Advanced Systems Lab Report

Autumn Semester 2018

Name: Stefano Peverelli
Legi: 19-980-396

Grading

Section	Points
1	
2	
3	
4	
5	
6	
7	
Total	

1 System Overview (75 pts)

Following is a description of the class structure of the Middleware:

- `MW` - responsible for accepting incoming socket connections, instantiating the `Writer` threads, and for enqueuing each request.
- `Request` - this class represent a request object, it contains the request's type and measures (presented below).
- `Worker` - each `Worker` establishes a socket connection with the memchached servers, dequeues a request, performs load balancing, and process the request (sends to servers, handle responses and collects some statistics.
- `Statistic` - container for all measured statistics.
- `Writer` - when shutting down the Middleware, it collects all the statistics from each `Worker`, aggregates them and save them to disk.

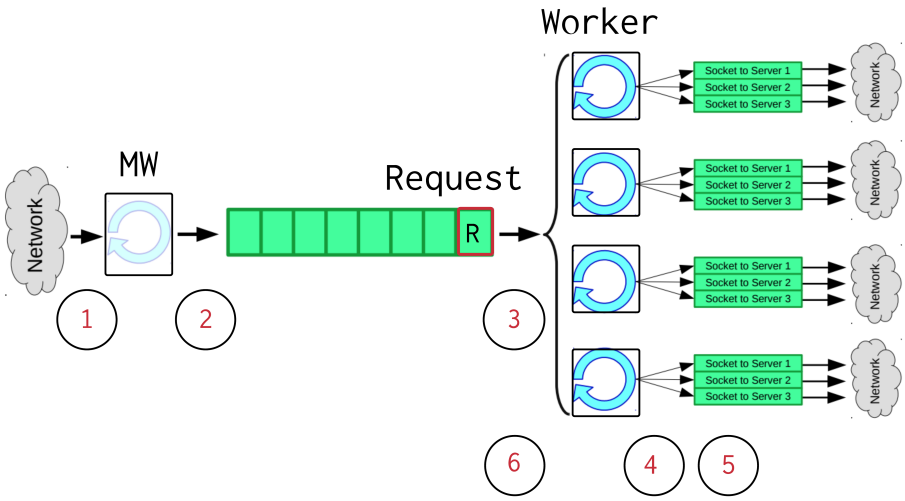

Figure 1: Middleware architecture.

The Middleware is instrumented at six points (shown in Fig.1):

1. R_c - Request created
2. R_e - Request enqueued
3. R_d - Request dequeued
4. R_f - Request forwarded to memcached instances.
5. R_r - Response received from memcached instances.
6. R_a - Request answered to memtier instance.

1.1 Load Balancing

In order to guarantee that each server gets the same amount of jobs, each request's key gets hashed to a specific index that identify a memcached instance.
This is done by method `getServerFromKey` in `Worker` class, and further tested in class `LoadBalancer` of package `test.java.asl`. Following is the result of a test with 1M random strings and 10 servers:

```
Server 0 got 100047 jobs.
Server 1 got 100172 jobs.
Server 2 got 99590 jobs.
Server 3 got 99714 jobs.
Server 4 got 100506 jobs.
Server 5 got 100081 jobs.
Server 6 got 99962 jobs.
Server 7 got 100221 jobs.
Server 8 got 99809 jobs.
Server 9 got 99898 jobs.
```

This indeed shows that the distribution is uniform and that each memcached instance receives the same amount of requests as the others.
NOTE: This is done only for `GET` and *non-sharded* `MULTI-GET`s requests, as `SET`s need to be replicated, and *sharded* `MULTI-GET`s are splitted across memcached instances.

1.2 The system

There are two main components, the `MW` and the `Worker`s. They communicate between each other using a dynamic-sized queue where requests are passed in.

1.3 The queue

The queue is designed to grow as much as needed although in practice it can only grow to the number of clients memtier is using. A fixed-sized queue may have done the job as well, but having a dynamic-sized one has less impact on the memory usage.

1.4 Non-blocking IO

The Middleware communicate both with the clients and the memcached instances via the `java.NIO.SocketChannel`'s library. In `MW`'s constructor `Worker`'s are instantiated and started. Each `Worker`, connects to the memcached instances in a non-blocking fashion.

1.5 Request Protocol

Each request is assumed to be well-formed, and only the first character is checked in order to determine the request type. As a request may be sent into multiple chunks, it is essential to read it without losing any byte. Thisis done by saving the partial content of the `ByteBuffer` the `SocketChannel` has written to, into a `ByteArrayOutputStream`. A request is assumed to be completed when the last bytes are equals to`"\r\n"`. The same assumption is done for responses from the memcached instances, by checking `"STORED\r\n"` or `"END\r\n"`.

1.6 Handling incoming connections

The Middleware listens for incoming connections by memtier clients. This is achieved by using the `java.NIO` package that allows non-blocking IO operations on multiple channels. A `Selector` monitors channels for changes and signal them in a `SelectionKey` object, which contains a set of keys registered with the channel.
Whenever a `SelectionKey`'s interest is set on `ACCEPT`, a `SocketChannel` connection can be established between the client issuing the request and the Middleware. After that, the interest of the `SelectionKey` is set to `READ`, waiting for data from the client.

1.7 Handling incoming requests

When the `SelectionKey`'s interest is set to `READ`, data is read from the `SocketChannel`. From this moment the Middleware starts recording the `responseTime` of the request (Point 1 in Fig.1). When the whole request has been read, it gets enqueued to a `BlockingQueue`, its `queueWaitingTime` is started (Point 2 in Fig.1), and the `SelectionKey`'s interest is set to `WRITE`.

1.8 Forwarding requests

When a new request is ready to be processed by a `Worker`, before sending it to the memcached instances, some operations take place (shown in Fig.1.8):

- The request's `queueWaitingTime` is stopped (Point 3 in Fig.1).
- The request gets copied into a `pendingRequest` object.
- Based on its type, a `multiRequest` object gets created.

Then, when the first entry of the `multiRequest` object gets sent, the `pendingRequest`'s `serviceTime` is started (Point 4 in Fig.1).

1.9 Handling responses

When handling an incoming response each `Worker` does the following:

- Checks if the response is completed
- Increments a counter of the number of responses received, and compares it with the size of the `multiRequest` object created for that `pendingRequest` (expected number of responses).
- Then, in case it has received the expected number of responses:
  - Stops the `pendingRequest`'s `serviceTime` (Point 5 in Fig.1).
  - Creates a `Statistic` object that wraps `pendingRequest` measures.
  - Answers back to the client that issued the request.
  - Stops the `pendingRequests` `responseTime` (Point 6 in Fig.1).

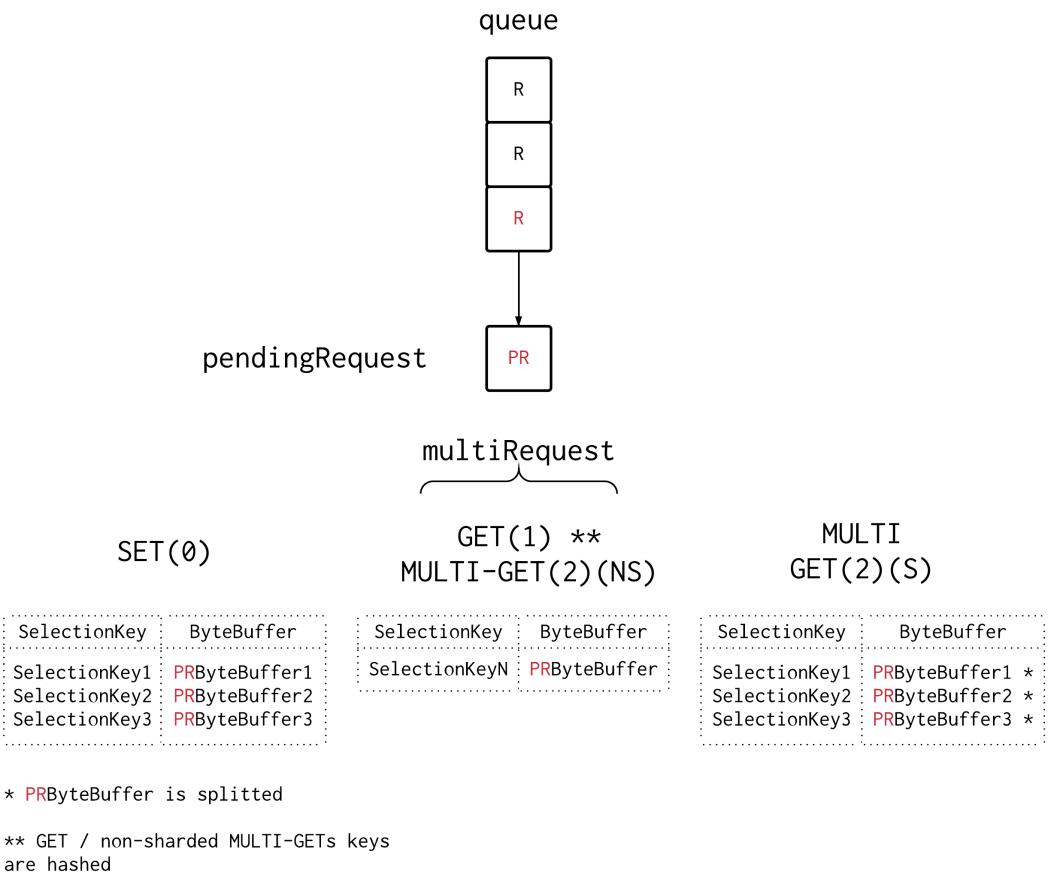

Figure 2: `loadRequest()` behavior.

1.10 Statistics

As already pointed out in 1.9, each `Worker` collects the following statistics:

- The current system's time (`ns`).
- The time the request spent in the queue (`ns`).
- The time elapsed between forwarding the first `multiRequest` and the last response (`ns`).
- The time elapsed between reading the request from the client and the last response (`ns`).
- The current queue size.
- The number of misses of the request.
- The number of keys in the request.

NOTE: All the time-related statistics are then converted in milliseconds (`ms`) when writing them to the logs.

2 Baseline without Middleware (75 pts)

In this experiments you study the performance characteristics of the memtier clients and memcached servers.

2.1 One Server

Both, for a read-only and write-only workload plot the throughput and the response time as a function of NumClients. All clients are connected to a single memcached instance.

Use 3 load generating VMs, with one memtier (CT=2) each, and vary the number of virtual clients (VC) per memtier thread between 1 and 32. Show how the behavior of the server changes as we add more clients.

Number of servers	1
Number of client machines	3
Instances of memtier per machine	1
Threads per memtier instance	2
Virtual clients per thread	[1..32]
Workload	Write-only and Read-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	N/A
Worker threads per middleware	N/A
Repetitions	3 or more

2.1.1 Explanation

Describe in which phase the memcached servers are under-saturated, saturated, or over-saturated. Describe how throughput and response time correlate. Explain what further conclusions can be drawn from the experiment.

2.2 Two Servers

For a read-only and write-only workload plot throughput and response time as a function of NumClients. The clients are connected to two memcached instances.

Use 1 load generating VM, with one memtier (CT=1) connected to each memcached instance (two memcache instances in total), and vary the number of virtual clients (VC) per memtier thread between 1 and 32. Show how the behavior of the server changes and explain what conclusions we can draw from this experiment.

Number of servers	2
Number of client machines	1
Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1..32]
Workload	Write-only and Read-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	N/A
Worker threads per middleware	N/A
Repetitions	3 or more (at least 1 minute each)

2.2.1 Explanation

Describe how this experiment compares to the previous section. Which results are the same and which ones differ? Explain what further conclusions can be drawn from the experiment.

2.3 Summary

Based on the experiments above, fill out the following table:

Maximum throughput of different VMs.

	Read-only workload	Write-only workload	Configuration gives max. throughput
One memcached server			
One load generating VM			

Write at least two paragraphs about how both results relate. Describe what is the bottleneck of this setup is. If the maximum throughput for both experiments is the same, explain why. If it is not the case, explain why not. Write down key take-away messages about the behaviour of the memtier clients and the memcached servers.

3 Baseline with Middleware (90 pts)

In this set of experiments, you will have to use three load generator VMs and 1 memcached server, measuring how the throughput of the system changes when increasing the number of clients. Scaling virtual clients inside memtier has to be done as explained in the previous sections. Plot both throughput and response time as measured on the middleware.

3.1 One Middleware

Connect three load generator machines (one instance of memtier with CT=2) to a single middleware and use 1 memcached server. Run a read-only and a write-only workload with increasing number of clients (between 2 and 64) and measure response time *both at the client and at the middleware*, and plot the throughput and response time measured in the middleware.

Repeat this experiment for different number of worker threads inside the middleware: 8, 16, 32, 64.

Number of servers	1
Number of client machines	3
Instances of memtier per machine	1
Threads per memtier instance	2
Virtual clients per thread	[1..32]
Workload	Write-only and Read-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	1
Worker threads per middleware	[8..64]
Repetitions	3 or more (at least 1 minute each)

3.1.1 Explanation

Provide a detailed analysis of the results (e.g., bottleneck analysis, component utilizations, average queue lengths, system saturation). Add any additional figures and experiments that help you illustrate your point and support your claims.

3.2 Two Middlewares

Connect three load generator machines (two instances of memtier with CT=1) to two middlewares and use 1 memcached server. Run a read-only and a write-only workload with increasing

number of clients (between 2 and 64) and measure response time *both at the client and at the middleware*, and plot the throughput and response time as measured in the middleware.

Repeat this experiment for different number of worker threads inside the middleware: 8, 16, 32, 64.

Number of servers	1
Number of client machines	3
Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1..32]
Workload	Write-only and Read-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	2
Worker threads per middleware	[8..64]
Repetitions	3 or more (at least 1 minute each)

3.2.1 Explanation

Provide a detailed analysis of the results (e.g., bottleneck analysis, component utilizations, average queue lengths, system saturation). Add any additional figures and experiments that help you illustrate your point and support your claims.

3.3 Summary

Based on the experiments above, fill out the following table. For both of them use the numbers from a single experiment to fill out all lines. Miss rate represents the percentage of GET requests that return no data. Time in the queue refers to the time spent in the queue between the net-thread and the worker threads.

Maximum throughput for one middleware.

	Throughput	Response time	Average time in queue	Miss rate
Reads: Measured on middleware				
Reads: Measured on clients			n/a	
Writes: Measured on middleware				n/a
Writes: Measured on clients			n/a	n/a

Maximum throughput for two middlewares.

	Throughput	Response time	Average time in queue	Miss rate
Reads: Measured on middleware				
Reads: Measured on clients			n/a	
Writes: Measured on middleware				n/a
Writes: Measured on clients			n/a	n/a

Based on the data provided in these tables, write at least two paragraphs summarizing your findings about the performance of the middleware in the baseline experiments.

4 Throughput for Writes (90 pts)

4.1 Full System

Connect three load generating VMs to two middlewares and three memchached servers. Run a write-only experiment. You need to plot throughput and response time measured on the middleware as a function of number of clients. The measurements have to be performed for 8, 16, 32 and 64 worker threads inside each middleware.

Number of servers	3
Number of client machines	3
Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1..32]
Workload	Write-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	2
Worker threads per middleware	[8..64]
Repetitions	3 or more (at least 1 minute each)

4.1.1 Explanation

Provide a detailed analysis of the results (e.g., bottleneck analysis, component utilizations, average queue lengths, system saturation). Add any additional figures and experiments that help you illustrate your point and support your claims.

4.2 Summary

Based on the experiments above, fill out the following table with the data corresponding to the maximum throughput point for all four worker-thread scenarios.

Maximum throughput for the full system

	WT=8	WT=16	WT=32	WT=64
Throughput (Middleware)				
Throughput (Derived from MW response time)				
Throughput (Client)				
Average time in queue				
Average length of queue				
Average time waiting for memcached				

Based on the data provided in these tables, draw conclusions on the state of your system for a variable number of worker threads.

5 Gets and Multi-gets (90 pts)

For this set of experiments you will use three load generating machines, two middlewares and three memcached servers. Each memtier instance should have 2 virtual clients in total and the number of middleware worker threads is 64, or the one that provides the highest throughput in your system (whichever number of threads is smaller).

For multi-GET workloads, use the `--ratio` parameter to specify the exact ratio between SETs and GETs. You will have to measure response time on the client as a function of multi-get size, with and without sharding on the middlewares.

5.1 Sharded Case

Run multi-gets with 1, 3, 6 and 9 keys (memtier configuration) with sharding enabled (multi-gets are broken up into smaller multi-gets and spread across servers). Plot average response time as measured on the client, as well as the 25th, 50th, 75th, 90th and 99th percentiles.

Number of servers	3
Number of client machines	3
Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	2
Workload	ratio=1:<Multi-Get size>
Multi-Get behavior	Sharded
Multi-Get size	[1..9]
Number of middlewares	2
Worker threads per middleware	max. throughput config.
Repetitions	3 or more (at least 1 minute each)

5.1.1 Explanation

Provide a detailed analysis of the results (e.g., bottleneck analysis, component utilizations, average queue lengths, system saturation). Add any additional figures and experiments that help you illustrate your point and support your claims.

5.2 Non-sharded Case

Run multi-gets with 1, 3, 6 and 9 keys (memtier configuration) with sharding disabled. Plot average response time as measured on the client, as well as the 25th, 50th, 75th, 90th and 99th percentiles.

Number of servers	3
Number of client machines	3
Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	2
Workload	ratio=1:<Multi-Get size>
Multi-Get behavior	Non-Sharded
Multi-Get size	[1..9]
Number of middlewares	2
Worker threads per middleware	max. throughput config.
Repetitions	3 or more (at least 1 minute each)

5.2.1 Explanation

Provide a detailed analysis of the results (e.g., bottleneck analysis, component utilizations, average queue lengths, system saturation). Add any additional figures and experiments that help you illustrate your point and support your claims.

5.3 Histogram

For the case with 6 keys inside the multi-get, display four histograms representing the sharded and non-sharded response time distribution, both as measured on the client, and inside the middleware. Choose the bucket size in the same way for all four, and such that there are at least 10 buckets on each of the graphs.

5.4 Summary

Provide a detailed comparison of the sharded and non-shareded modes. For which multi-GET size is sharding the preferred option? Provide a detailed analysis of your system. Add any additional figures and experiments that help you illustrate your point and support your claims.

6 2K Analysis (90 pts)

For 3 client machines (with 64 total virtual clients per client VM) measure the throughput and response time of your system in a 2k experiment with repetitions. All GET operations have a single key. Investigate the following parameters:

- Memcached servers: 1 and 3
- Middlewares: 1 and 2
- Worker threads per MW: 8 and 32

Repeat the experiment for (a) a write-only and (b) a read-only workload. For each of the two workloads, what is the impact of these parameters on throughput, respectively response time?

Number of servers	1 and 3
Number of client machines	3
Instances of memtier per machine	1 (1 middleware) or 2 (2 middlewares)
Threads per memtier instance	2 (1 middleware) or 1 (2 middlewares)
Virtual clients per thread	32
Workload	Write-only and Read-only
Multi-Get behavior	N/A
Multi-Get size	N/A
Number of middlewares	1 and 2
Worker threads per middleware	8 and 32
Repetitions	3 or more (at least 1 minute each)

7 Queuing Model (90 pts)

Note that for queuing models it is enough to use the experimental results from the previous sections. It is, however, possible that the numbers you need are not only the ones in the figures we asked for, but also the internal measurements that you have obtained through instrumentation of your middleware.

7.1 M/M/1

Build queuing model based on Section 4 (write-only throughput) for each worker-thread configuration of the middleware. Use one M/M/1 queue to model your entire system. Motivate your choice of input parameters to the model. Explain for which experiments the predictions of the model match and for which they do not.

7.2 M/M/m

Build an M/M/m model based on Section 4, where each middleware worker thread is represented as one service. Motivate your choice of input parameters to the model. Explain for which experiments the predictions of the model match and for which they do not.

7.3 Network of Queues

Based on Section 3, build a network of queues which simulates your system. Motivate the design of your network of queues and relate it wherever possible to a component of your system. Motivate your choice of input parameters for the different queues inside the network. Perform a detailed analysis of the utilization of each component and clearly state what the bottleneck of your system is. Explain for which experiments the predictions of the model match and for which they do not.