CS667 Distributed Operating Systems

Spring 2019

Lab 3 Performance & Evaluation Document

Team Name: STIMA

Name: Aggrey Muhebwa
Github Handle: amuhebwa
Student ID: 32055729

Name: Zeal Shah
Github Handle: zealshah95
Student ID: 31737150

1 Experimentation Summary

Three different kinds of experiments were conducted:

1. Average response time measurement
2. Overhead and latencies of cache consistency operations
3. Latencies of fault tolerance operations

2 Average Response Times

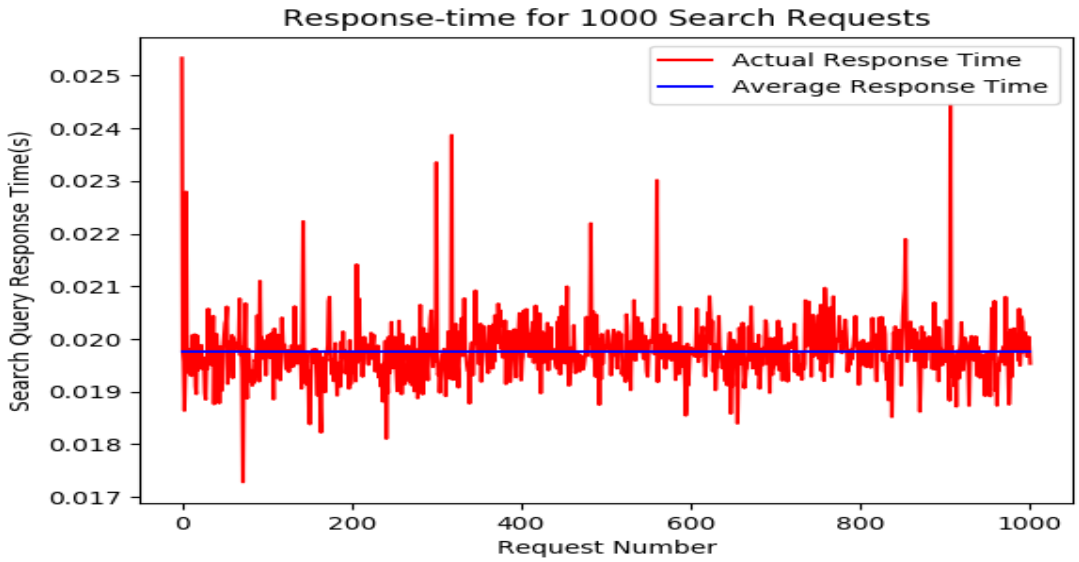
2.1 Sequential Requests- Search, Lookup, Buy

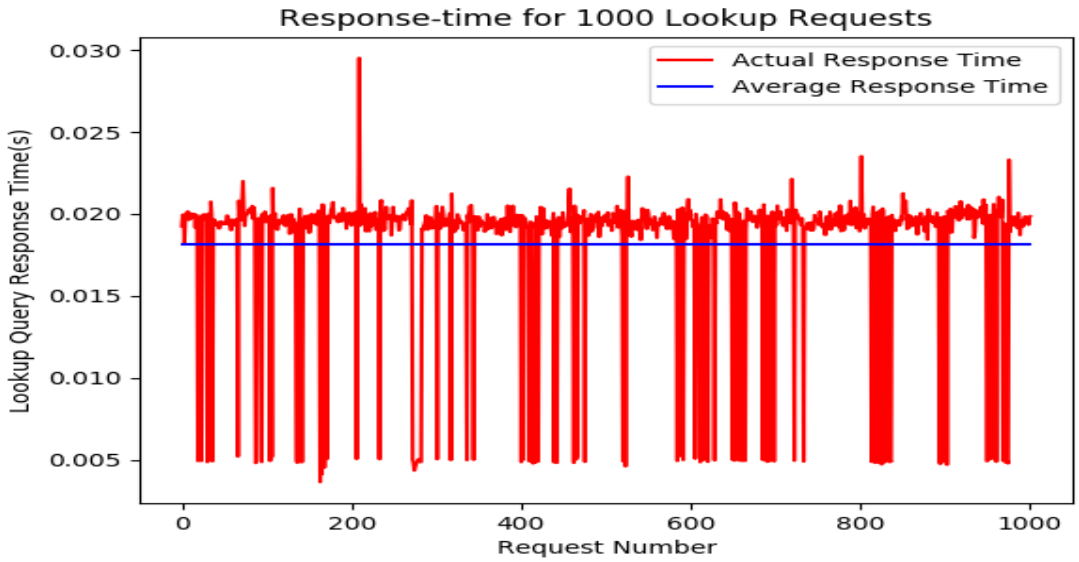
2.1.1 Summary

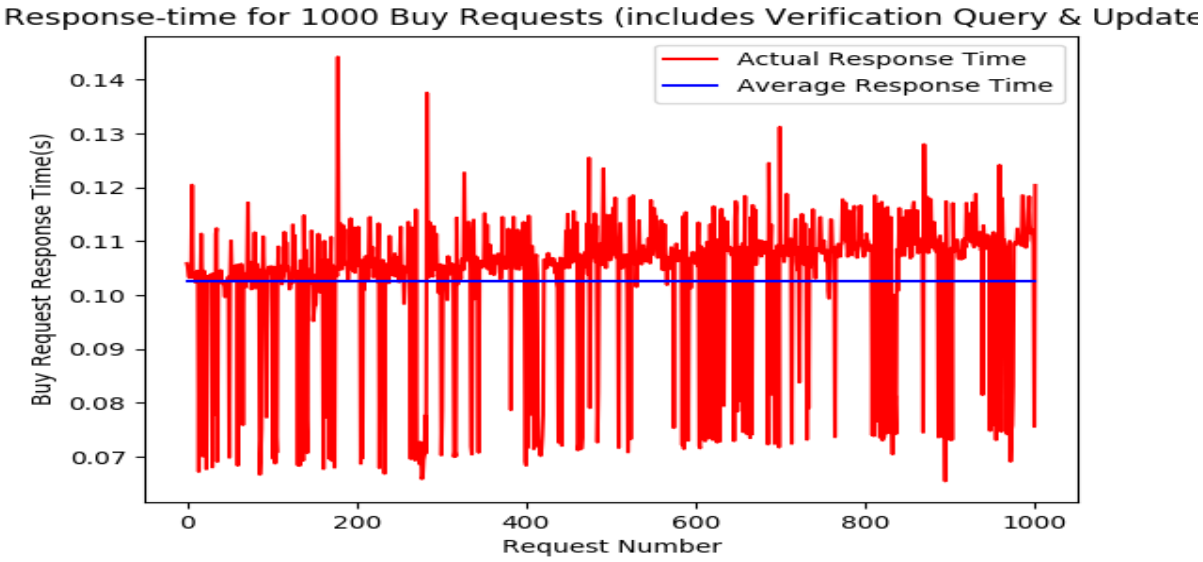
- *How to run?* sequential_requests_times.py file should be run as a client.
- Client sends 1000 sequential- search, lookup, and buy requests to the system.
- We measure the response time per client search, lookup, and buy request. We also measure the end-to-end response time which covers search, lookup, and buy response times.
- Note that in this experiment, the client tries to buy the very item it sends the lookup request for. This means that when a new lookup request arrives, the data is cached, but then the same item is requested to be bought which invalidates the cache. So, this experiment doesn't necessarily give us the effect of caching on performance. In order to observe the effect of caching on performance of lookup queries, please check the next subsection.

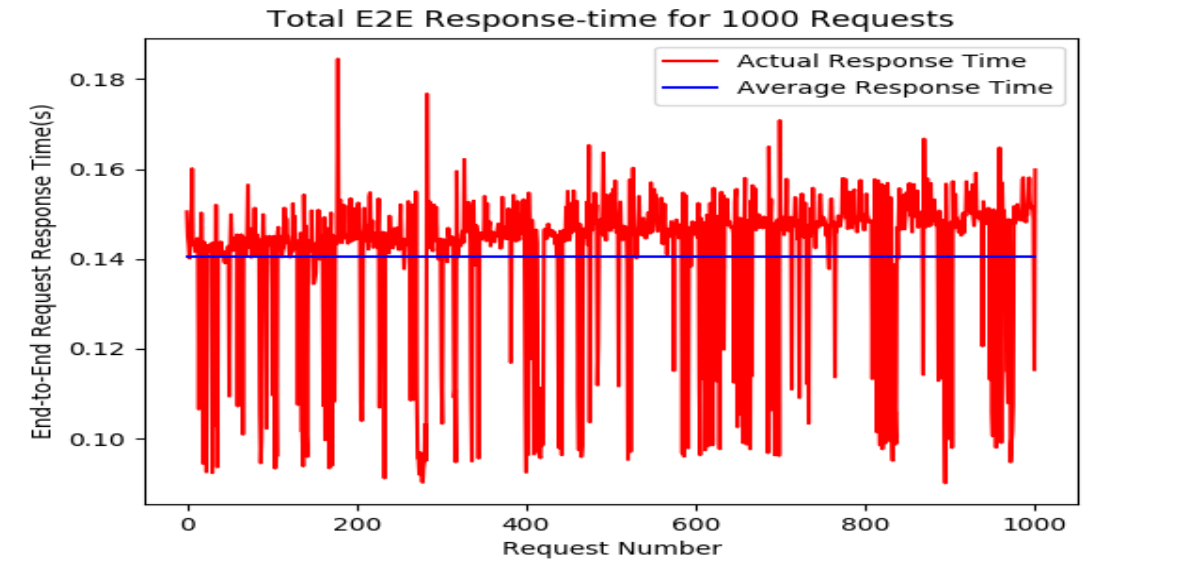
Experiment	**Search**	**Look-up**	**Buy**	**E2E**
1000 Sequential Requests w/ Caching	0.019	0.018	0.103	0.141
1000 Sequential Requests w/o Caching	0.019	0.020	0.187	0.200

2.1.2 Plots for Experiments With Caching

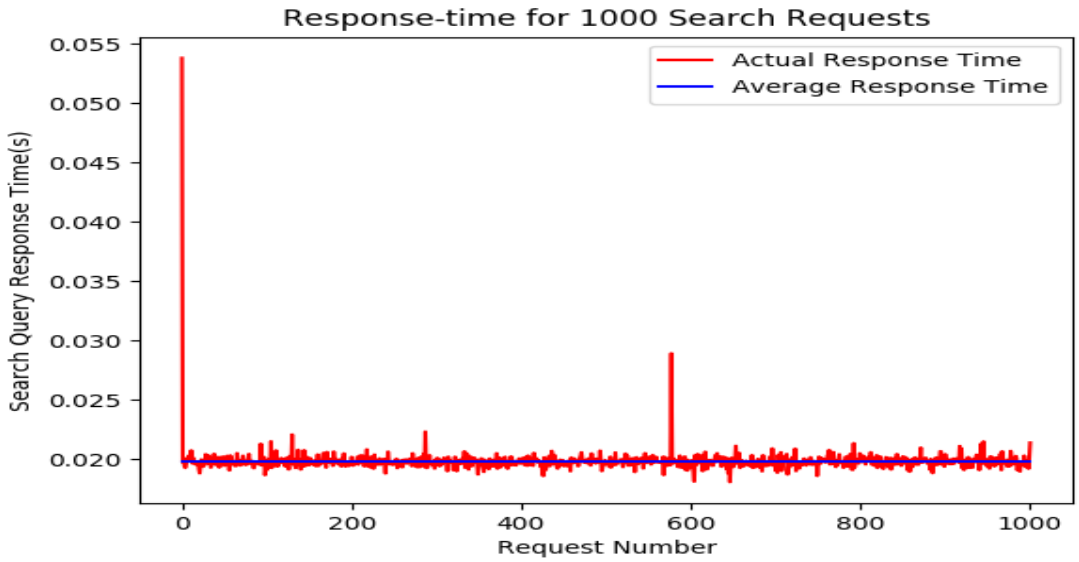


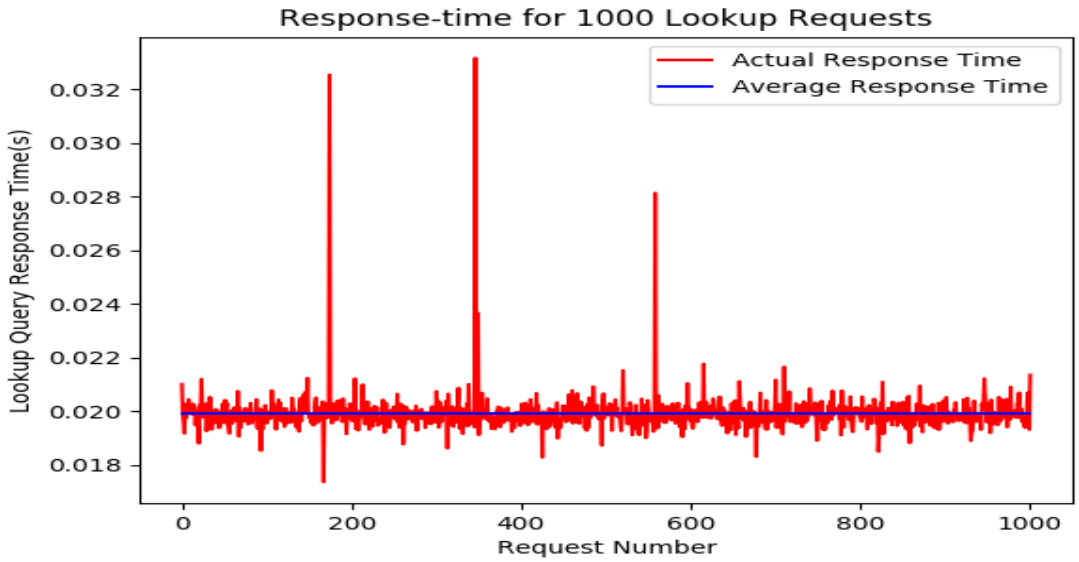


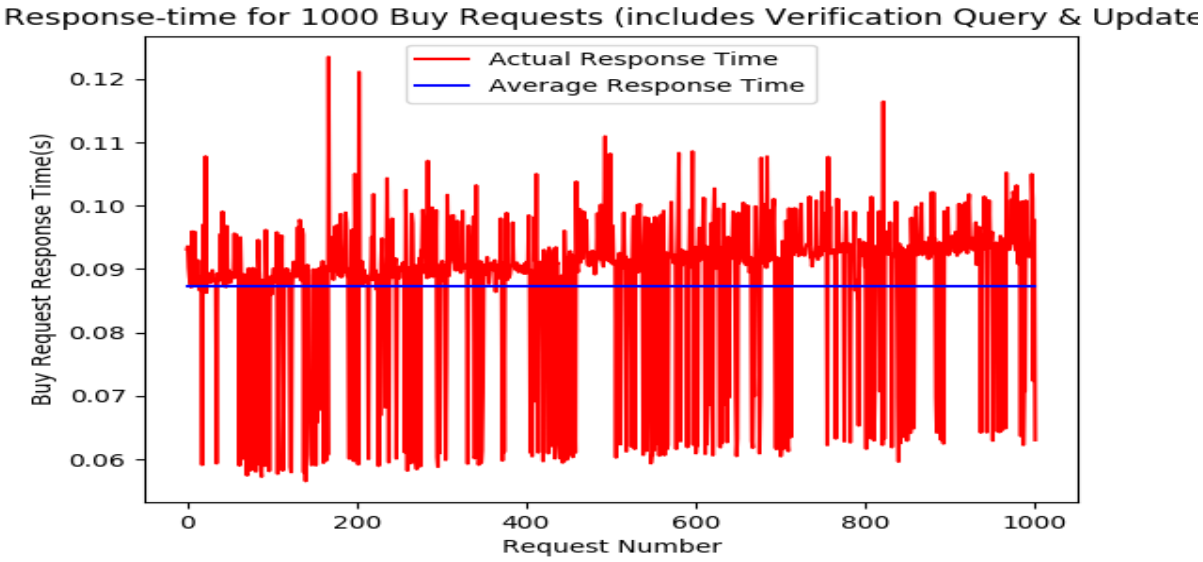

Response-time for 1000 Buy Requests (includes Verification Query & Update
Actual Response Time
Average Response Time
Buy Request Response Time(s)
0.14
0.13
0.12
0.11
0.10
0.09
0.08
0.07
0
200
400
600
800
1000
Request Number

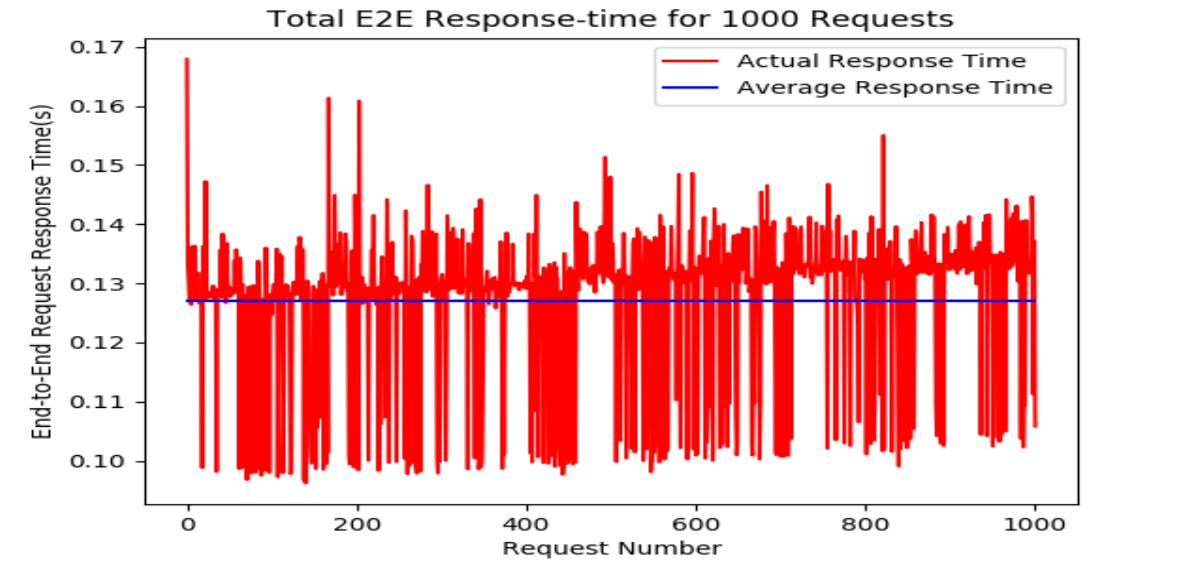

Total E2E Response-time for 1000 Requests
Actual Response Time
Average Response Time
End-to-End Request Response Time(s)
0.18
0.16
0.14
0.12
0.10
0
200
400
600
800
1000
Request Number

2.1.3 Plots for Experiments Without Caching









2.2 Sequential Requests- Only Lookup

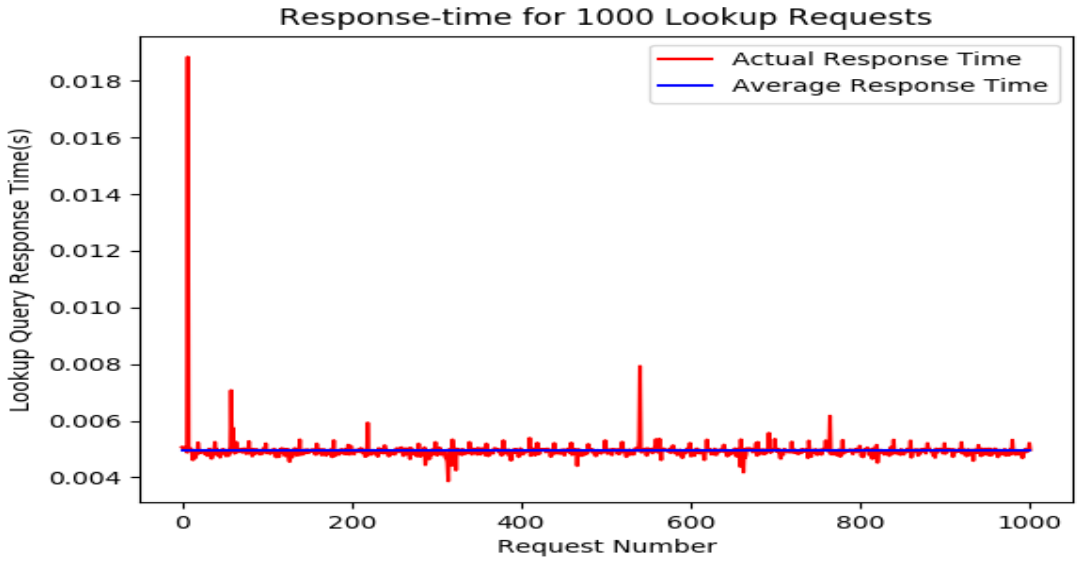
2.2.1 Summary

- *How to run?* cache_latency_lookup.py file should be run as a client.
- Client sends 1000 sequential- lookup requests to the system.
- We measure the response time per client lookup request.
- This experiment is aimed at demonstrating the effect of cache so it was conducted in two phases- without cache and with cache.
- NOTE: If the client file output gets stuck, please restart all the servers and try running the file again.

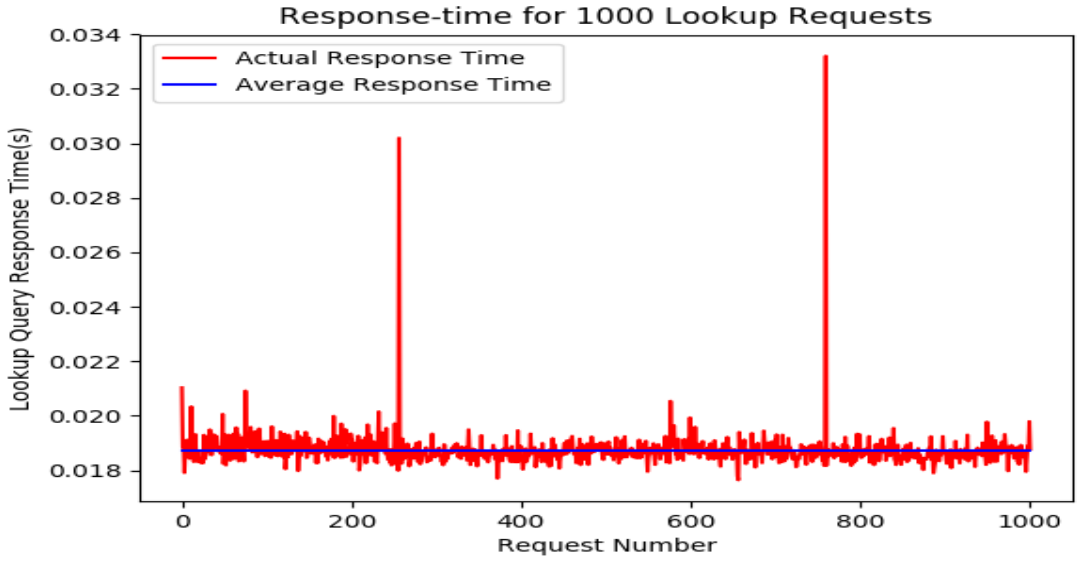
Parameter	Time (s)
Average response time without cache	0.0187
Average response time with cache	0.00494
Response time difference due to cache	0.01376

Response by cached system is 13.8 ms faster than the non-cached one. This shows that system's handling of lookup queries gets faster when data is cached is present in the front end server. Graphs below show the lookup response time measured versus request number for with cache and without cache scenarios.

2.2.2 Plots for Experiment With Caching



2.2.3 Plots for Experiment Without Caching



2.3 Concurrent Requests- Search, Lookup, Buy

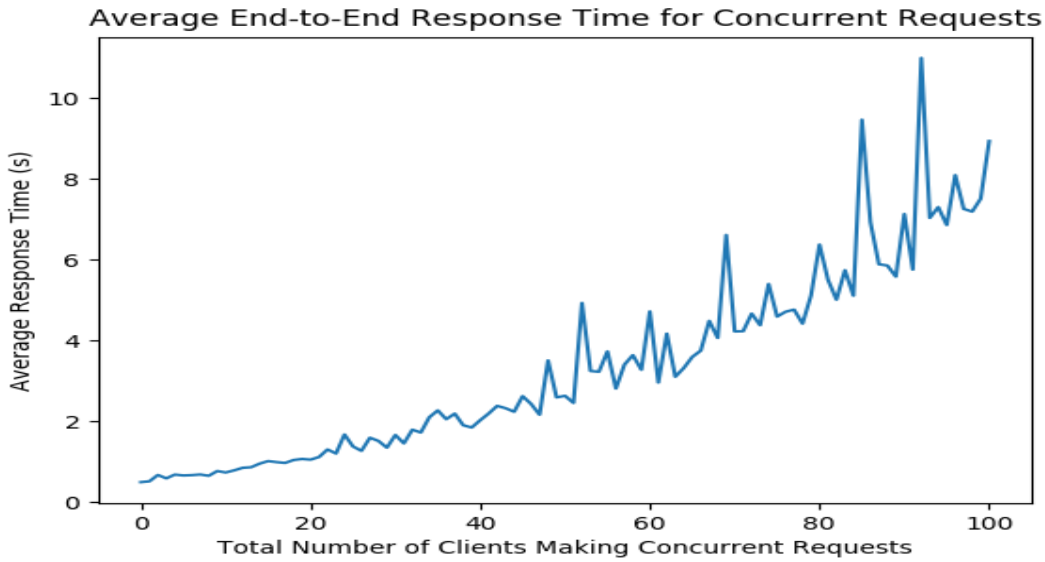
2.3.1 Summary

- *How to run?* concurrent_requests.py file should be run as a client.
- Multiple clients make concurrent requests to the system. We increase the number of clients making concurrent requests from 1 to 100.
- We measure the end-to-end response time observed by every group of clients. So basically, we vary the number of clients making concurrent requests and observer how the response time changes.
- Note that in this experiment, the client tries to buy the very item it sends the lookup request for. This means that when a new lookup request arrives, the data is cached, but then the same item is requested to be bought which invalidates the cache. So, this experiment doesn't necessarily give us the effect of caching on performance. This experiment is aimed at analyzing the overall performance of the system for handling concurrent requests.

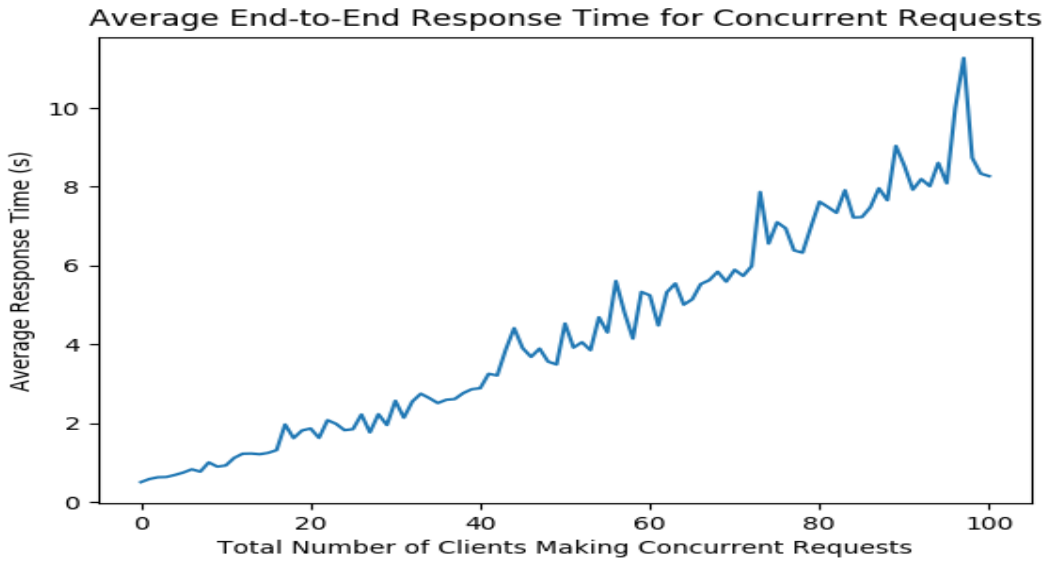
The following table gives end-to-end response times (in seconds) observed by varying the number of clients sending concurrent requests. Number of clients sending concurrent requests in the actual experiment went from 1 to 100. But in the following table we summarize the results for number of clients going from 10 to 100 in steps of 10. Each column represents how many clients were sending concurrent requests. Each row represents a scenario- S1 is with caching, and S2 is without caching.

	10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
S1	0.50	0.74	1.06	1.67	2.03	2.64	4.73	4.23	6.38	7.14
S2	0.50	0.93	1.86	2.57	2.88	4.53	5.24	5.89	7.62	8.54

2.3.2 Plot for Experiment With Caching



2.3.3 Plot for Experiment Without Caching



3 Cache Consistency Operations

3.1 100 sequential lookup requests only

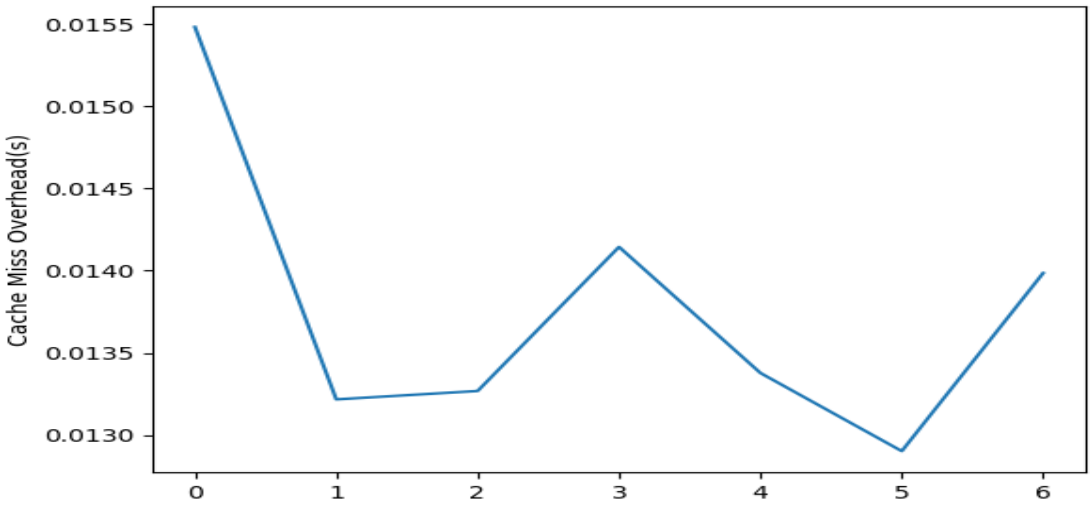
3.1.1 Summary

- *How to run?* cache_latency_lookup.py file should be run as a client
- This experiment is similar to the sequential lookup requests experiment which is discussed in the above section. But, in this experiment the client sends only 20 sequential lookup requests and we record the response time for each request.
- We look at this experiment from the point-of-view of cache miss latency and try to address the question- "What is the latency due to a cache miss?"
- If the response data for lookup request is found in the front end server cache, it is immediately sent over to the client. But if the data is not present in cache, front end server forwards the request to back end server and then the response received is forwarded to the client. The second case is definitely time consuming. So, in short, response time is less if data is found in cache and response time is more if data is not found in cache. This can be seen in the plot below. The plot 2 contains 7 peaks where each peak represents a cache miss for one unique book. As soon as all the books have been requested for at least once, the cache will contain details associated with all the available books then. This means that any lookup request from this point on-wards will be served by front end server's cache which is much faster.
- All the peaks in the plot 2 represent response time on cache misses. All the non-peak values represent response time on cache hits.
- In order to be able to visualize cache miss latency in a better manner, plot 1 shows the miss time for every miss. It has exactly 7 points i.e. one for each book.

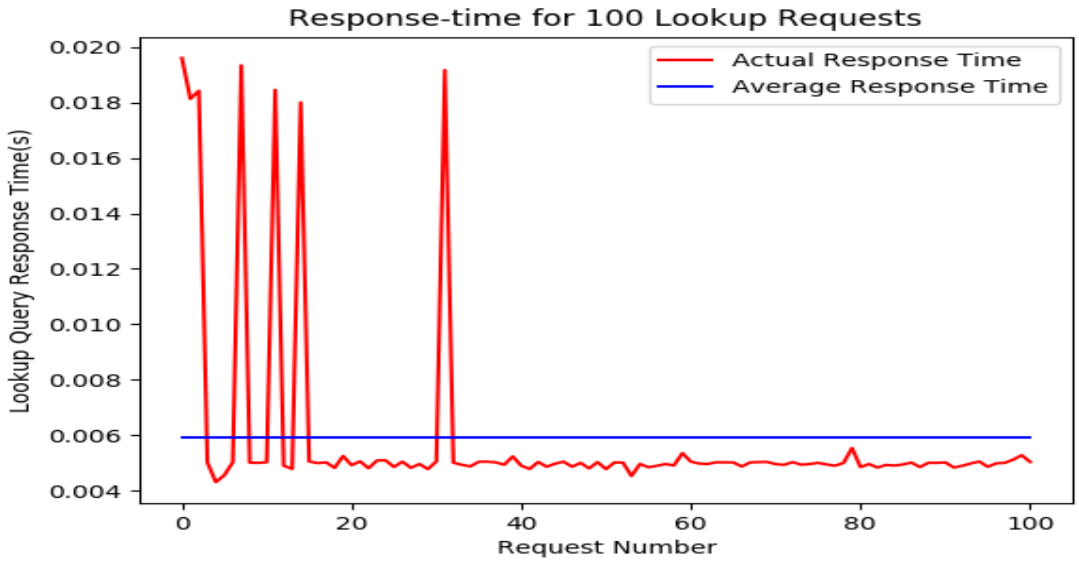
- Please note that this experiment is not meant to demonstrate the effect of cache invalidation and so we have intentionally made the client to just send lookup requests and no buy requests. Cache invalidation latency is discussed in the next subsection.

Parameter	Time (ms)
Average cache miss time	5.9136

3.1.2 (Plot1) Cache Miss Latency Plot



3.1.3 (Plot 2) Lookup Response Time on Cache Hits/Misses



3.2 100 sequential buy requests

3.2.1 Summary

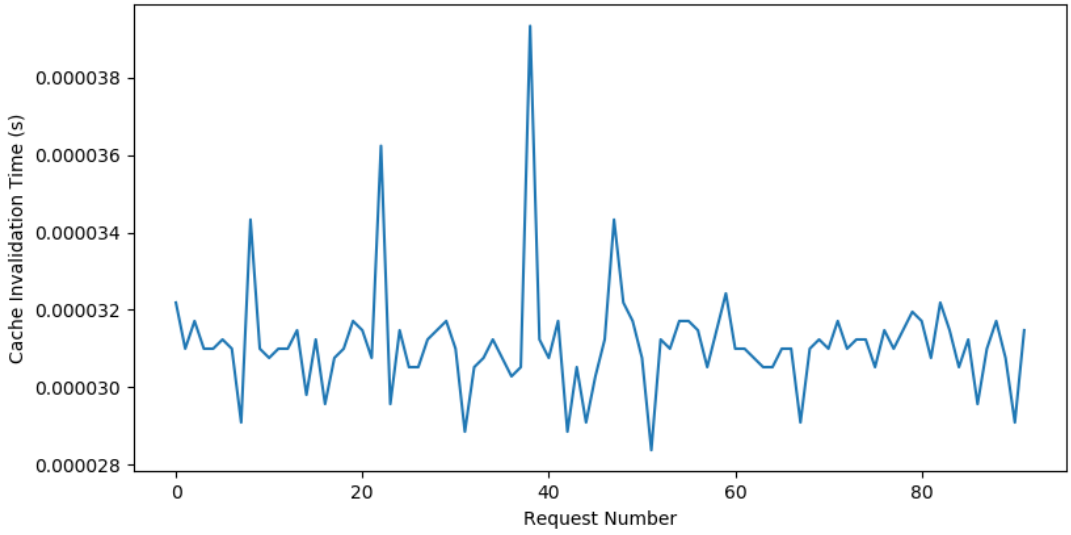
- *How to run?* cache_invalidation_overhead_exp.py should be run as a client.
- Client sends 20 sequential lookup and buy requests.
- Every buy request entails cache invalidation and so in this experiment we try to address the question- "What are the overhead of cache invalidation operations?"

Parameter	Time (microseconds)
Average cache invalidation time	31.13
Average E2E response time	118.86

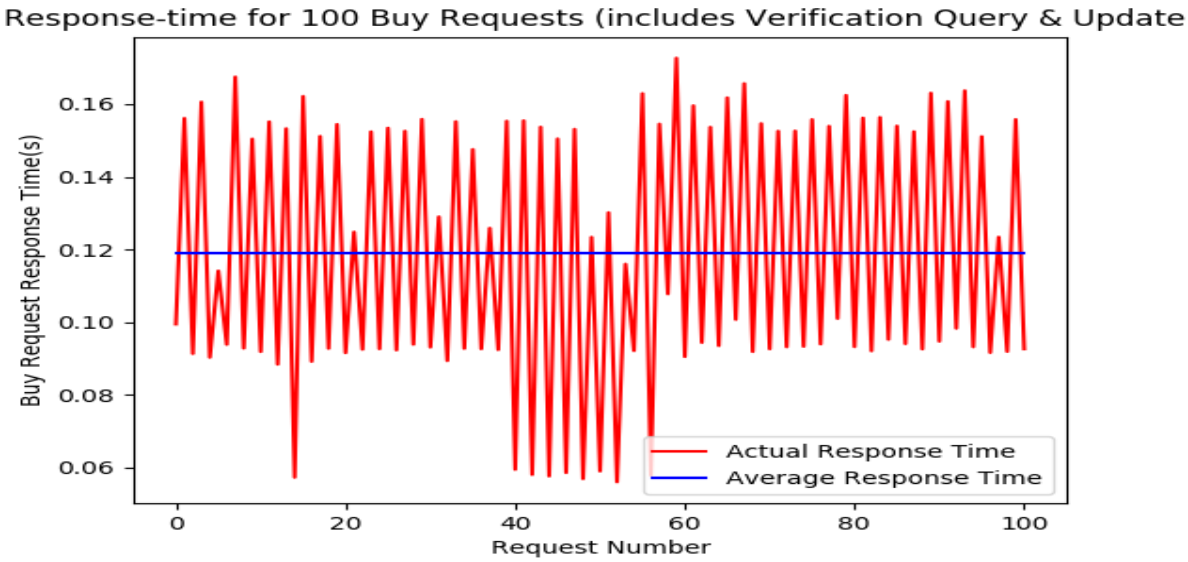
Thus, overhead due to cache invalidation is 31.13 microseconds which comprises of 26.19% of average end to end response time.

3.2.2 Cache Invalidation Time vs. Request Number

Graph 1: Cache invalidation time per request versus request number.



3.2.3 E2E Response Time vs. Request Number



4 Fault Tolerance Operations

4.1 Failure of Catalog Replica

This set of experiments if focused on failure of catalog replica and the latencies and overhead associated with it.

4.1.1 Overhead of detecting a failure and switching over

- *How to run?* Run client.py file as a client.
- While performing this experiment, we introduced a delay of 10 seconds in catalog replica's query_by_item() interface and query_by_subject(). This delay gives us enough time to crash a replica in order to measure the performance of our system.
- Replica is crashed once the client sends in a search/lookup request. We record the time it takes to detect the failure and also the time to switch from one replica to another.
- Results are given in the table present at the end of this section.

4.1.2 Time to recover and re-synchronize

- We run sequential_requests_times.py file where client tries to send 100 requests to the system.
- We crash one catalog server manually and wait for 10 seconds during which only the non-faulty one will be serving the requests.
- We restart the catalog server. As soon as it is restarted, it tries to re-synchronize it's database with the non-faulty replica.

- In this experiment, we measure the time taken by the faulty replica to re-synchronize it's database and come online i.e. to fully recover.
- Results are given in the table present at the end of this section.

Experiment	**Time Taken(ms)**
Fault detection and Switch-over	4.98
Recover and re-synchronize	535.3

4.2 Failure of Order Replica

This set of experiments if focused on failure of order replica and the latencies and overhead associated with it.

4.2.1 Overhead of detecting a failure and switching over

- *How to run?* Run client.py file as a client.
- While performing this experiment, we introduced a delay of 10 seconds in order replica's buy_request() interface. This delay gives us enough time to crash an order replica to be able to measure the performance of our system.
- Replica is crashed once client's buy request reaches one of the buy replicas. We record the time it takes to detect the failure and also the time to switch from one replica to another.
- Results are given in the table present at the end of this section.

4.2.2 Time to recover and re-synchronize

- We run sequential_requests_times.py file where client tries to send 100 requests to the system.
- We crash one order server manually and wait for 10 seconds during which only the non-faulty order replica will be serving the requests.
- We restart the faulty order server. As soon as it is restarted, it tries to re-synchronize it's database with the non-faulty replica.
- In this experiment, we measure the time taken by the faulty replica to re-synchronize it's database and come online i.e. to fully recover.
- Results are given in the table present at the end of this section.

Experiment	**Time Taken(ms)**
Fault detection and Switch-over	9.33
Recover and re-synchronize	1073.9