Performance Evaluation

- To evaluate the performance of the system, especially, PUT, GET and DELETE operations, I have created a Test program (file DHTTest.java)
- The Test programs asks you the type of operation you want to evaluate – GET/PUT/DELETE, number of threads (clients) to run in parallel and number of times the specified operations must be executed by each thread.
- This program was run on various combinations of the input parameters. So the test environment was as follows:

  Total no. of peers connected: 3

  Configuration of each peer: 2 core processor, 1 GB RAM virtual machine (Ubuntu Linux OS)

<u>100k Requests on 1 Client:</u>

Test program running on no. of peers in parallel: 1

Total no. of requests sent to the server: 100,000

Results are as follows:

Sr. No.	PUT Operation	GET Operation	DEL Operation
1	4425 seconds	4253 seconds	4207 seconds

Average Time per PUT Request: 0.044 seconds

Average Time per GET Request: 0.042 seconds

Average Time per DEL Request: 0.043 seconds

<u>100k Requests on 2 Client:</u>

Test program running on no. of peers in parallel: 2

Total no. of requests sent to the server: 200,000

Results are as follows:

Sr. No.	PUT Operation	GET Operation	DEL Operation
1	4652 seconds	4220 seconds	4238 seconds
2	4645 seconds	4232 seconds	4240 seconds
AVG.	**4648 seconds**	**4226 seconds**	**4249 seconds**

Average Time per PUT Request: 0.046 seconds

Average Time per GET Request: 0.042 seconds

Average Time per DEL Request: 0.043 seconds

<u>100k Requests on 4 Client:</u>

Test program running on no. of peers in parallel: 4

Total no. of requests sent to the server: 400,000

Results are as follows:

Sr. No.	PUT Operation	GET Operation	DEL Operation
1	4852 seconds	4251 seconds	4225 seconds
2	4860 seconds	4257 seconds	4230 seconds
3	4854 seconds	4248 seconds	4227 seconds
4	4868 seconds	4260 seconds	4228 seconds
AVG.	**4858 seconds**	**4254 seconds**	**4227 seconds**

Average Time per PUT Request: 0.048 seconds

Average Time per GET Request: 0.042 seconds

Average Time per DEL Request: 0.043 seconds

100k Requests on 8 Client:

Test program running on no. of peers in parallel: 8

Total no. of requests sent to the server: 800,000

Results are as follows:

Sr. No.	PUT Operation	GET Operation	DEL Operation
1	5318 seconds	4246 seconds	4241 seconds
2	5301 seconds	4246 seconds	4242 seconds
3	5301 seconds	4243 seconds	4240 seconds
4	5302 seconds	4245 seconds	4242 seconds
5	5318 seconds	4243 seconds	4241 seconds
6	5317 seconds	4243 seconds	4241 seconds
7	5316 seconds	4242 seconds	4243 seconds
8	5316 seconds	4245 seconds	4242 seconds
AVG.	**5311 seconds**	**4244 seconds**	**4241 seconds**

Average Time per PUT Request: 0.053 seconds

Average Time per GET Request: 0.042 seconds

Average Time per DEL Request: 0.042 seconds

Observation:

- In the above evaluations, we can observe that the average time for PUT request is around 0.047 seconds (47 milliseconds). Most of the time when I was debugging my program, the time to serve PUT request was around 0.045 seconds (± 0.005 seconds).
- Similarly, the average time for GET request was around 0.042 seconds (42 milliseconds) and for DEL (DELETE) request was 0.042 seconds (42 milliseconds). While debugging the, most of the time response time for GET and DEL request was around 0.045 seconds (± 0.005 seconds).

- This time i.e. 0.042 seconds involve connection to the peer time, server processing time and sending the results and connection closing time.
- Each combination took around 4200 – 5300 seconds (70 – 88 minutes) because the 100k requests were sequential in case of only one peer whereas in case of 8 peers, 800k requests were divided into 100k requests on each peer, thus creating 100k sequential requests on each peer.
- From the above evaluations it is clearly understood that multi-threading increases the performance because time taken to process 100k requests -> 1 thread and 800k requests -> 8 threads is almost same.
- Note that in case the hash function returns the node id of self-node (self-peer), then the request time is around 0.004 seconds (4 milliseconds) (± 0.002 seconds).
- So, if we are processing only 100k requests, we can divide it into multiple threads to get the job done faster. For instance, 100k requests if divided into 5 threads will complete the entire job in approximately 10 – 15 minutes in my Virtual Machine.

<u>Evaluation of Hash function:</u>

- To test my implementation of hash function, I have created a program (file HashTest.java) which reads unique words from a text file and performs hash on all those words and adds it to the Hash Table. Note that all keys are unique for this evaluation.
- I then check how many values are present in each bucket of Hash Table. This evaluates how the hash functions spreads keys across the various peers.
- Proper spreading is important because if we have 10k keys and 5 nodes and it shouldn't be the case that 2 nodes get around 4k keys each and remaining two get around 1k keys each.
- Results of Hash function evaluation were:

  Total number of unique words (Keys): 12602

  Simulation of distribution of keys across the network –

Node No.	<Key, Value> pairs count
1	1304
2	1180
3	1275
4	1290
5	1238
6	1318
7	1243
8	1251
9	1230
10	1273

<u>Observation:</u>

As we can observe in above table that the distribution of keys amongst the distributed hash tables is good. The difference is not that bad.