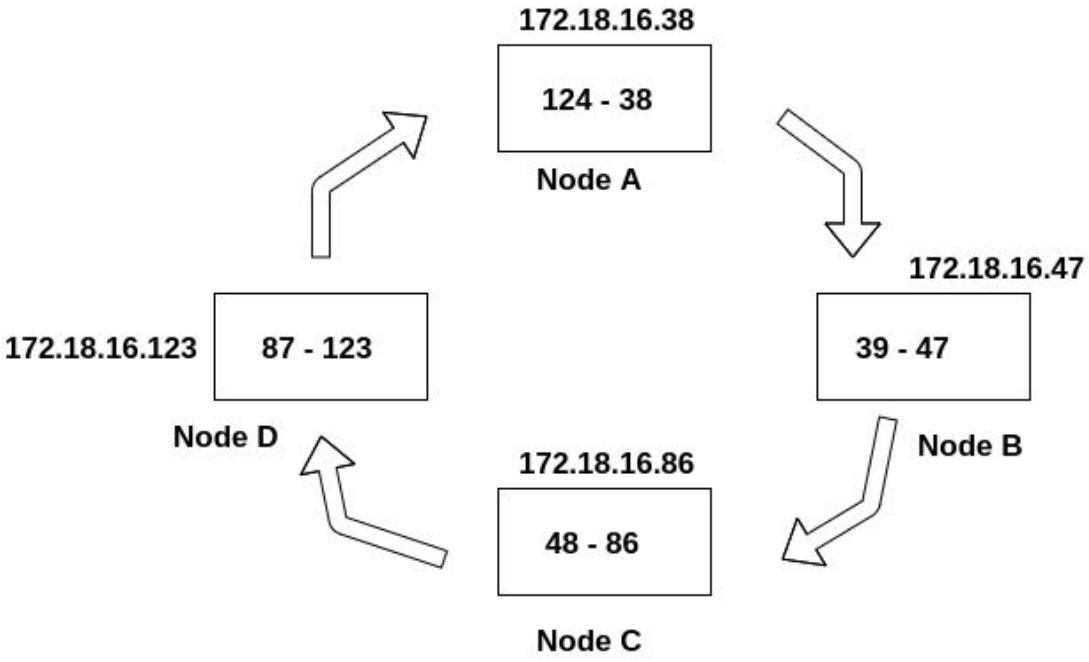

Fig. 1 Topology

Note that the list is sorted so that further process of creating/deleting bucket, creating/deleting any file will be easier.

Design considerations:

Number of nodes in the system (n)	4
Number of nodes necessary for read operation (R)	2
Number of nodes necessary for write operation (W)	2
Replication factor (N)	3

Now let us consider operations on our system one by one.

Creating the bucket:

As of now we will assume that “REST-WEB-SERVER” is running on 172.18.16.47. We have set the value of “N” as 3, meaning that when bucket/file is created on node A, its replicas will be maintained on node B and node C. Let us say, a request to create “bucket1” has come to “REST-WEB-SERVER”. Now hash will be calculated as per the name of the bucket. Let us say this hash value comes out to be 50. Now a **preference list** will be fetched on the REST SERVER which will comprise of the following nodes:-

1. 172.18.16.86 - Node C
2. 172.18.16.123 - Node D
3. 172.18.16.38 - Node A

Now a folder named as “bucket1” will be created on 172.18.16.86. And as N=3, it will be replicated to other nodes of preference list as well which are 172.18.16.123 and 172.18.16.38.

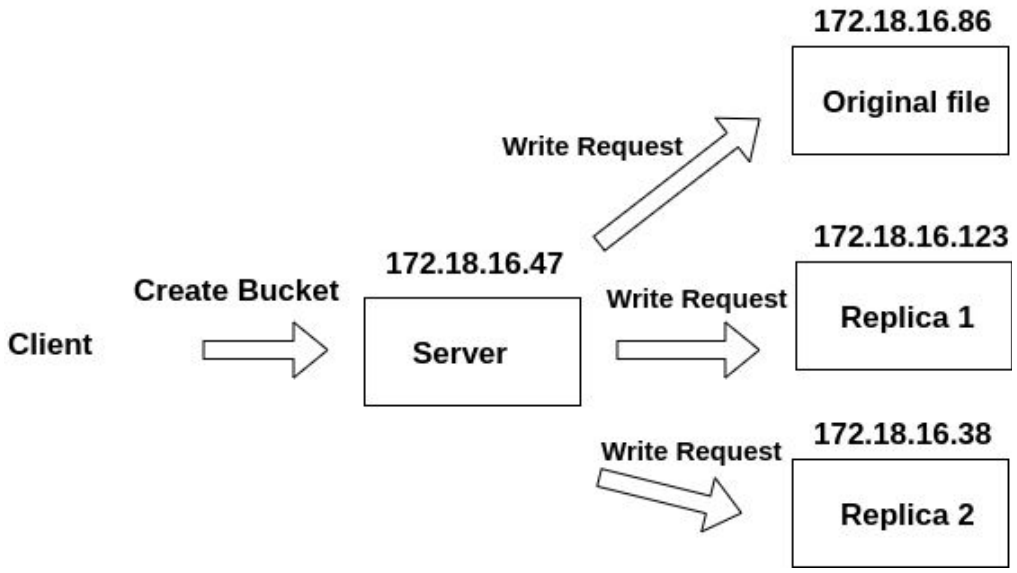

Fig. 2 Creating Bucket

Creating/Updating the file:

Let us say, user wants to upload the file named as “file1” in “bucket1”. Now this request comes to REST SERVER. It will create the file replica on all nodes in the preference list which are

172.18.16.86(Node C), 172.18.16.123(Node D) and 172.18.16.38(Node A). We are handling communication between nodes using socket programming.

As we have set the value of **"W" as 2**, once any two nodes reply as positive, it will send positive acknowledgement to client.

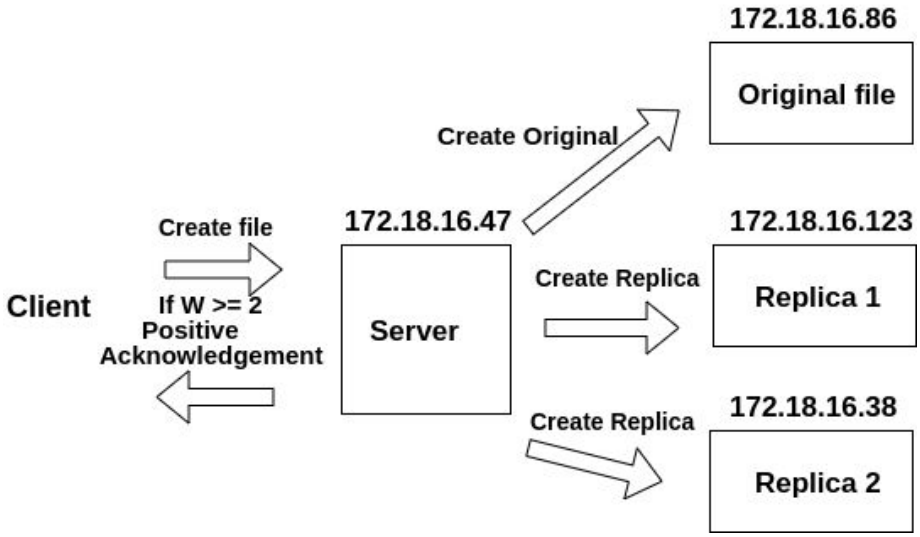

Fig. 3 Creating file

Handling vector clock:

For every file we are keeping one metadata file. In this metadata file we are keeping version number of the data which is nothing but an integer. Now when client updates any file, we are returning version number which is nothing but vector clock. This is how we have implemented **vector clock mechanism**.

Reconciliation / Read repair

Now if client receives multiple version clocks with respect to single update, latest version clock will be considered at client's end and will also be updated in the system. This is how we are implementing **reconciliation mechanism** and **read repair mechanism**.

Hinted hand-off and sloppy quorum:

Let us say, write request comes to write a file on node 172.18.16.38 (Node A). Now as two replicas are to be maintained, file should also be saved on 172.18.16.47 (Node B) and 172.18.16.86 (Node C). Note that circular order according to consistent hashing is maintained. If those two nodes are unavailable, we are saving the file on the node 172.18.16.123 (Node D) along with the hint. Now whenever the unavailable nodes becomes available, hint is sent to those nodes by node D. Now the recovered nodes will save the file as per told. This is how we have implemented **hinted hand-off mechanism.**

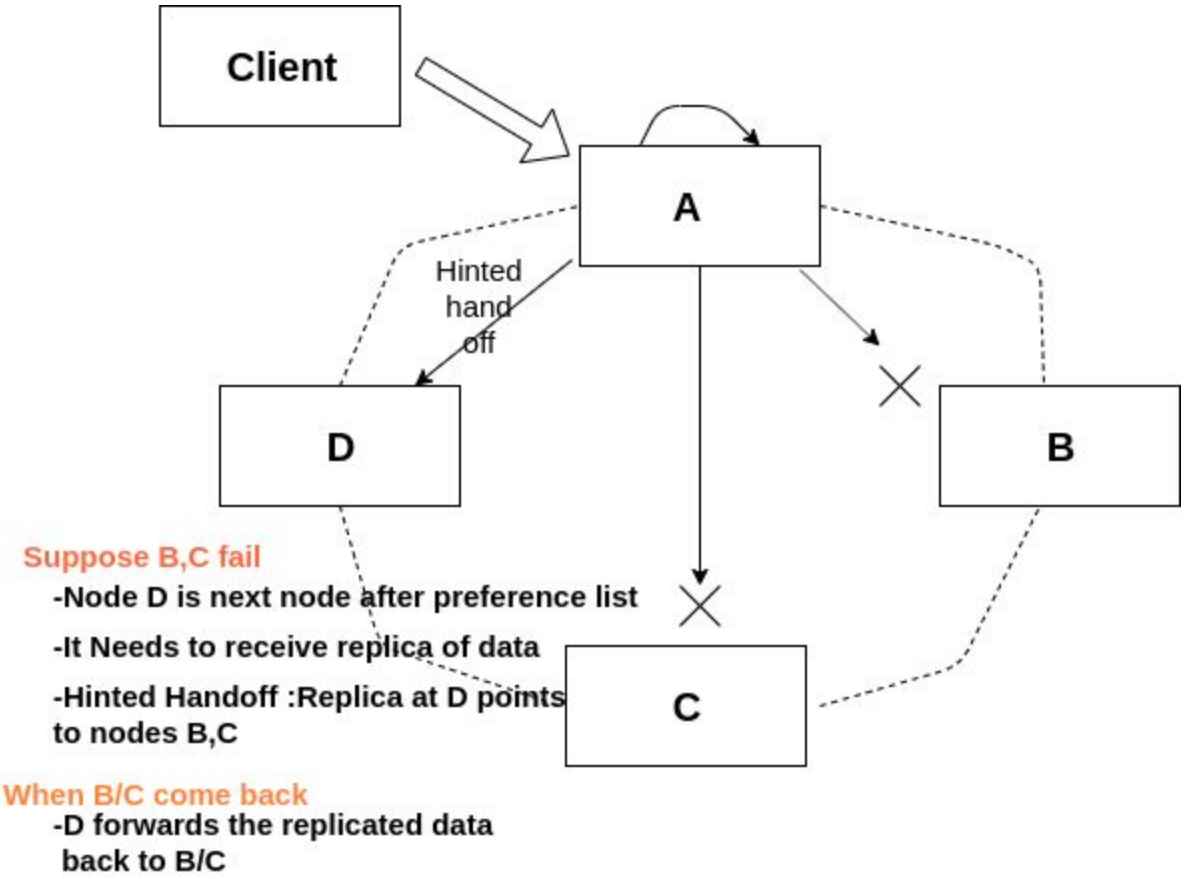

Fig. 4 hinted hand-off

Now actually it was not the responsibility of node D to save that file, but still file got saved on node D. So in a way D acted as a backup server for a while. However it is actually part of our system only. So in this way we have implemented **sloppy quorum mechanism** where the following factor is successfully implemented and verified.

R + W > N

Where,

R = Number of nodes necessary for read operation (2)
W = Number of nodes necessary for write operation (2)
N = Replication factor N (3)

Output:

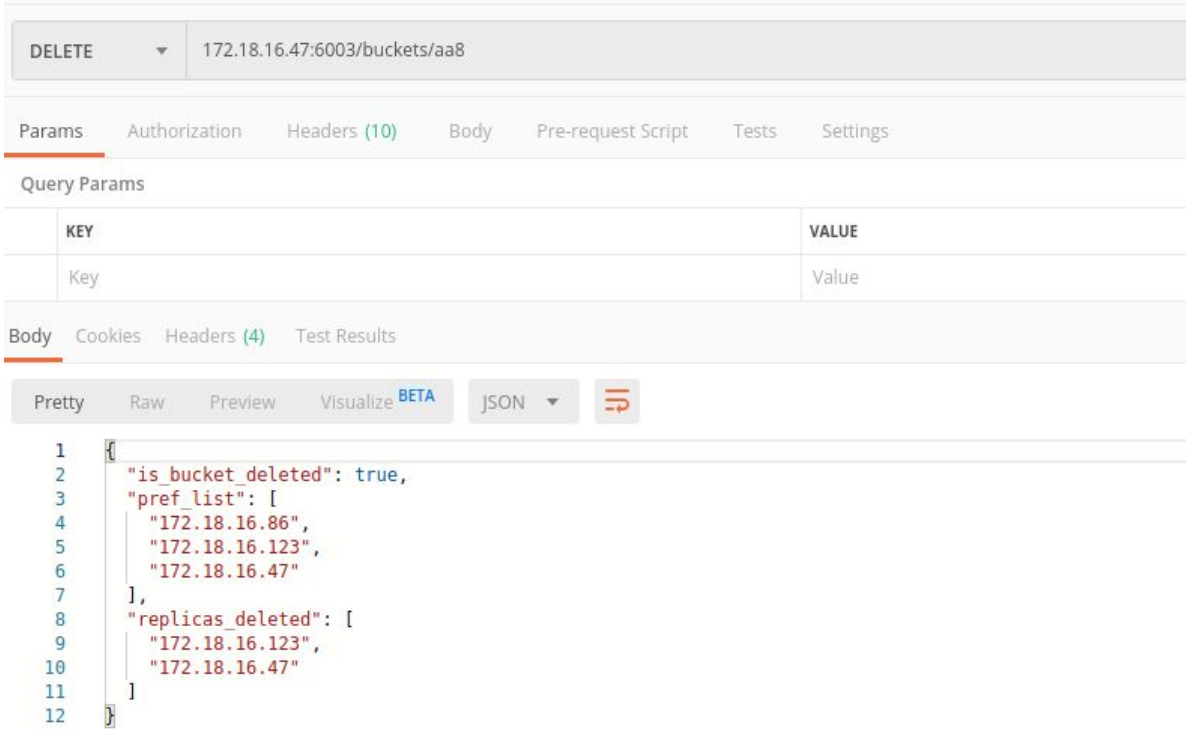

Fig 5. Bucket Deletion

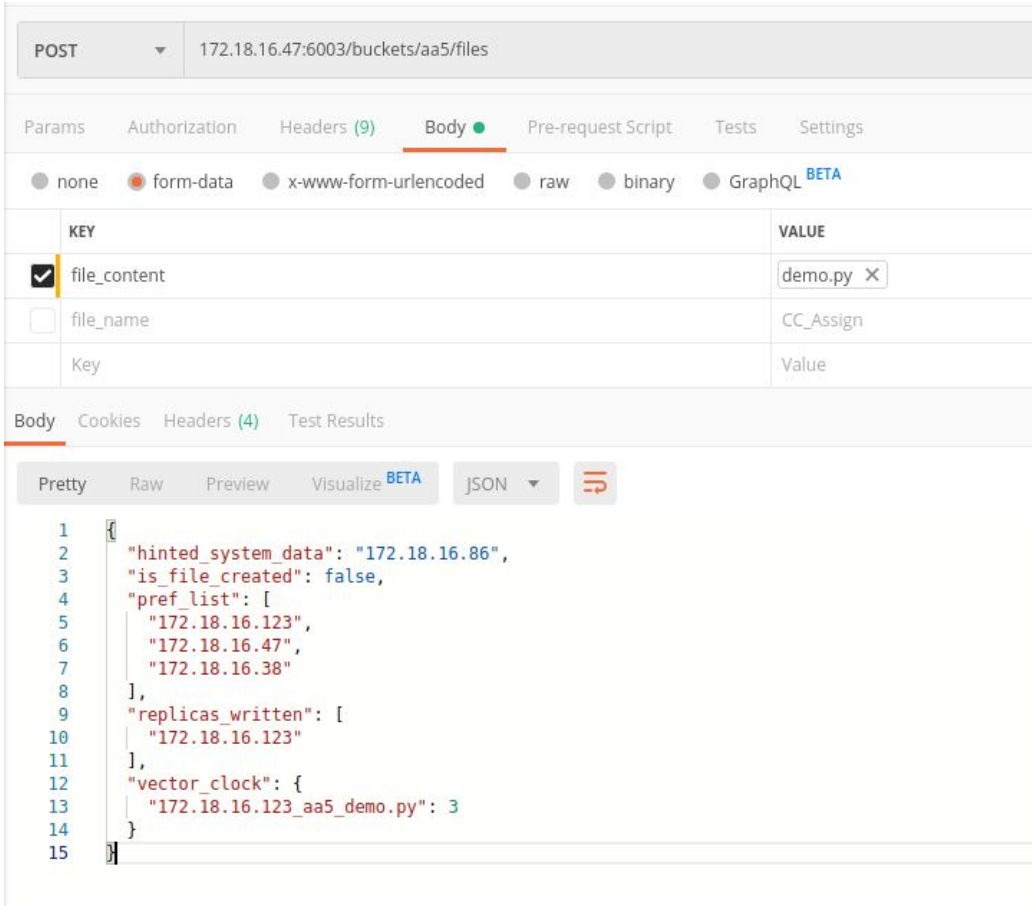

Fig 6. File Creation with Hinted Hand Off

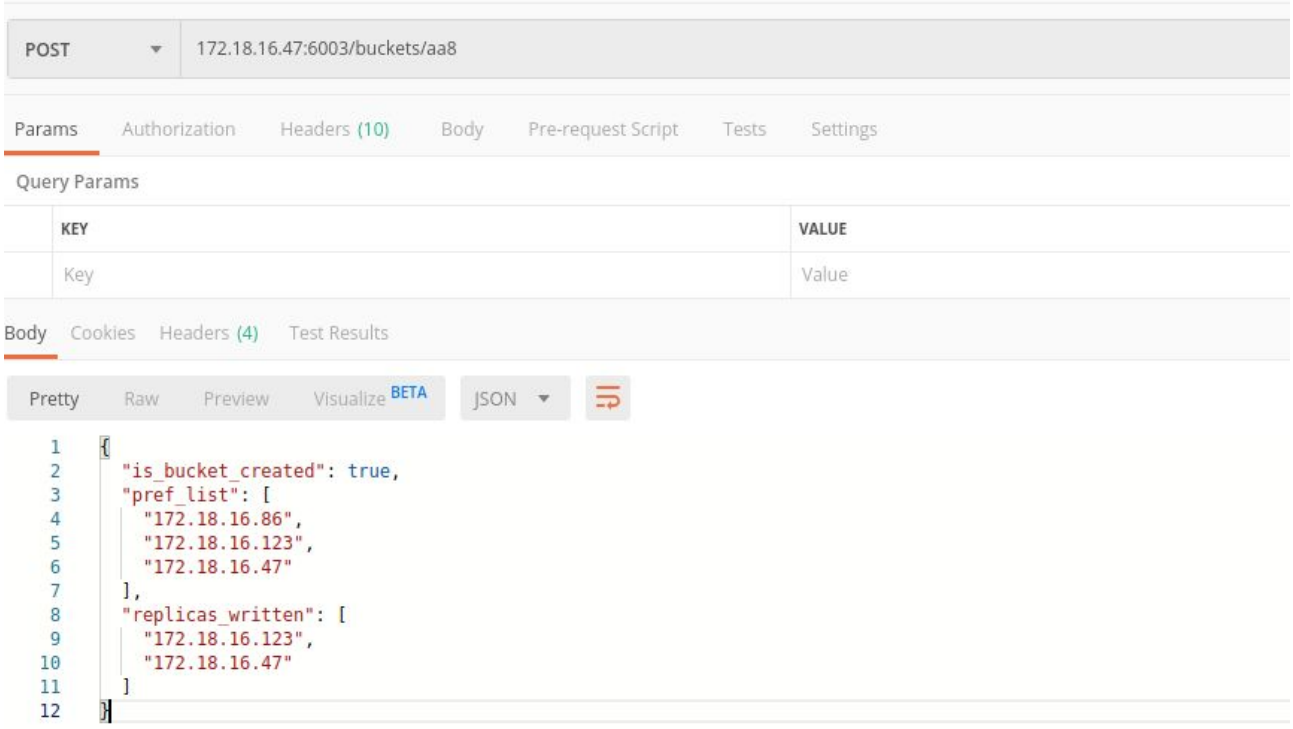

Fig 7. Bucket Creation

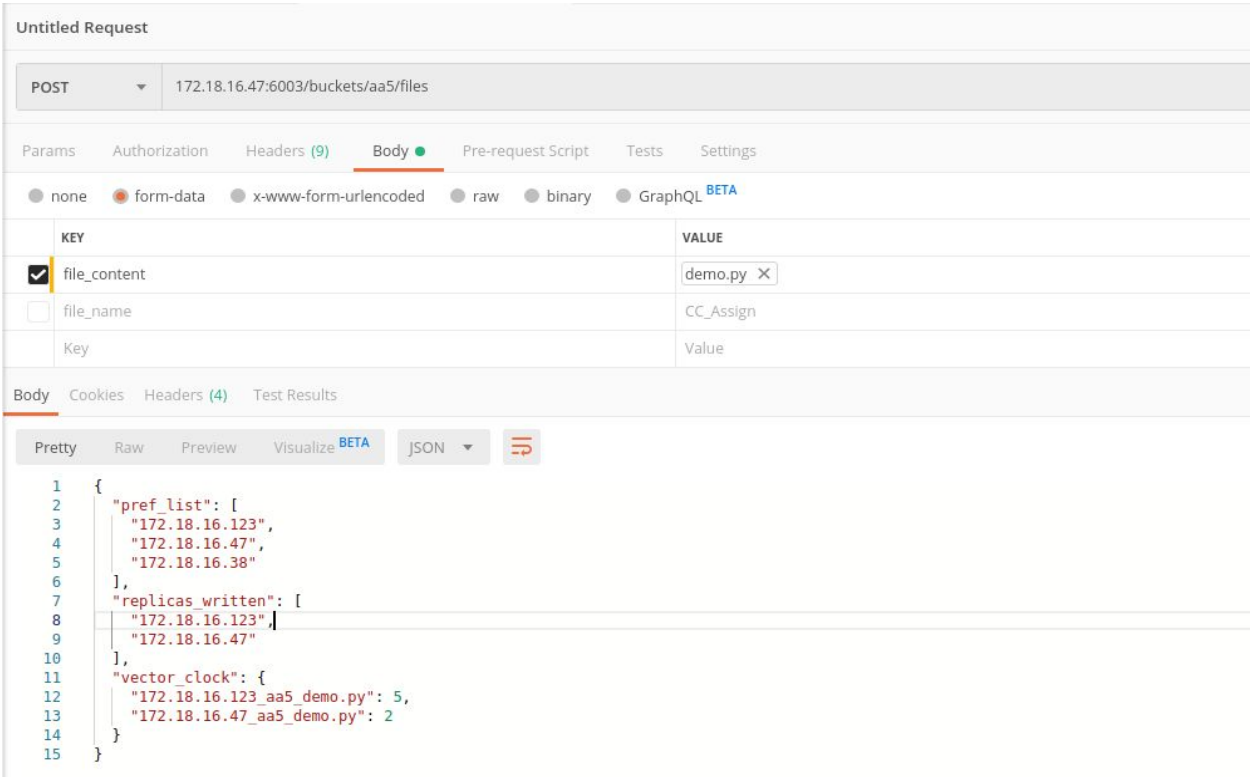

Fig 8. File Creation Returning Replicas of Vector Clock