# Is SystemOne “Slow”?

Rommel V. Bulalacao , Ivy Joy U. Aguila, Joseph Anthony C. Hermocilla
Institute of Computer Science
rvbulalacao@up.edu.ph, iuaguila@up.edu.ph, jchermocilla@up.edu.ph

SystemOne Decaf (S1) is a web application running on Linux-Apache-MySQL-PHP(LAMP) stack that supports the student registration process of UPLB. It is accessed over the Internetand deployed using the Client-Server application architecture. In this paper, we will answer the frequently asked question by its main users, the students: Why is SystemOne “slow”?

It is easy to say that a system is slow. But it will be more convincing if it said with some quantitative proof. Thus, in this work, we present some numbers that can quantifywhetherS1is indeed “slow”, or not.

We are interested in measuring the Quality of Service (QoS) of S1 which is defined in terms of response time, throughput, and availability. This is done by loadtesting S1, sending requests that emulates the behavior of the students during the peak of the registration period to the server via scripts, and obtaining measurements. An overview of S1’s software architecture is presented to enumerate the different paths taken by clients’ requests to the server, and back. These paths affect the QoS. We obtain measurements for different system deployment configurations for comparison. The result of load testing can tell us the best system configuration for S1 and more importantly it can tell us why it is slow or if it is indeed slow. It can also give us some insights on how and where we can improve the system.

Initial results of our laboratory tests indicate that the main bottleneck is in the storage/database(MySQL) component of S1. This result is validated by analysing logs obtained from actual deployments of S1 for the past semesters.

Keywords: web application architecture, workload, quality of service, response time, throughput

# Is SystemOne "slow"?

Rommel V. Bulalacao, Ivy Joy U. Aguila, Joseph Anthony C. Hermocilla
Institute of Computer Science
rvbulalacao@up.edu.ph, iuaguila@up.edu.ph, jchermocilla@up.edu.ph

## Abstract

SystemOne Decaf (S1) is a web application running on Linux-Apache-MySQL-PHP(LAMP) stack that supports the student registration process of UPLB. It is accessed over the Internet and deployed using the Client-Server application architecture. In this paper, we will answer the frequently asked question by its main users, the students: Why is SystemOne "slow"?

It is easy to say that a system is slow. But it will be more convincing if it said with some quantitative proof. Thus, in this work, we present some numbers that can quantify whether S1 is indeed "slow", or not.

We are interested in measuring the Quality of Service (QoS) of S1 which is defined in terms of response time and availability. This is done by load testing S1, sending requests that simulates the behavior of the students during the peak of the registration period to the server, via scripts, and obtaining measurements. An overview of S1's software architecture is presented to enumerate the different paths taken by clients' requests to the server, and back. These paths affect the QoS. We obtain measurements for different system deployment configurations for comparison. The result of load testing can tell us the best system configuration for S1 and more importantly it can tell us why it is slow or if it is indeed slow. It can also give us some insights on how and where we can improve the system.

Initial results of our laboratory tests indicate that the main bottleneck is in the storage/database(MySQL) component of S1. This result is validated by analysing logs obtained from actual deployments of S1 for the past semesters.

**Keywords**: web application architecture, workload, quality of service, response time, throughput

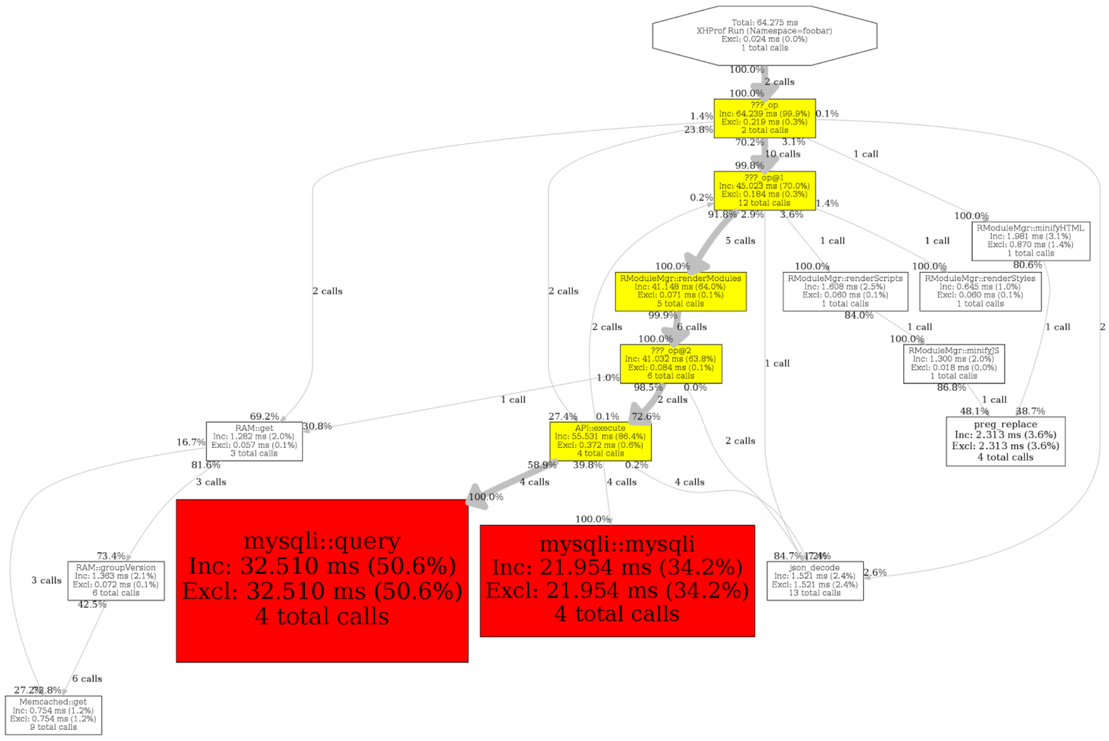
## Introduction

SystemOne Decaf (S1) is a Web application to support the registration process of UPLB. It runs on a Linux-Apache-MySQL-PHP (LAMP) stack. In this paper, we will try to answer the frequently asked question by its main users, the students: Why is SystemOne "slow"?

S1 is affected by the Slashdot/flash crowd effect especially during the peak of the registration period. This makes the application virtually inaccessible, or as perceived by students as "slow"[14][15]. Thus, in this work, we present some numbers that can quantify whether S1 is indeed 'slow', or not. First, we give an overview of S1 application and deployment architecture.

### S1 Application Architecture

The bulk of S1 is written in the PHP scripting language. Functionalities are implemented as 'modules' and there is a PHP class which serves as the application programming interface (API), exposing and managing module execution. Majority of the domain-specific logic related to the registration process are implemented as MySQL stored procedures and functions. Modules, when executed by API, call these stored procedures through MySQL-related PHP functions, *mysqli()* and *mysqli_query()*. What is sent back to the browser (HTML, Javascript, CSS, and images) is generated by functions from the *RModuleMgr* class. Figure 1 shows the function call graph when a user visits the index page. Some notable features of S1 include compression, javascript and CSS minification, and caching[17].



Total: 64.275 ms XHProf Run (Namespace=foobar) Excl: 0.024 ms (0.0%) 1 total calls
???_op Inc: 64.239 ms (99.9%) Excl: 0.219 ms (0.3%) 2 total calls
???_op@1 Inc: 45.023 ms (70.0%) Excl: 0.184 ms (0.3%) 12 total calls
RModuleMgr::renderModules Inc: 41.148 ms (64.0%) Excl: 0.071 ms (0.1%) 5 total calls
???_op@2 Inc: 41.032 ms (63.8%) Excl: 0.084 ms (0.1%) 6 total calls
API::execute Inc: 55.531 ms (86.4%) Excl: 0.372 ms (0.6%) 4 total calls
RModuleMgr::minifyHTML Inc: 1.981 ms (3.1%) Excl: 0.870 ms (1.4%) 1 total calls
RModuleMgr::renderScripts Inc: 1.608 ms (2.5%) Excl: 0.060 ms (0.1%) 1 total calls
RModuleMgr::renderStyles Inc: 0.645 ms (1.0%) Excl: 0.060 ms (0.1%) 1 total calls
RModuleMgr::minifyJS Inc: 1.300 ms (2.0%) Excl: 0.018 ms (0.0%) 1 total calls
preg_replace Inc: 2.313 ms (3.6%) Excl: 2.313 ms (3.6%) 4 total calls
RAM::get Inc: 1.282 ms (2.0%) Excl: 0.057 ms (0.1%) 3 total calls
RAM::groupVersion Inc: 1.363 ms (2.1%) Excl: 0.072 ms (0.1%) 6 total calls
Memcached::get Inc: 0.754 ms (1.2%) Excl: 0.754 ms (1.2%) 9 total calls
mysqli::query Inc: 32.510 ms (50.6%) Excl: 32.510 ms (50.6%) 4 total calls
mysqli::mysqli Inc: 21.954 ms (34.2%) Excl: 21.954 ms (34.2%) 4 total calls
json_decode Inc: 1.521 ms (2.4%) Excl: 1.521 ms (2.4%) 13 total calls

Figure 1. Function call graph when visiting the index page of S1.

**S1 Deployment Architecture**

S1 deployment follows the client-server architecture. The client sends *requests* (for resources such as HTML documents, images, scripts, etc.) to the server and the server sends *responses* the client. Performance(response time) is usually a measure of how long it takes for the responses to arrive to the client after a set of requests is sent. When the server receives a request and processes it is called a *hit*. There can be multiple clients connecting to a single server. The client and the server may be geographically separated in which case requests and responses will travel across a network, usually over the Internet.

From the user's end or client-side, S1 is accessed via a web browser, such as Mozilla Firefox or Google Chrome, by typing S1's web address(URL), *http://systemone.uplb.edu.ph*, on the address bar. The browser sends HTTP[16] requests to the server (identified by the web address) hosting the application. What the user sees are the resources or files (HTML, CSS, Javascript, images), which are the responses to the requests, processed by the browser and displayed or rendered on the screen.

At the back-end or server-side, there are a lot of things going on. Every request from the browser will have to be processed by different subsystems or services. These include the base operating system(Linux) network stack implementation, firewall(iptables), web server(Apache), the server-side scripting engine(PHP), modules of S1 itself, and the database server(MySQL). Figure 2 shows the client-server deployment of S1.

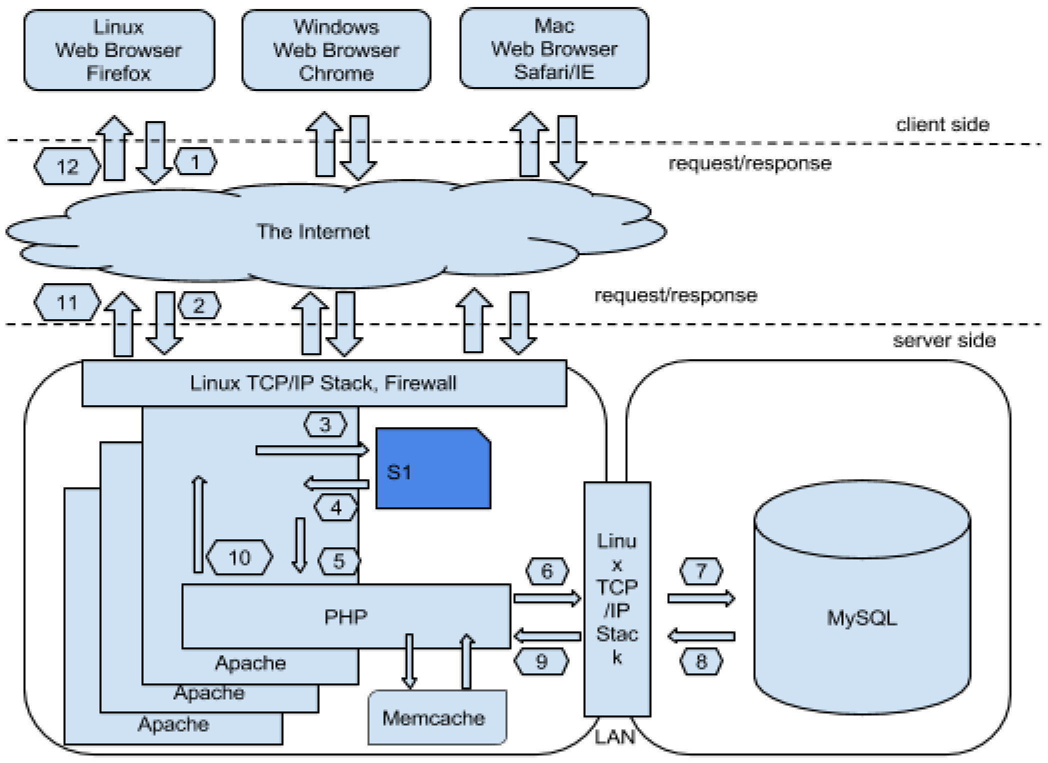


Figure 2. The client-server architecture. Arrows represent the flow of data from the initial request (1) to the final response (12). At (10), the dynamic content has been generated. Multiple Apache processes handle multiple user requests.

**Web Application Deployment Architectures**

Although different Web applications would have different deployment requirements, S1 shares features common to most Web applications. That is, end-users can view, create, update, and delete a piece of information through its Web interface. Hence, it is prudent to compare its deployment architecture against that of other Web applications. Figure 3 shows a typical deployment architecture for web applications with emphasis on network topology. The presented deployment architecture has three significant benefits: performance, reliability, and security.

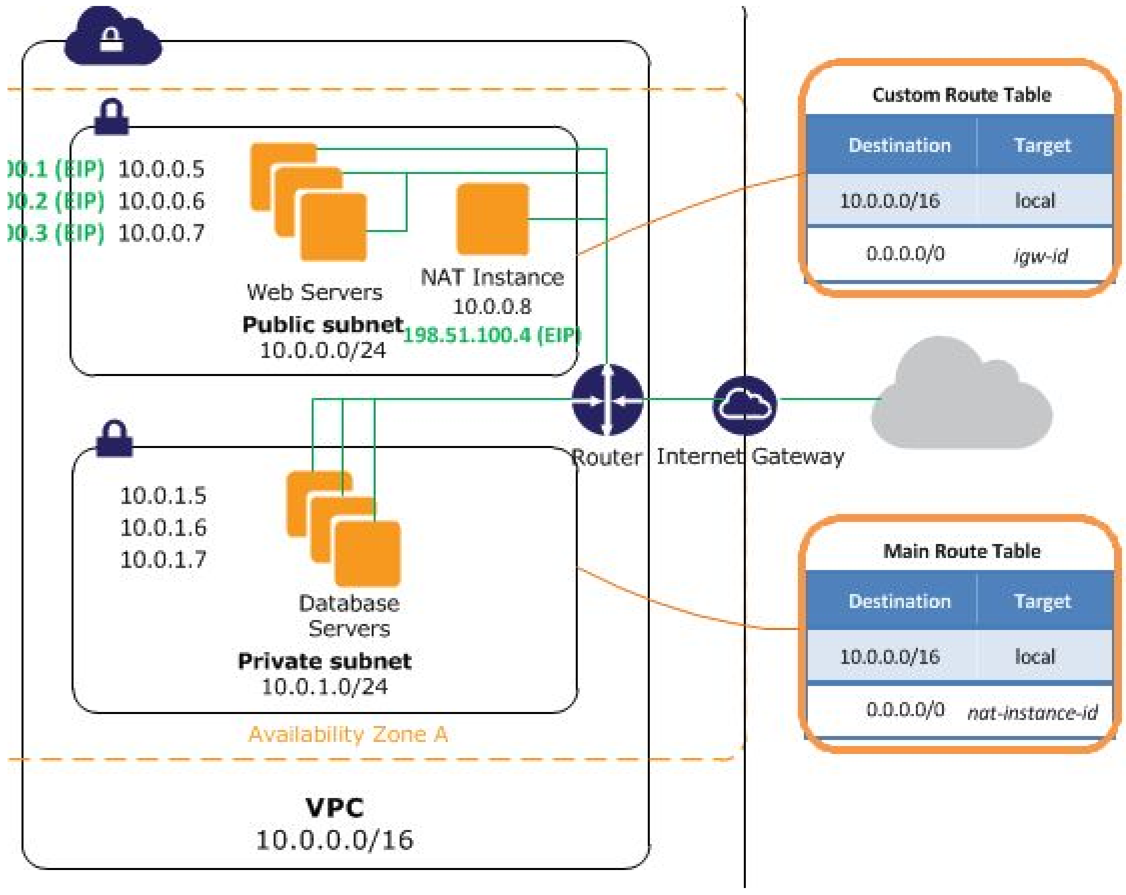


Figure 3. Typical deployment architecture for web applications.

*Performance*

The primary goal of improving the performance of a Web application is to minimize the perceived delay that the user experiences between initiating the interaction to seeing the result of the initiated interaction. This perceived delay is affected by several factors, including the application and deployment architectures.

We are interested in measuring the Quality of Service (QoS) of S1. This is measured in terms of *response time*, *throughput*, and *availability*[1]. By load testing S1, we will be able to assess how it will support its expected workload, especially during the registration period. This will be accomplished by running a set of scripts that emulate the behavior of students as they use S1. At the deployment's end, system engineers will be able to assess the behavior of the application to gain more information needed to tweak the setup. The parameters to vary in a load include *workload intensity*, *workload mix*, and *customer behavior patterns*[1].

## Methodology

The steps we took in this study is summarized below. Details are discussed in the subsections.

1. Prepare S1 version and dataset to use.
2. Create a basic S1 setup. Record interaction of students using JMeter[7].
3. Setup S1 using different deployment configurations.
4. Identify variables to measure (both client side and server side).
5. Run JMeter using recorded interactions to obtain baseline measurements[21]. Use data derived past deployments to generate test workload.
6. Increase number of users and concurrent connections to simulate actual workload.
7. Analyze and evaluate results.

### Dataset

The dataset used was retrieved at 8pm on August 2, 2015 from a live S1 deployment for the First Semester AY 2015-2016.

### Load Generation

In order to simulate the workload, we loaded the dataset to a two-server test setup then asked ten(10) students to use S1 as they normally do during the registration period. We recorded the students  interactions with S1 using JMeter. These recorded interactions was then used as the workload for the JMeter test plan. We also considered logging(on the server side using the dumpio Apache module) everything during actual deployments in order to generate the actual workload. However, we discarded this idea because doing so will further degrade the performance due to high disk activity when writing logs.

We focused our evaluation on the *login* workflow. This workflow starts when, after landing on the index page, the student enters the username and password then presses the login button. The workflow is complete when the profile page of the student is fully loaded.

### Server Configuration

As shown in Figure 2, the client-server architecture allows the different server-side components to be placed on different servers. In the case of S1, these components are Apache and MySQL. Through the years, S1 has been deployed with the following server configurations. In this study, we used the *Single-server Setup*, using the same hardware used in the live deployment(aka Rodolfo). SystemOne Operations Manual[10] was used to prepare this setup.

*Single-server Setup*

In the single-server setup we have one server that runs both Apache and MySQL. This setup was used until the Second Semester AY 2014-2015 S1 deployment using bare-metal server[8 X Intel(R) Xeon(R) CPU E5640 @ 2.67GHz, 32GB RAM, 53GB SSD w/ RAID 1].

*Two-server Setup*
In the two-server setup, one server runs Apache and another server runs MySQL. This setup was used during the Mid-year AY 2014-2015 S1 deployment using virtual servers on the cloud[18].

*Four-server Setup*
In the four-server setup, we have one server running Nginx for load balancing, two servers for Apache and one server for MySQL. This setup was used during the First Semester AY 2015-2016 S1 deployment using virtual servers on the cloud[18].

**Metrics Collected**
On the client side, the following metrics were obtained using JMeter for a given run:

*Average:* The average time(ms) of a set of results.
*Median:* The middle value for the response time(ms).
*90th Percentile:* 90% of the sample took no more than this time.
*Median:* Divides the samples into two halves.
*Min:* Shortest time for the run.
*Max:* Longest time for the run.
*Error%*: Error rate per run.

On the server side, the following metrics were obtained using custom scripts, sampling every two(2) seconds for each test run:

*mysql_processes:* The amount of memory used by MySQL.
*mysql_cpu*: The percentage of the CPU used by MySQL.
*apache_processes*: Amount of memory used by the Apache.
*system_mem_available*: Amount of free system memory.

The following variables related to MySQL were also noted after each run:

*aborted_clients*: The total number of aborted connections.
*queries_executed*: The total number of queries executed.
*queries_quitted*: The total number of queries quitted.
*slow_queries*: The total number of slow queries.

**S1 Post-deployment Web Server Statistics**
We analyzed web server statistics from three deployments of S1 to provide additional inputs in creating the test plan to determine the workload intensity. These statistics were generated by awstats[11] from log files. From Table 1, 1st Semester 2015-2016 has the least number of hits. This may be attributed to the filtering done by CloudFlare[19], which was used for the first time on that semester.

Table 1. General statistics.

| Semester | Unique Visitors | Number of Visit | Pages | Hits | Bandwidth (GB) |
|---|---|---|---|---|---|
| 2nd Semester 2014-2015 (18-31 January 2015) | 28,206 | 109,789 | 5,697,011 | 10,575,319 | 38.24 |
| Mid Year 2014-2015 (11-24 June 2015) | 18,304 | 36,547 | 2,061,920 | 3,676,543 | 30.03 |
| 1st Semester 2015-2016 (1-2 August 2015) | 6,115 | 10,647 | 2,165,960 | 2,367,529 | 11.13 |

Table 2. Maximum values generated for a single day(24 hours) during a deployment.

| Semester | Max # of Visits | Max # of Pages | Max # of Hits | (Average Hits/second) |
|---|---|---|---|---|
| 2nd Semester 2014-2015 ( January19, 2015) | 22,543 | 2,246,762 | 4,151,650 | 48 |
| Mid Year 2014-2015 (June 23, 2015) | 10,868 | 912,662 | 1,963,444 | 23 |
| 1st Semester 2015-2016 (August 1, 2015) | 3,643 | 1,441,016 | 1,447,879 | 16 |

## Number of Threads/Users per Test Computer

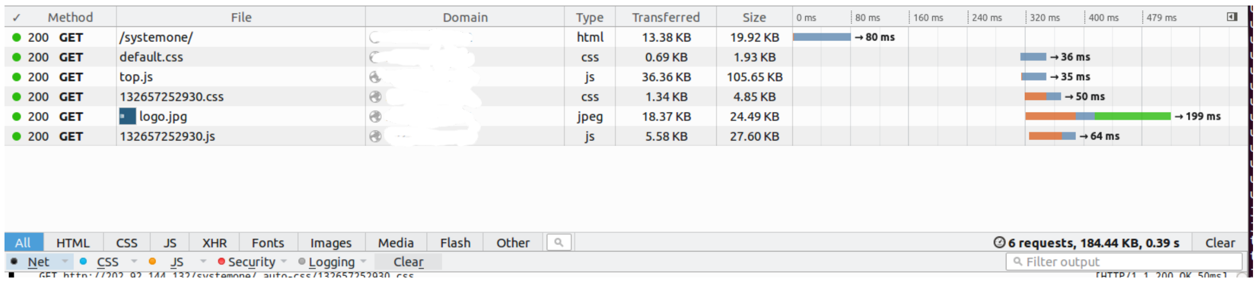
We measured the maximum number of threads/users per computer to have a 90th percentile of 10 seconds with no errors per run. Then we simultaneously run the tests by scheduling them to run at a specified time.

There were five data sets tested on 18 computers: 100, 500, 1000, 1500, 2000. We concurrently run each test by scheduling them to execute at a specified time. The goal is to at least simulate the max number of visits per day which is 22,543 (refer on Table 2) and to see the trend with different workloads on the server.

## Results and Discussion

### Baseline Measurements

Visiting the index page of S1 generates six(6) GET requests from the browser, with an average payload size of 31KB. Firefox can use six(6) concurrent connections[13] for a single domain. As shown in Figure 4, five(5) connections are used concurrently to generate the requests for the resources in the index page. For the login workflow(Figure 5), ten(10) requests are sent to the server. Some of the requests are non-viewable (redirects). Figures 6 and 7 shows requests when index and login workflow are accessed outside of the campus.

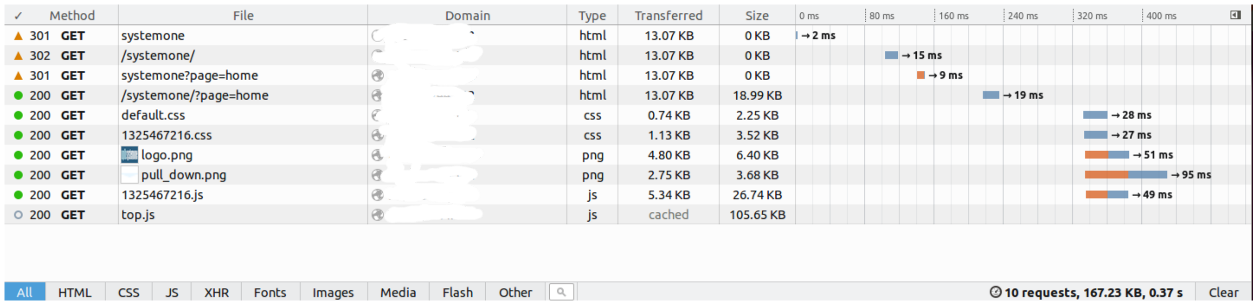


| ✓ | Method | File | Domain | Type | Transferred | Size | 0 ms – 479 ms |
|---|---|---|---|---|---|---|---|
| ● 200 | GET | /systemone/ | [redacted] | html | 13.38 KB | 19.92 KB | → 80 ms |
| ● 200 | GET | default.css | [redacted] | css | 0.69 KB | 1.93 KB | → 36 ms |
| ● 200 | GET | top.js | [redacted] | js | 36.36 KB | 105.65 KB | → 35 ms |
| ● 200 | GET | 132657252930.css | [redacted] | css | 1.34 KB | 4.85 KB | → 50 ms |
| ● 200 | GET | logo.jpg | [redacted] | jpeg | 18.37 KB | 24.49 KB | → 199 ms |
| ● 200 | GET | 132657252930.js | [redacted] | js | 5.58 KB | 27.60 KB | → 64 ms |

6 requests, 184.44 KB, 0.39 s

```
GET http://202.92.144.132/systemone/_auto-css/132657252930.css                [HTTP/1.1 200 OK 50ms]
```

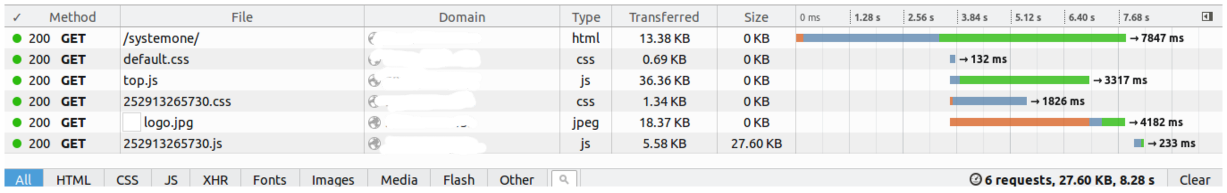
Figure 4. Index(within campus network). It took 0.39s to complete.



| ✓ | Method | File | Domain | Type | Transferred | Size | 0 ms – 400 ms |
|---|---|---|---|---|---|---|---|
| ▲ 301 | GET | systemone | [redacted] | html | 13.07 KB | 0 KB | → 2 ms |
| ▲ 302 | GET | /systemone/ | [redacted] | html | 13.07 KB | 0 KB | → 15 ms |
| ▲ 301 | GET | systemone?page=home | [redacted] | html | 13.07 KB | 0 KB | → 9 ms |
| ● 200 | GET | /systemone/?page=home | [redacted] | html | 13.07 KB | 18.99 KB | → 19 ms |
| ● 200 | GET | default.css | [redacted] | css | 0.74 KB | 2.25 KB | → 28 ms |
| ● 200 | GET | 1325467216.css | [redacted] | css | 1.13 KB | 3.52 KB | → 27 ms |
| ● 200 | GET | logo.png | [redacted] | png | 4.80 KB | 6.40 KB | → 51 ms |
| ● 200 | GET | pull_down.png | [redacted] | png | 2.75 KB | 3.68 KB | → 95 ms |
| ● 200 | GET | 1325467216.js | [redacted] | js | 5.34 KB | 26.74 KB | → 49 ms |
| ○ 200 | GET | top.js | [redacted] | js | cached | 105.65 KB | |

10 requests, 167.23 KB, 0.37 s

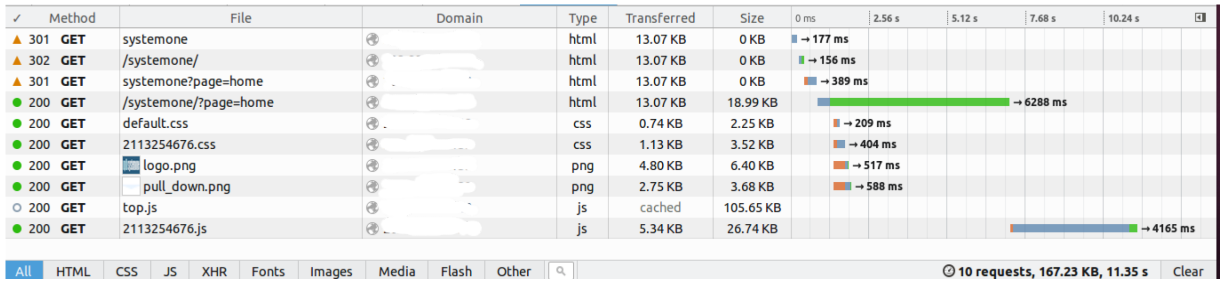
Figure 5. Login(within campus network). It took 0.37s to complete.



| ✓ | Method | File | Domain | Type | Transferred | Size | 0 ms – 7.68 s |
|---|---|---|---|---|---|---|---|
| ● 200 | GET | /systemone/ | [redacted] | html | 13.38 KB | 0 KB | → 7847 ms |
| ● 200 | GET | default.css | [redacted] | css | 0.69 KB | 0 KB | → 132 ms |
| ● 200 | GET | top.js | [redacted] | js | 36.36 KB | 0 KB | → 3317 ms |
| ● 200 | GET | 252913265730.css | [redacted] | css | 1.34 KB | 0 KB | → 1826 ms |
| ● 200 | GET | logo.jpg | [redacted] | jpeg | 18.37 KB | 0 KB | → 4182 ms |
| ● 200 | GET | 252913265730.js | [redacted] | js | 5.58 KB | 27.60 KB | → 233 ms |

6 requests, 27.60 KB, 8.28 s

Figure 6. Index(off- campus network). It took 8.28s to complete.



| ✓ | Method | File | Domain | Type | Transferred | Size | 0 ms 2.56 s 5.12 s 7.68 s 10.24 s |
|---|---|---|---|---|---|---|---|
| ▲ 301 | GET | systemone | [redacted] | html | 13.07 KB | 0 KB | → 177 ms |
| ▲ 302 | GET | /systemone/ | [redacted] | html | 13.07 KB | 0 KB | → 156 ms |
| ▲ 301 | GET | systemone?page=home | [redacted] | html | 13.07 KB | 0 KB | → 389 ms |
| ● 200 | GET | /systemone/?page=home | [redacted] | html | 13.07 KB | 18.99 KB | → 6288 ms |
| ● 200 | GET | default.css | [redacted] | css | 0.74 KB | 2.25 KB | → 209 ms |
| ● 200 | GET | 2113254676.css | [redacted] | css | 1.13 KB | 3.52 KB | → 404 ms |
| ● 200 | GET | logo.png | [redacted] | png | 4.80 KB | 6.40 KB | → 517 ms |
| ● 200 | GET | pull_down.png | [redacted] | png | 2.75 KB | 3.68 KB | → 588 ms |
| ○ 200 | GET | top.js | [redacted] | js | cached | 105.65 KB | |
| ● 200 | GET | 2113254676.js | [redacted] | js | 5.34 KB | 26.74 KB | → 4165 ms |

All HTML CSS JS XHR Fonts Images Media Flash Other

10 requests, 167.23 KB, 11.35 s Clear

Figure 7. Login(off- campus network). It took 11.35s to complete.

At the server side, a single login generates the following MySQL statistics:

*connections_established*: 6
*queries_quitted*: 2
*queries_executed*: 8
*aborted_clients*: 4

Given the above data, 36,000 users who login will generate 360,000 hits if all requests were successfully processed. As for the MySQL statistics, we simply multiply the values of the variables with the number of users. However, this is the ideal scenario. The next section presents the actual numbers.

**Test Results**

Figure 8 shows the actual number of requests sent by JMeter side by side with the received and processed by the server. The figure shows that some of the samples did not reach the servers due to some errors or reached the servers but encountered some errors while processing it. To understand what happens we present data both on the client and server sides.

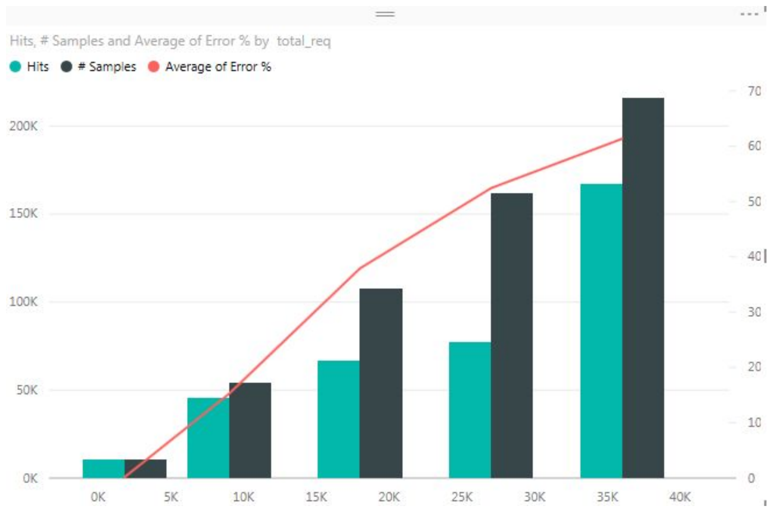


Figure 8. Actual Sample vs. Requests processed by the server

*Client Side*

Figure 9 shows the 90% Line trend per data size. It can be concluded here that, with 9000 users, there will be a sudden increase of time to finish the Login request experienced by the 90% of the users in comparison with 1800 concurrent users. With 1800 users, the 90% of the user finished logging in within 5 secs while with 9000 users, the 90% line is 126.07 secs.

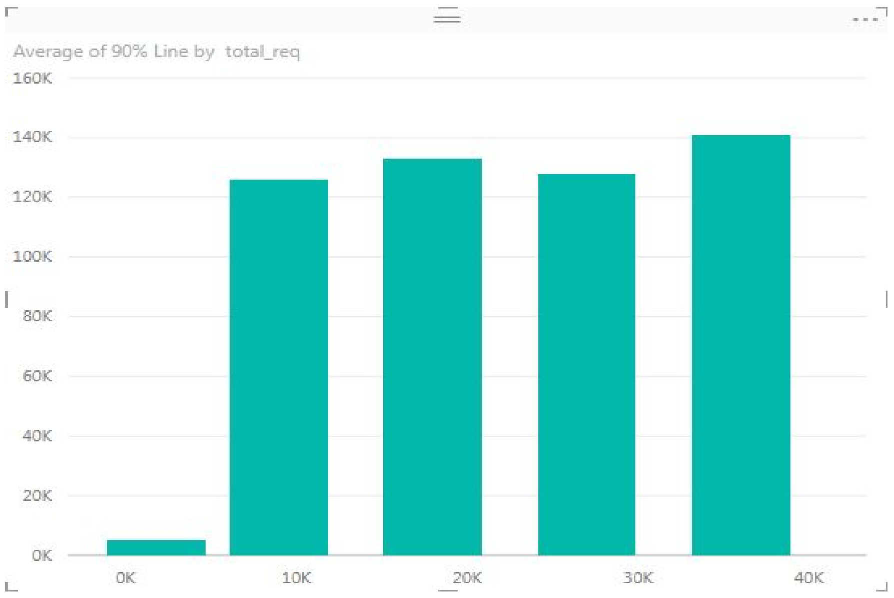


Figure 9. Average time each 90th percentile of the samples finished the request-response process

In Figure 10, the distribution of these errors for 36,000 and 18,000 users is shown.

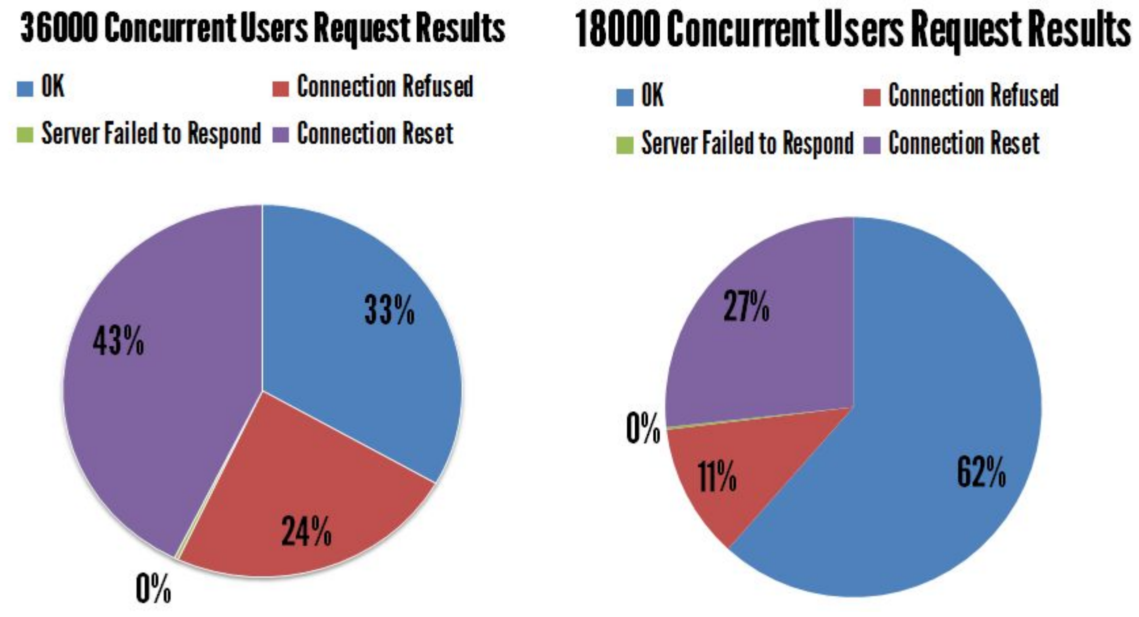


Figure 10. Distribution of the requests status after the simulation as perceived in the client side

The following are the request results encountered by JMeter:

- OK - the request is served successfully.
- Connection Refused -
- Connection Reset - normally encountered every time there is an unsynchronized connection between the client and server.
- Server Failed to Respond

In Figure 11, we compared the % error occurred for each page loaded for the login workflow. The following were the six pages compared in the graph. The */systemone/ajax/processor.php?login* page occured to have the most % errors among all the pages listed above. This page is dynamically generated.

- */systemone/ajax/processor.php?login*
- */systemone/*
- */systemone*
- */systemone/*
- */systemone/_auto-css/2113254676.css*
- */systemone/_auto-js/2113254676.js*

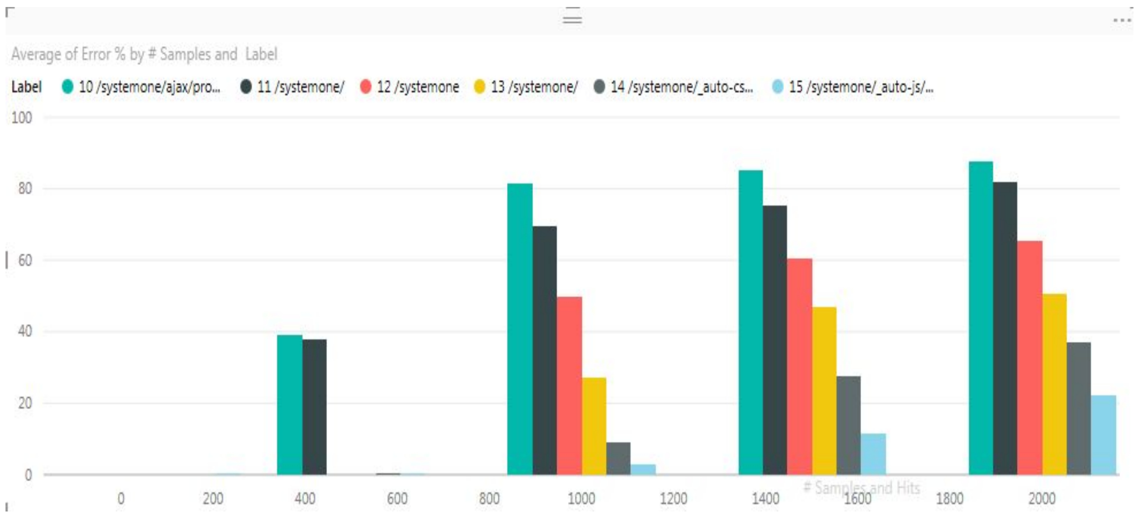


Figure 11. Encountered % Error per page of each sample size

Figure 12 shows the average elapsed time(ms) to complete the request-response process.

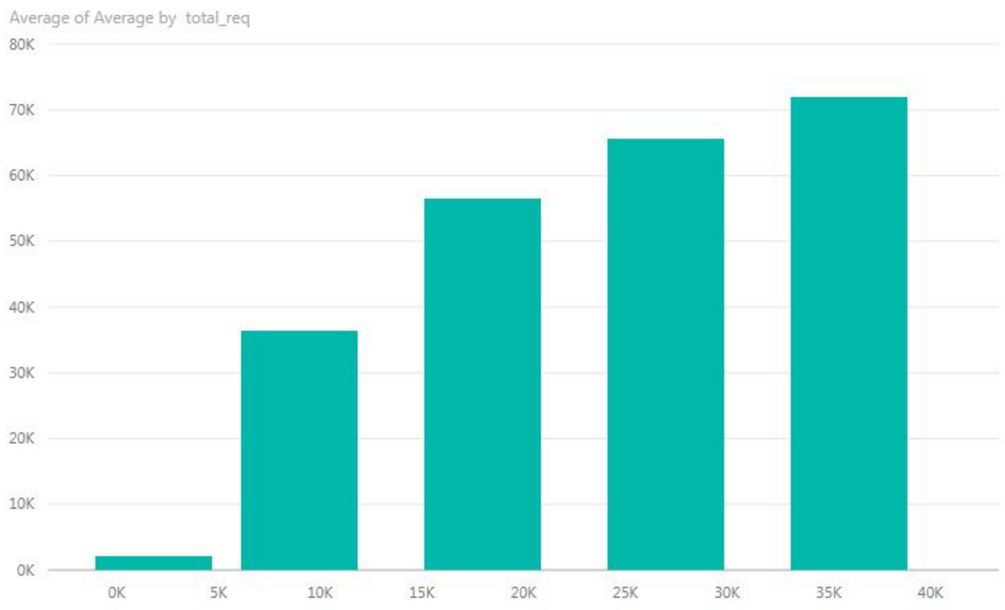


Figure 12. Average completion time per number of requests

Table 3 represents the above data:

Table 3. Average completion time per number of requests

| No. of Samples (requests) | Average Time (secs) |
|---|---|
| 1,800 | 2.091 |
| 9,000 | 36.395 |
| 18,000 | 56.342 |
| 27,000 | 65.524 |
| 36,000 | 71.931 |

*Server Side*
Figure 13 shows the total system memory consumed. Even if there are 36,000 concurrent users accessing the server, there is still enough available memory to run the server.

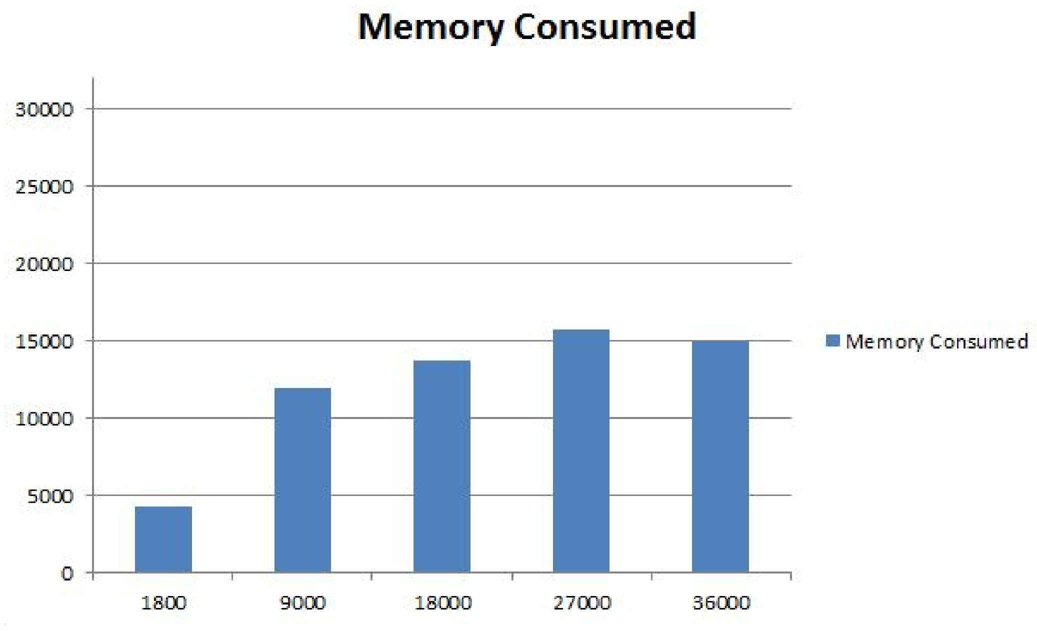


Figure 13. Total system memory consumed

In Figure 14, Apache limit(approximately 6000 processes) has been reached with the data size starting at 18,000 concurrent users and above.

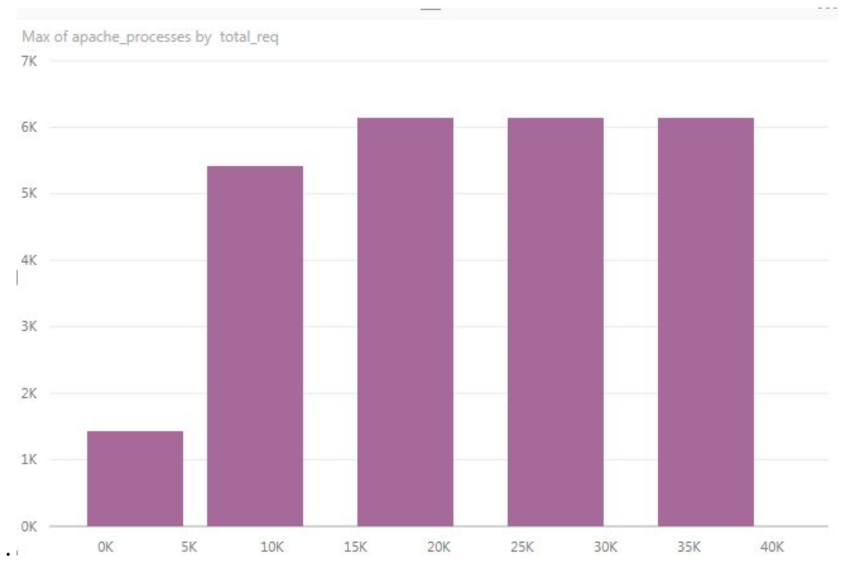


Figure 14. Maximum apache processes encountered for each sample

Figure 15 shows the statistics for MySQL for each set of requests. The number of slow queries and quitted queries tends to become constant(40,000), at around 39000, as the number of requests increases.

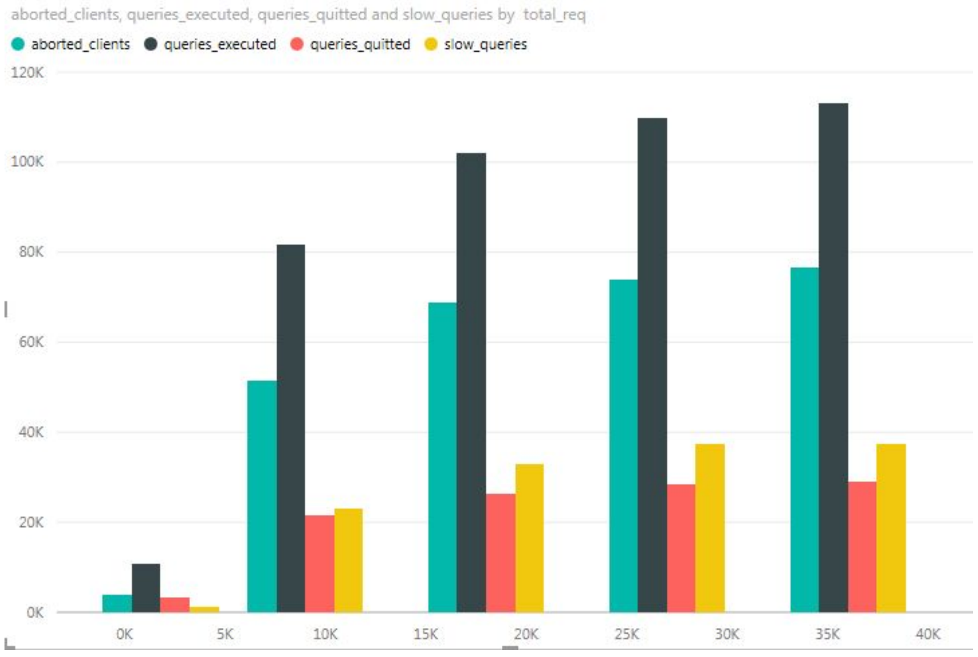
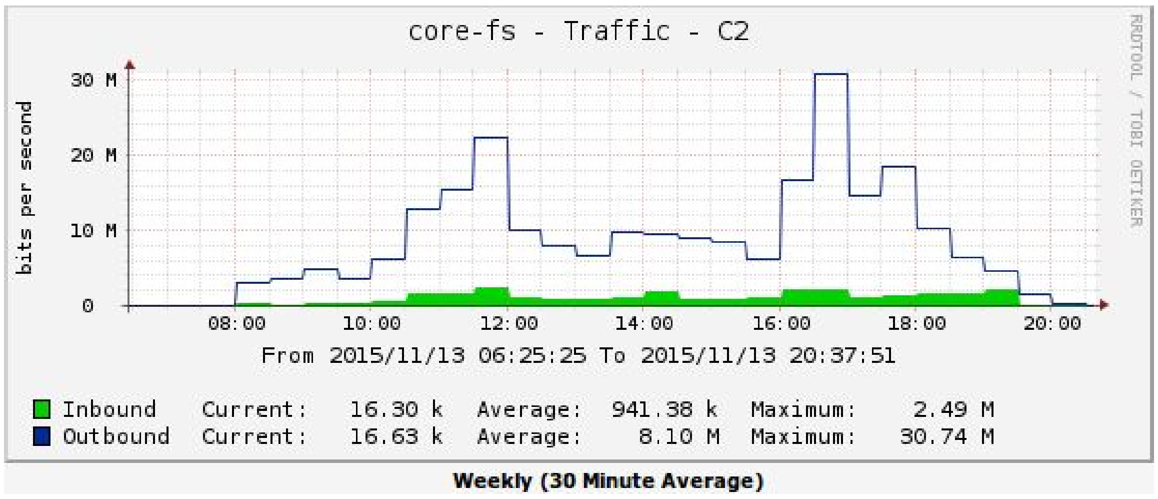


Figure 15. Queries executed, aborted clients, queries quitted, and slow queries per sample size

Figure 16[20] shows the network traffic from the laboratory network where the test was conducted at about 16:00-20:00. The test was conducted with other students also accessing the network because of regular classes. Outbound traffic represents the traffic containing requests which peaked to 30Mbps. The largest number of concurrent requests(36000) was made at around 18:45-19:15, with about 6Mbps traffic generated.



```
From 2015/11/13 06:25:25 To 2015/11/13 20:37:51

🟩 Inbound    Current:   16.30 k  Average:  941.38 k  Maximum:    2.49 M
🟦 Outbound   Current:   16.63 k  Average:    8.10 M  Maximum:   30.74 M
```

Figure 16. ICS Laboratory network traffic during the time of experiment

## Related Work

The work of Menascé gives an overview of load testing web applications[1]. Anderson discussed some common mistakes in load testing applications[2]. Shaw conducted a study to assess the performance of an online learning application[3]. Benchmarking solutions exist for such as TCP-W[4] and others[5] exists. Characterizing workload has been studied by Arlitt[6].

## Conclusion and Future Work

In conclusion, given the *Single-Server Setup*, with 9000 users, there will be a sudden increase of time to finish the login request experienced by the 90% of the users in comparison with 1800 concurrent users. With 1800 users, the 90% of the users finished logging in within 5 secs while with 9000 users, the 90% line is 126.07 secs.

## Acknowledgment

We would like to thank our colleagues and students for their suggestions and comments.

## References

[1] D. Menascé and others, “Load testing of web sites,” *Internet Computing, IEEE*, vol. 6, no. 4, pp. 70–74, 2002.
[2] M. D. Anderson, “The top 13 mistakes in load testing applications,” *Software Testing and Quality Engineering Magazine*, vol. 1, no. 5, pp. 30–41, 1999.
[3] J. Shaw, “Web application performance testing—a case study of an on-line learning application,” *BT Technology Journal*, vol. 18, no. 2, pp. 79–86, 2000.
[4] W. D. Smith, *TPC-W: Benchmarking an ecommerce solution*. 2000.
[5] C. Amza, A. Chanda, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, “Specification and implementation of dynamic web site benchmarks,” in *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, 2002, pp. 3–13.
[6] M. F. Arlitt and C. L. Williamson, “Internet web servers: Workload characterization and performance implications,” *IEEE/ACM Transactions on Networking (ToN)*, vol. 5, no. 5, pp. 631–645, 1997.
[7] “Apache JMeter - Apache JMeter$^{TM}$.” [Online]. Available: http://jmeter.apache.org/. [Accessed: 05-Nov-2015].
[10] Systemone Operations Manual.2015.
[11] “AWStats - Free log file analyzer for advanced statistics (GNU GPL).” [Online]. Available: http://www.awstats.org/. [Accessed: 06-Nov-2015].
[12] “Using Apache Bench and getting meaningful results.” [Online]. Available: http://tales.itnobody.com/2011/12/ab-apache-bench-understanding-and-getting-tangible-results.html. [Accessed: 07-Nov-2015].
[13] “Developers Corner: Maximum concurrent connections to the same domain for browsers.” [Online]. Available: http://sgdev-blog.blogspot.sg/2014/01/maximum-concurrent-connection-to-same.html. [Accessed: 07-Nov-2015].
[14] “Slashdot effect - Wikipedia, the free encyclopedia.” [Online]. Available: https://en.wikipedia.org/wiki/Slashdot_effect. [Accessed: 10-Nov-2015].
[15] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. E. Long, “Managing flash crowds on the Internet,” 2003, pp. 246–249.
[16] [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.txt. [Accessed: 10-Nov-2015].
[17] “memcached - a distributed memory object caching system.” [Online]. Available: http://memcached.org/. [Accessed: 12-Nov-2015].
[18] “Simple Cloud Infrastructure for Developers | DigitalOcean.” [Online]. Available: https://www.digitalocean.com/. [Accessed: 12-Nov-2015].
[19] “CloudFlare | CloudFlare | The web performance & security company.” [Online]. Available: https://www.cloudflare.com/. [Accessed: 12-Nov-2015].
[20] UPLB Information Technology Center.

[21] “How To Use JMeter To Record Test Scenarios | DigitalOcean.” [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-jmeter-to-record-test-scenarios. [Accessed: 20-Nov-2015].

NOTES:

[8] “ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.2.” [Online]. Available: https://httpd.apache.org/docs/2.2/programs/ab.html. [Accessed: 05-Nov-2015].

**Simulating Denial-of-Service (DoS) Attacks**
DoS sends request at vast speeds and amount that it prevents legitimate users from accessing S1. We simulate DoS by running Apache Bench(ab)[8] simultaneously while running the JMeter test plan. ab runs access the index page of S1 only to simulate directed attacks.

| # of users/threads | Pages | Hits | Bandwidth (KB) | Average Response Time (ms) | 90th Percentile (ms) | Average Throughput (req/sec) | Average Throughput (KB/sec) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 77.77 | 477 | 795 | 2.1 | 26.7 |
| 10 | 10 | 60 | 721.79 | 1898 | 4856 | 3.9 | 50.95 |

Table 1. Machine configuration for servers used in the study.

| Hostname | Flavor | Purpose | IP Addresses |
|---|---|---|---|
| s1apache-sfrc2 | Ubuntu-14.04-server-amd64,<br>1 VCPU, 512MB RAM, 20GB Disk | 2nd Apache/PHP5 | 192.168.0.15 |
| s1nginx-sfrc | Ubuntu-14.04-server-amd64,<br>1 VCPU, 1GB RAM, 20GB Disk | NGINX as proxy/load balancer | 192.168.0.14<br>10.0.3.247 |
| s1apache-sfrc | Ubuntu-14.04-server-amd64,<br>1 VCPU, 512MB RAM, 20GB Disk | 1st Apache/PHP5 server | 192.168.0.13 |
| s1db-sfrc | Ubuntu-14.04-server-amd64,<br>2 VCPU, 1.5GB RAM, 20GB Disk | MySQL DB server | 192.168.0.12<br>10.0.3.245 |

## Hardware

***Single Server (Rodolfo)***

| | |
|---|---|
| **RAM** | 32 GB |
| **Hard Disk** | 53 GB w/ RAID 1<br>INTEL SSDSC2CT06 (SSD) |
| **CPU** | 8 X Intel(R) Xeon(R) CPU<br>E5640 @ 2.67GHz |

**Software**
*Technology Used*

| | |
|---|---|
| **WebServer** | Apache 2.2.22-13 |
| **Database** | MySQL<br>5.5.31+dfsg-0+wheezy1 |
| **Operating System** | Debian 7.1 Wheezy |
| **Scripting Language Used** | PHP 5.4.4-14+deb7u2 |

The SystemOne Application has the following user roles:
- Administrator
- Department Head
- Enlistor
- Student

The SystemOne Application currently has the following modules/pages:
**All Users**
- Home Page
- Index
- Search
- Login(?)

**Students**
- Manage Profile
- Dream Schedule
- Problem Resolution
- Straggler’s Time
- Change of Matriculation
- View Mini Checklist

**Enlistor**
- Enlistor App
- View Mini Checklist

**Department Head**

- Add Slots
- Classlist

**Administrator**
- Problem Manager
- Add Student
- Menu Manager
- Account Manager
- Role Manager
- Manage Announcements
- Content Manager
- Site Status
- Search Student

**Testing Tools**
- JMeter
- Profiler
- MySQL Tuner

First we create a baseline measurement using a single client accessing the application. We measure the response time when a user access each module. blah blah blah

| Machine Name | Flavor | Purpose | IPaddress |
|---|---|---|---|
| s1apache-sfrc2 | Ubuntu-14.04-server-amd64,<br>1 VCPU, 512MB RAM, 20GB Disk | 2nd Apache/PHP5 | 192.168.0.15 |
| s1nginx-sfrc | Ubuntu-14.04-server-amd64,<br>1 VCPU, 1GB RAM, 20GB Disk | NGINX as proxy/load balancer | 192.168.0.14<br>10.0.3.247 |
| s1apache-sfrc | Ubuntu-14.04-server-amd64,<br>1 VCPU, 512MB RAM, 20GB Disk | 1st Apache/PHP5 server | 192.168.0.13 |
| s1db-sfrc | Ubuntu-14.04-server-amd64,<br>2 VCPU, 1.5GB RAM, 20GB Disk | MySQL DB Server | 192.168.0.12<br>10.0.3.245 |