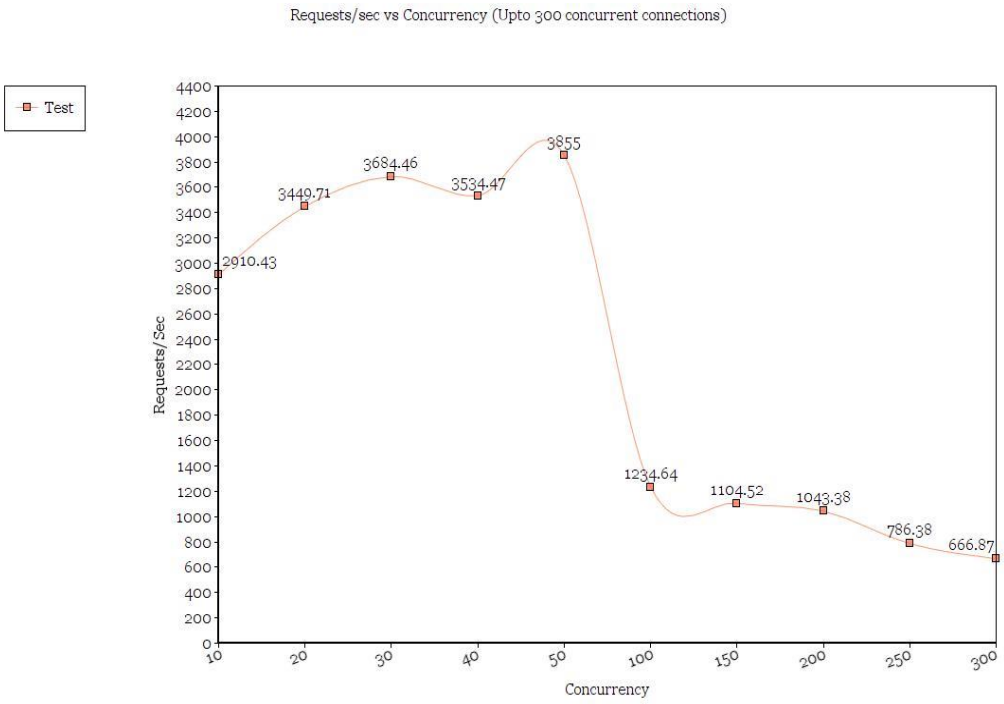
1. Benchmark 1

Request per sec versus concurrency



In the above experiment, total number of requests sent were kept constant to 50000 where as concurrency was increased till 300. The thread-pool size on the server was configured as 100.

The experiment result shown that, server's requests per second handling number was abruptly decreased to 1234.64. Moreover, in the further experiment the request per second handling rate was gradually decreased.

2. Benchmark 2

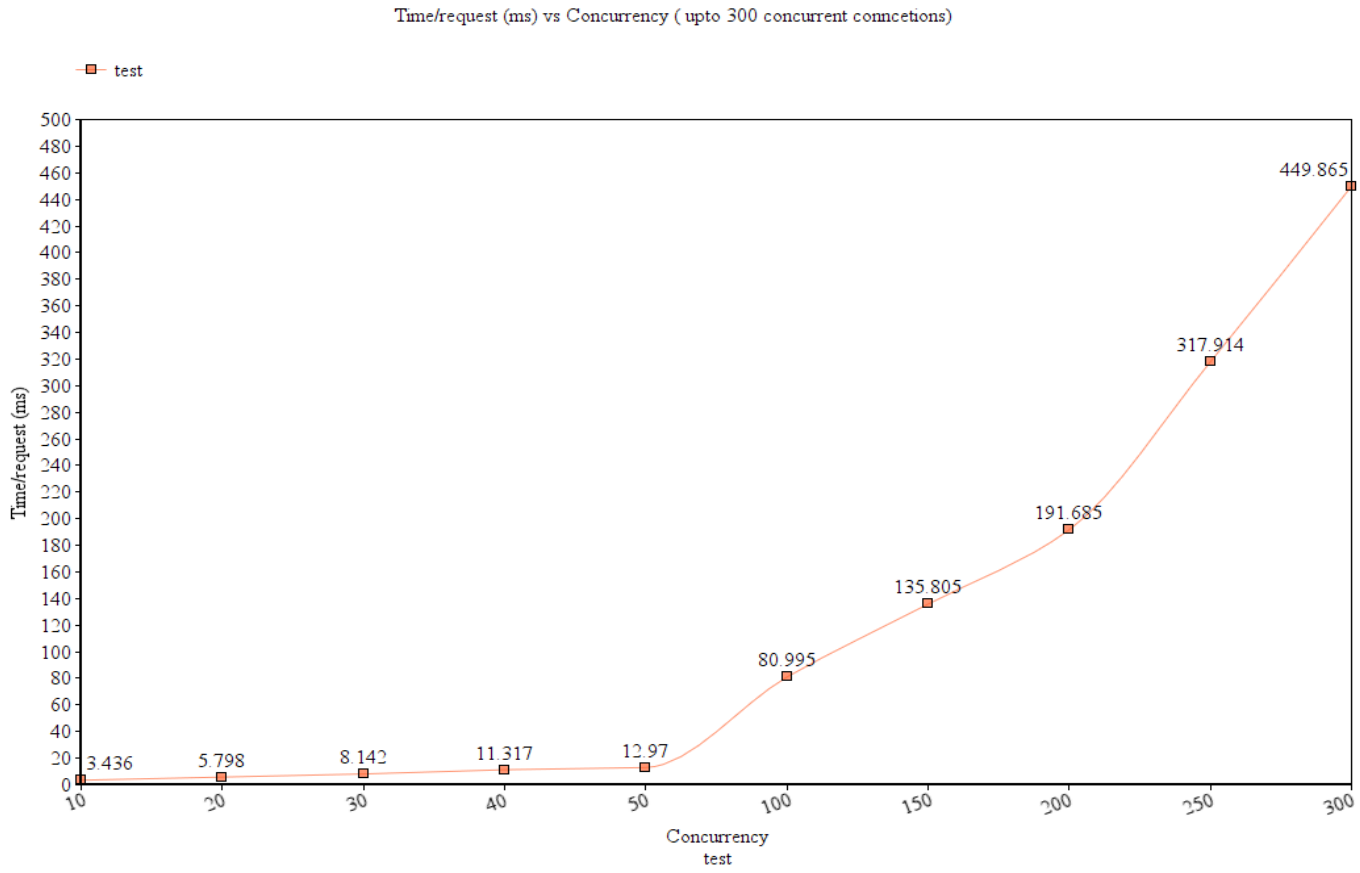


Figure 2. Time/req vs concurrency test

In the above experiment, the test environment was kept same i.e. total 50000 transaction requests were sent, while increasing concurrency from 10 to 300. Thread pool was configured to 100.

In the Graph of above experiment in figure (2), we observed that, average time per request was increased abruptly when concurrency crossed to ~80. It went on increasing till the full experiment to concurrency 300.

3. Possible problems and solution for the Server implementation

1. I think, the environment could be the problem, we can benefit from multithreading when we have enough cores. Ideally (no of threads = no of cores) is the ideal equation. Experiment can be carried out on cloud like infrastructure where client and server both running on different VM's. WE can also configure server with high configuration such as Memory and CPU cores size.
2. May server threads are starving when obtaining the lock and doing the transaction. Need to look into using some tool to identify bottlenecks
3. Need to check Hashing key for the HashMap. May be there are Hash conflicts and values are being put in the bucket link list.