# Advanced Systems Lab

Autumn Semester 2016 (263-0007-00L)

Gustavo Alonso

## 1 Course Overview

### 1.1 Prerequisites

This is a project based course operating on a self-study principle. The course relies on the students own initiative, aside from a few tutorials discussing the project and some basic supporting material, there will be no lectures in the conventional sense. The project as well as the course requires that the student complements the material covered in the tutorials with his/her own study as not all relevant and/or needed material will be covered. The grading will be based on the project; there is no exam. The pre-requisites for this course include knowledge at the Bachelors Level of the following topics:

- Algorithms and data structures
- Programming in Java (general knowledge of OO programming, threading, concurrency, and debugging)
- Networks (understanding of how computer networks operate and the basis of performance evaluation in networks)
- Socket Programming (TCP/IP networking)
- System Design (operating systems, concurrent systems, low level optimizations, instrumentation)
- Statistics

These topics will not be covered during the course but knowledge of them is assumed. If a student lacks the necessary background, it is expected that he/she will acquire the necessary knowledge on his/her own. No course or project requirements will be changed for individuals because of lack of previous knowledge in any of these areas. Students who do not have the necessary background or system development skills may want to consider taking instead one of the other two Advanced Labs offered by the department to comply with the requirements for the masters degree in computer science.

### 1.2 Objectives

The main goal of this course is to learn how to evaluate the performance of complex computer and software systems. The course offers an opportunity to bring together the knowledge and expertise acquired in different areas (networking, systems programming, parallel programming) by allowing students to build a multi-tier, distributed information system. The course focuses less on system design and development and more on the evaluation and modeling of the system: understanding the behavior, modeling its performance, code instrumentation, bottleneck detection, performance optimizations, as well as analytical and statistical modeling. The ability to

explain the behavior of the system plays a bigger role in the grading than the actual building of the system. However, we expect designs and code that have been thought through – major design mistakes and bad coding practices will affect the grade.

## 1.3 Grading

The course will be evaluated through a project. The project consists of three milestones, the first covers the system design and implementation. The second covers experimental evaluation. The third milestone consists of an analytical evaluation of the system based on the experimental results. All milestones need to be delivered to obtain a passing grade in the project. The three milestones are worth 200 points each, from which at least 100 need to be obtained to pass a milestone. In the end a minimum of 400 out of 600 points need to be collected to pass the project. Failing the project implies failing the course. Work in the project will be done on an individual basis. The milestones include delivering system code, a report, and experimental data. Students might be required to present the results in person and explain the system or results in the report. The dates for submitting the deliverables due at each milestone are fixed and cannot be changed. Failure to submit the results on time will result in failing the project. Submission happens through a repository hosted by ETH – details will be presented in the exercise session.

**The deadlines for the milestones are as follows:**

M1: **21st October, 14:00**

M2: **25th November, 14:00**

M3: **23rd December, 14:00**

It is important to note that the evaluation of the project is not based on completing a number of clearly specified task. The goal is to build a system and explain its behavior. Developing a system that works but for which no explanation can be produced on why it behaves the way it does is not sufficient to reach a passing grade. Similarly, collecting experimental data or developing a model is not enough to pass the corresponding milestones. A passing grade is reached by having the proper explanations for the data and the model, including its potential discrepancies with the implemented system.

## 1.4 Supporting Material

The following text book (available in the library [1]) can be very useful for the course: *The Art of Computer Systems Performance Analysis*, Raj Jain, Wiley Professional Computing, 1991.
Of particular relevance are the following chapters:

- Chapters 1, 2, 3 = general introduction, common terminology
- Chapters 4, 5, 6 = workloads
- Chapter 10 = data presentation
- Chapters 12, 13, 14 = probability and statistics
- Chapters 16, 17, 18, 20, 21, 22 = experiment design
- Chapters 30, 31, 32, 33, 36 = queuing theory

[1] `http://tiny.cc/artofcomputer`

# 2 Project Details

## 2.1 General Remarks

This course is a demanding one not because the task at hand is in itself difficult but because for many students it is the first time they are confronted with designing a system with several degrees of freedom and where skills from several areas of computer science need to be brought to bear into the problem to solve it. *Waiting until close to the deadline to complete a milestone will not work on this course.* Starting as early as possible is essential, and aiming at completing it well before the deadline is highly recommended so that there is enough time to analyze and understand what has been built. In this course, the focus is on understanding the system and being able to explain what it does and why it behaves the ways it does; this is more important than development of the system itself. No matter how much effort goes into the development, if the system is ready only shortly before the deadline, there will be no time for a complete experimental analysis and for developing a thorough understanding of the systems performance characteristics. However, it is mainly on these latter aspects where the project will be evaluated.

Experience from last editions of the course show that there are a number of typical problems with the project that occur because of poor time management:

- System works but behavior of performance traces cannot be explained (why response time grows over time?, why throughput peaks for a given number of clients?, what are the bottlenecks in the system?)
- System works but the traces are incomplete, experiments have not been repeated a sufficient number of times to achieve statistical significance, lack of proper statistical treatment of the data (confidence levels, deviation, factor analysis)
- Inconsistent results in the data collection with different experiments showing contradictory behavior
- Poor modeling or no explanations for the discrepancies between the model and the system
- Unexplained behavior of the system from a performance perspective

Even if the system is complete and works well, these situations typically lead to failing the milestones so make sure you devote sufficient time and effort to the analysis of the system.

In this project, a number of well established professional practices are highly advisable: create a timetable for your project and adhere to it, monitor your progress so that potential delays can be identified and dealt with early enough, use a version control system for development (will be provided), heavily document all your code, plan your experiments and experimental work, use a database to store your experimental data, keep a journal of experiments and experimental parameters, keep track of result files and data to make the analysis simpler later on. Make sure you have backup copies of your code, data, and reports.

**Communication** Please check the web page of the course [2] regularly. Also, make sure that your teaching assistant is informed of the progress you have made in the milestones. Avoid raising major issues in completing a milestone shortly before the deadline. An e-mail address is available for questions, requests, and feedback regarding the course [3].

[2] `http://www.systems.ethz.ch/courses/fall2016/asl`

[3] `mailto:sg-asl@lists.inf.ethz.ch`

**Infrastructure** The project requires a more extensive infrastructure than probably most students have ever used for a course project. All experiments will have to be run on the Microsoft Azure Cloud [4] platform using virtual machines running Linux. We will not accept any experiments ran on local computers. The course provides vouchers for access to the cloud platform, but you will have to first create your own account (no credit card needed) and apply for it with us. More details will follow in the exercise sessions and tutorials on the particular steps.

## 2.2 Problem Statement

The project consists of the design and development of a middleware platform for key-value stores as well as the evaluation and analysis of its performance, and development of an analytical model of the system, and an analysis of how the model predictions compare to the actual system behavior. The middleware will have to perform load balancing operations by steering traffic to different servers depending on the request content, and will also perform primary-secondary replication. While proper replication protocols and recovery after failures is necessary for real-world systems, for the purpose of this project we focus on the mechanism of writing data to multiple servers instead of one to make sure that even if there is a failure, no information is lost.

A key-value store is a simple data storage system that allows clients to store arbitrary data under keys of their liking. They are also called NoSQL databases because they are used similarly to how a traditional relational database would be used, but they relax some durability or correctness requirements in exchange of better performance. In this project, you will work with *memcached* [5], a very commonly used main-memory key-value store. The workloads that you will run on these servers are fully synthetic, generated using *memaslap* from the *libmemcached* C library (you will be provided with details on how to build it from source at the exercise sessions). In this project we will only rely on three memcached operations [6]: `get` to read a value associated with a key, `set` to set or update the value belonging to a key, and `delete` to remove a key-value pair from memcached.

The middleware is to be positioned in front of several memcached instances (servers) and fulfills the task of (a) load balancing through partitioning the key-space, and (b) replicating write requests to multiple servers. Figure 1 shows the internal structure of the middleware, which operates as follows:

i) It accepts connections and requests from clients through a predefined TCP socket. Use Java.NIO [7] to handle incoming requests.

ii) It extracts the key of a request and hashes the key to find the server to which the particular key belongs to (all servers should be assigned an equal portion of the global key-space using e.g. consistent hashing [8]), and depending on the operation type it pushes the request into the write or read queue belonging to that server.

iii-a) For read requests there is a thread pool dedicated for each server, which dequeue read requests from the read queue and execute them on their server. The answer to the read request is forwarded to the client as is.

[4] `https://azure.microsoft.com/en-us/`
[5] `https://memcached.org/`
[6] `https://github.com/memcached/memcached/blob/master/doc/protocol.txt`
[7] `https://en.wikipedia.org/wiki/Non-blocking_I/O_(Java)`
[8] `https://en.wikipedia.org/wiki/Consistent_hashing`

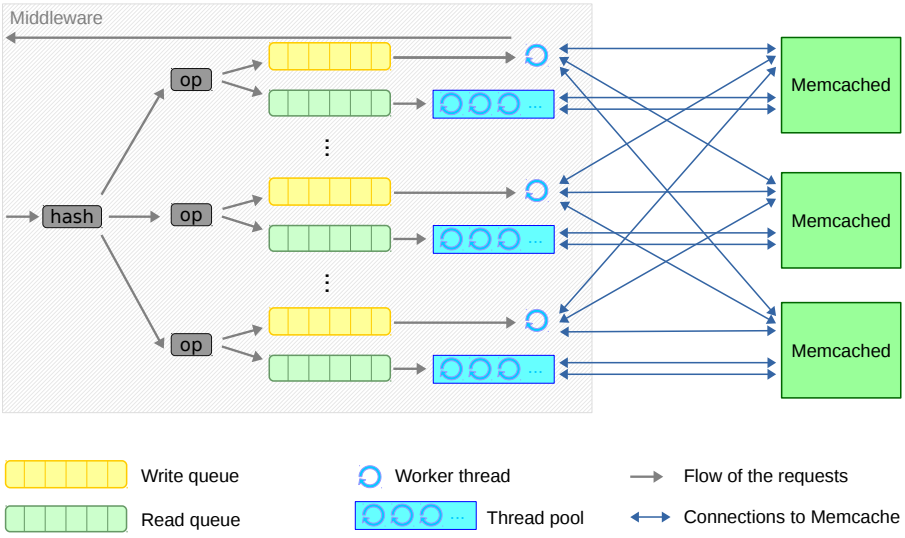


Figure 1: Middleware Architecture

iii-b) For each server there is a dedicated write thread that will dequeue write and delete requests one by one and forward them to the server. Depending on the replication strategy the write thread might have connections to multiple servers, and in this case it will have to forward each write request to all of them before the operation can be considered successful (see more information on replication below).

Replication is implemented using a primary-secondary approach. There is a primary location (copy) for each key, determined by the hash function. With $R$ being the number of servers the same write request is sent to before considering it finished, there are *R-1* secondary copies for each key. These are chosen as the "neighbors" of the server in the key-space. If $R$=1 we say the system is unreplicated. If $R$=N (N being the number of servers) all servers contain the same data. Note that read operations are always performed on the primary location regardless of the replication level.

## 2.3 Instrumentation

The system will be benchmarked by generating load with the memaslap clients. These clients measure throughput and response time on the client side, but do not provide a detailed breakdown of the relative costs inside the middleware and the servers. Therefore, the middleware code needs to be instrumented to measure for *each* request the following metrics:

- Time spent between receiving a request from client and sending final response.
- Time each request spends in a queue (time of enqueueing by the hashing thread - time of dequeueing by the processing thread)
- Processing time of the request, that is, the time between sending the request to the server and receiving the answer.
- In addition to the time measurement, each log entry should also encode whether the request has been executed successfully (success flag).

### 2.4 Middleware Parameters and Requirements

The middleware has to be parametrizable at startup using three parameters:

N: Number of memcached servers.

T: Number of threads in the read thread pool of *each* server. Note that there is only one write thread per server.

R: Replication factor, i.e., to how many servers a write is propagated to.

In order to successfully measure the system, the middleware has to be able to connect to at least up to 7 servers, and accept connections from at least 3 load generating machines. The middleware runs on a single machine, with at least 4 cores. Load generators and servers have to run on single or dual core machines. Use virtual machines of types *A Basic*, namely A2 for the load generators and memcached servers, and A4 for the middleware.

### 2.5 Milestone Structure and Deliverables

#### Milestone 1

At the end of this milestone, the system has to (a) be stable, (b) be able to forward requests to servers, and (c) collect results. It needs to be instrumented and its performance measured in a 1 hour trace that shows throughput and response time plotted over time. For the trace, the configuration is as follows: 3 client machines are connected to the middleware, which in turn connects to 3 servers. The trace has to be run with $R$=3 (write to all three).

In addition to the trace, the baseline performance of a single memcached server connected to one or multiple client machines should also be explored. The basic questions to answer are: what is the maximum throughput of the server and the response time without the middleware?, how much overhead does the middleware introduce over the baseline numbers?

For each milestone, we will provide a detailed report outline that has to be followed for submission. The details of these will be presented in an upcoming exercise session. However, we expect you to explain the architecture and main design decisions of the system in the first milestone.

#### Milestone 2

The aim of the second milestone is to conduct a detailed experimental evaluation of the system, and answer the following questions:

- How does a detailed time-breakdown of different operations look like? Are requests spending more time queueing in the system than executing?
- What is the maximum throughput of the system? In what configuration (T, client machines) is this throughput achieved?
- How does the workload (ratio of reads over writes) impact performance?
- What is the impact of the replication factor on the system? Is there a point where replication becomes so expensive that it limits the overall performance of the system?
- What is the most time-consuming operation for read and write requests? If it is in the middleware, how is it impacted by different parameters?

Note that for milestone 2, we expect you to measure and explain the behavior of your system, but no analysis (modeling with queueing theory) is required at that stage.

The report will consist of description of the experimental infrastructure, an overview of experiments and results, as well as statistical treatment of the data. We will require that you submit all experimental results with this milestone, so plan ahead in keeping and archiving results that you might use.

**Milestone 3**

The goal of the final milestone is to develop an analytical queuing model for each component of the system and for the system as a whole. Using the model, derive the performance characteristics that the model predicts and compare them with the results obtained in milestones 1 and 2. Explain where the model and data match and where they do not match, including sufficiently detailed explanations of the code/system/hardware characteristics behind the observed behavior. There should be no more experiments necessary to finish the third milestone (assuming a successful completion of the previous ones).

We will ask you to model the whole system first as a black box, using a naive model, and then as a more elaborate network of queues. To successfully model the system as a network of queues you will have to understand how requests are handled within the middleware and how long they spend in various processing stages.

In addition to the models, we will also ask for focusing on specific experiments from the second milestone and finding out how much each parameter influences the overall performance. More details will follow in the report outline.