PRAKTIKUM: CLOUD DATABASES
(WiSe 2015/16)

Milestone 3 - Performance Test Results

Team (In The Cloud): Spyros Lalos, Sreenath Premnadh, Athanasios Akanthos Chasapis

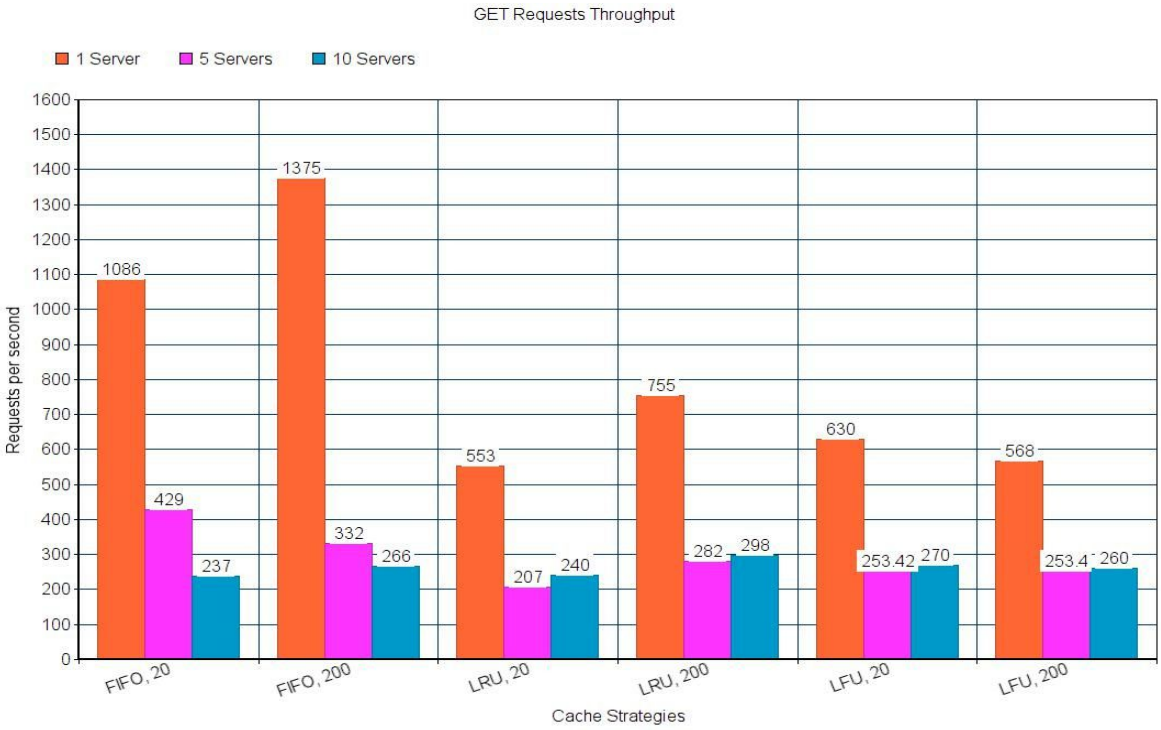
Objective:
To test the system with varying configurations and collect metrics pertaining to serving of GET and PUT requests.

Methodology:
A test suite was created in Java (present in src/performance package) that started an arbitrary number of server threads in different configurations. For the evaluation, the number of client threads were fixed at 200, in order to compare the efficiency and scaling of the service for different configurations under the same stress. For running more clients concurrently, we observed limitations with high number of open file descriptors in the systems we used, mostly due to the fact that the measurement tests covered a broad variety of server configurations (different strategies, cache sizes etc.). We tested all cache eviction policies (FIFO, LRU, LFU) and two different cache sizes of 20 and 200 slots each, so that with the smaller one the client keys do not fit and we need to use the persistence mechanism, while with the bigger one all client keys can be served from memory. In each of the configurations, a fixed number of GET and PUT requests were fired by the clients to the servers.

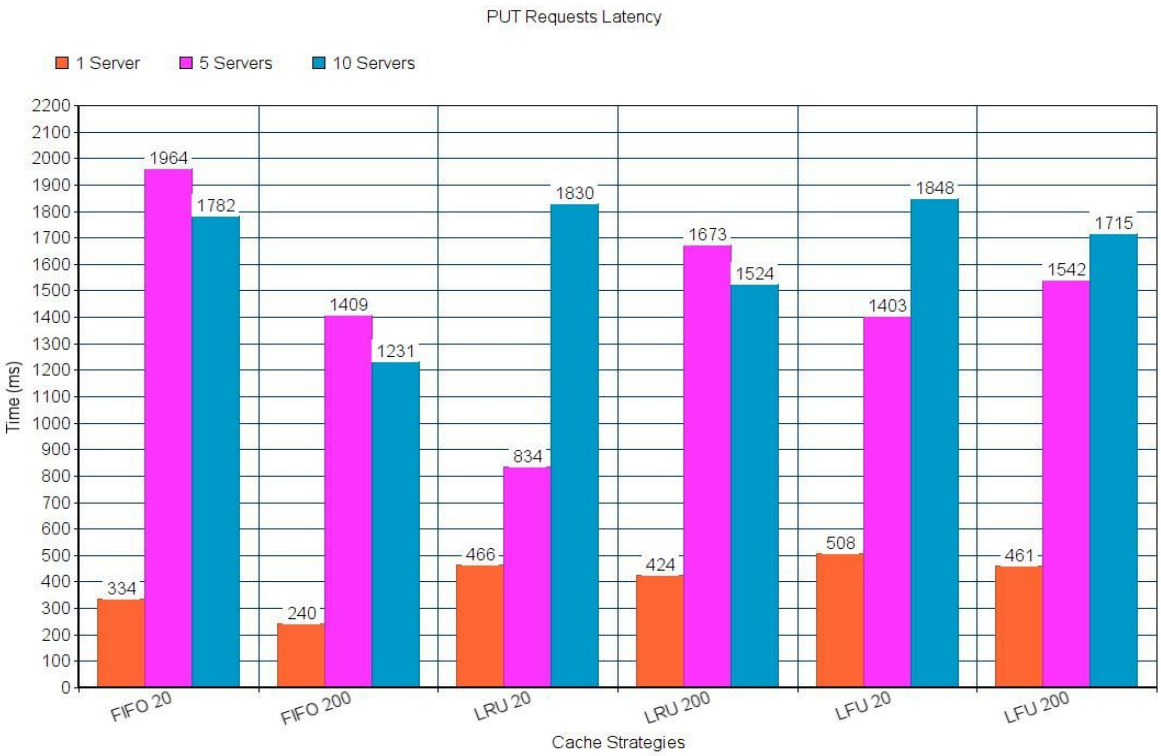
Two important metrics, namely latency of the requests and throughput (requests served per second) were calculated and printed out into 2 CSV files (one each for GET and PUT). The files were then used to create the following plots.

Plot 1: Latency of GET Requests:

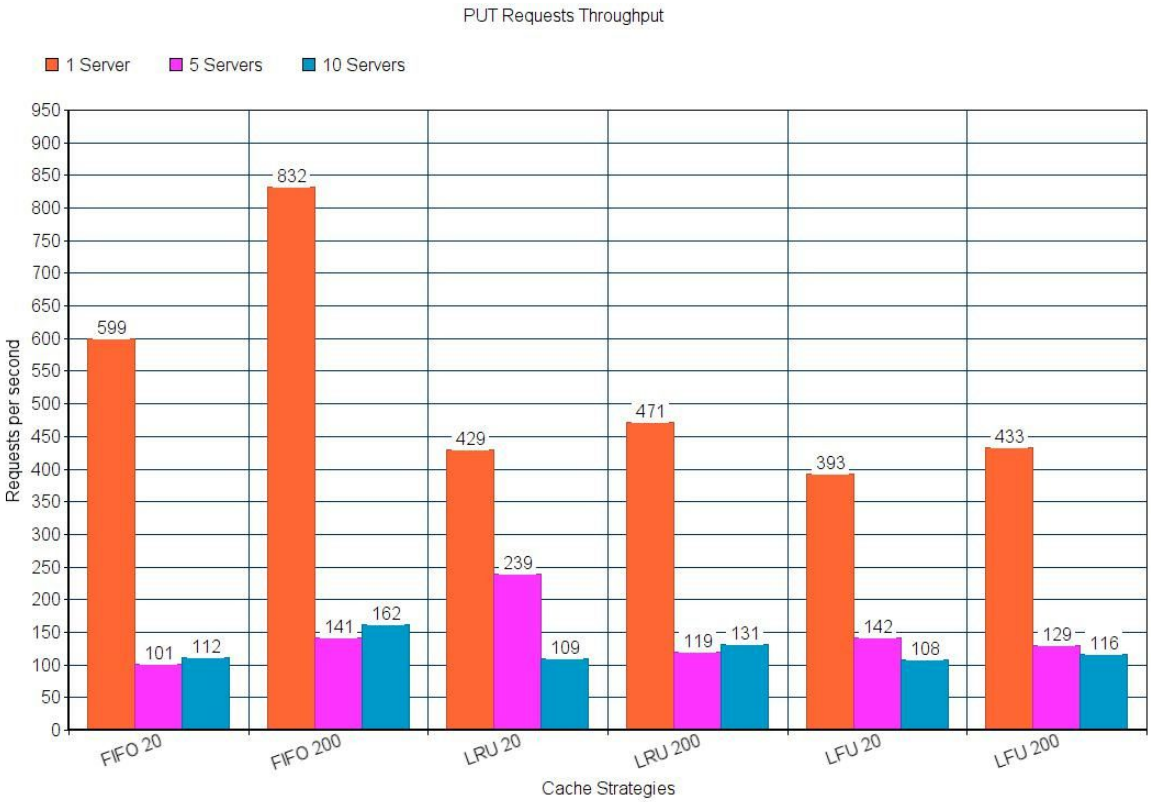
Plot 2: Throughput of GET requests:



Plot 3: Latency of PUT requests:



Plot 4: Throughput of PUT requests:



Observations

- One general inference from the plots is that, the performance of the system goes down (latency goes up, throughput comes down) once the number of servers are increased irrespective of the cache strategy. At first notice this can seem to be counter intuitive. However, it may be because of the following reasons:
  - As all the servers and clients are individual threads inside a single JVM, increasing the number of threads can result in a performance hit.
  - Since the keys that are queried are randomly generated by the test suite, the probability of creating a new connection for every request increases as the number of servers increases. This is a costly operation and can result in a lag in performance.
- For a single server configuration the best latency and throughput numbers are given by the FIFO cache strategy. This is because both LRU and LFU strategies works well only when the keys follow a particular pattern. Here, since keys are completely random FIFO is the best suited strategy.
- For FIFO caches, highest performance is obtained when it has a larger cache size. This is logical, since this would mean more requests getting served from the cache than from the disk.