# Performance Testing SAS

The document shows the performance tests that were executed against the SAS application. Performance tests were executed against the backend part of the application.

Tool: Apache JMeter 5.4.3

## Environment

Both the backend and the MySQL database were containerized in docker containers.
The containerized system and the performance test tool JMeter were both running on the same host machine but with limited CPU and memory access.

Host Machine
- OS: MacOS
- CPU: M1 Pro (10 Cores)
- RAM: 16GB

Containers configurations:
Backend:
- CPU: 1
- RAM: 1000 MB

MySQL:
- CPU: 1
- RAM: 1000 MB

## Test plans

### Load Test

According to the non-functional requirements performance specifications, the accepted performance requirements for the system were specified for a load of 5.500 users requesting to mark attendance over a period of 15 seconds.

Performance tests endpoints were set up for a more accurate load test, where teachers would create a lecture and start the attendance process, generating a valid code for the students.

JMeter configurations:

Load Test Teacher Thread Group:

Thread group's purpose was mainly to generate 110 valid lecture codes to properly load test the system. First creating the lectures and the valid codes in the system, allows us to have a more accurate scenario for the student load test.

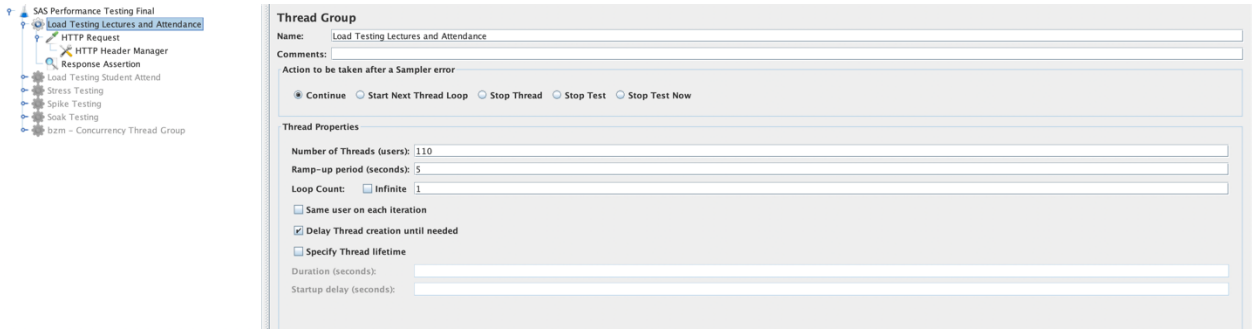



*Figure 1*

Figure 1: Teacher Load Thread Group Configuration.

Load Test Student Thread Group:
Figures 2 and 3, show the configuration for the student load testing thread group, where the total requests amount to 5500 over a period of 19 seconds. Simulating a scenario where all students in KEA would attend a lecture within those 19 seconds. The timer in Figure 3, creates some randomness for when the requests are being sent to the server.

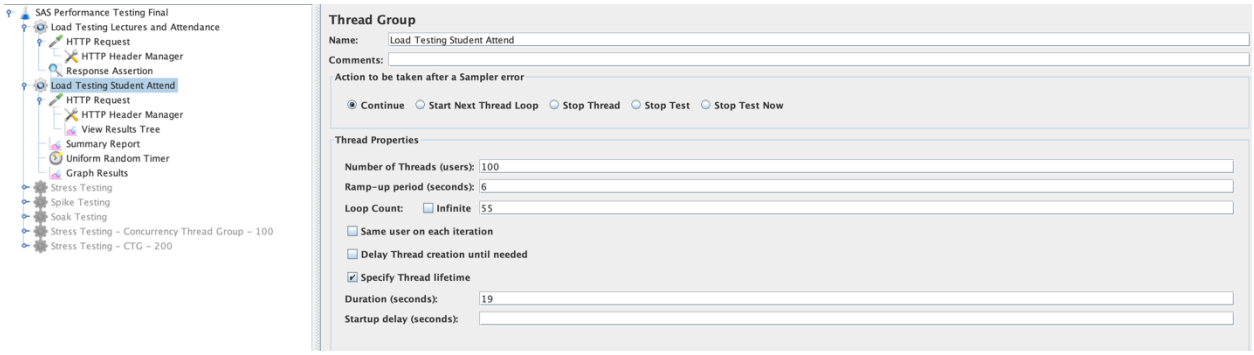



*Figure 2*

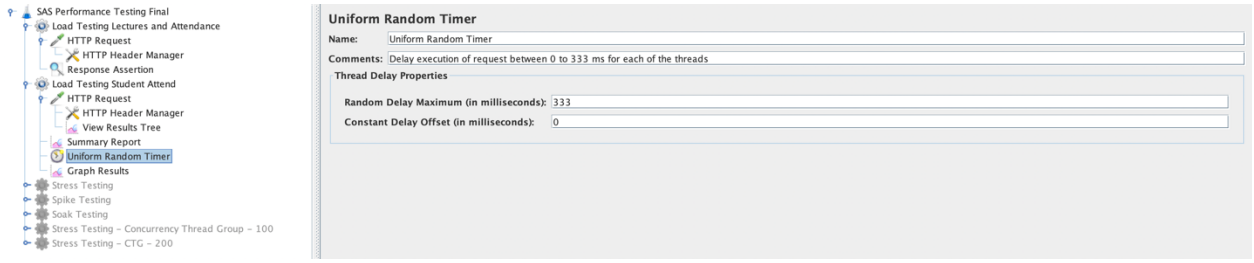



*Figure 3*

Results:

Results of the load test can be seen in Figure 4:

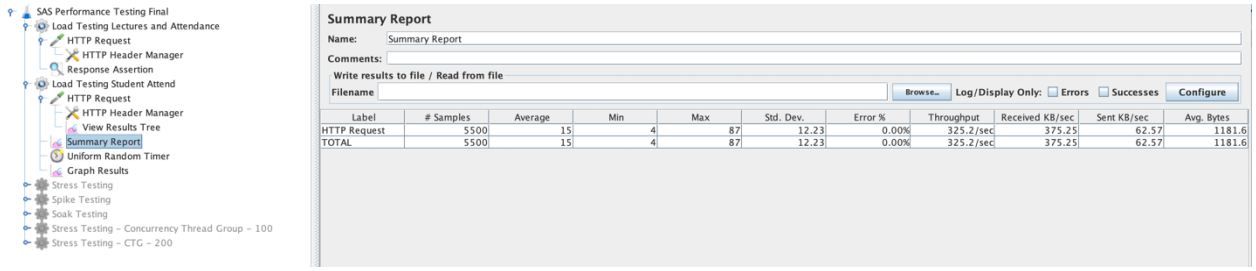



*Figure 4*

## Stress Test

To find the limitations of the system, we decided to use a different type of thread group called “bzm – Concurrency Thread Group” that allows us to test how the system would handle X amounts of concurrent threads sending requests to our server.
Starting with an arbitrary number of 100 threads, and decided to work our way up from there, holding the concurrent threads sending requests for 1 minute until the accepted performance requirements are no longer met.

### 100 Concurrent Threads

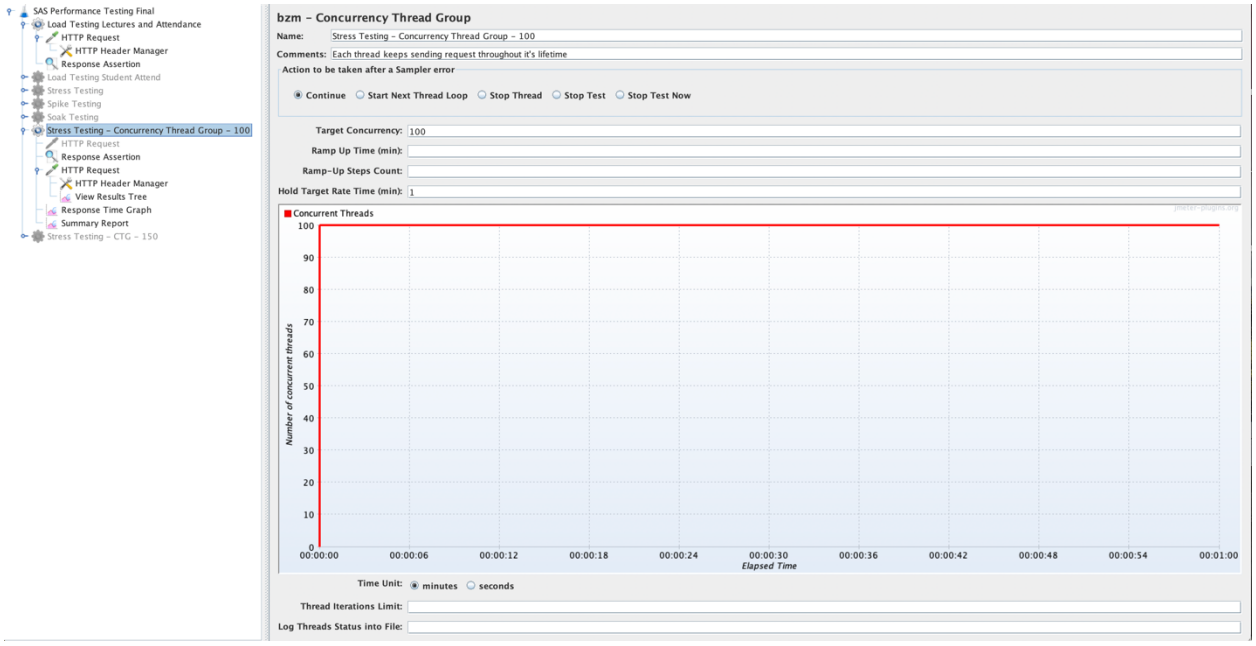



*Figure 5*

### 100 Concurrent Threads Results:

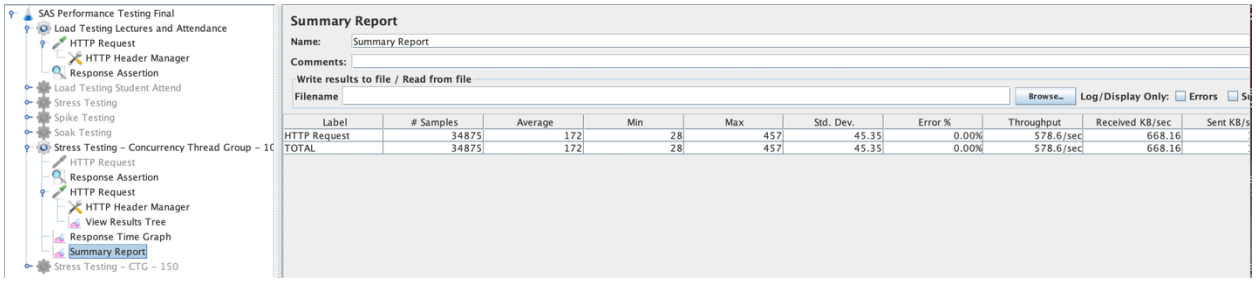



*Figure 6*

Figure 6 shows the “Summary Report” where we can see the total requests in the column “Samples”. The report shows that for one minute, there were 34.875 requests sent to the server with an average response time of 172ms.

Figure 7 shows the “Response Time Graph”, where interval (dot in the graph) is set to 250ms.

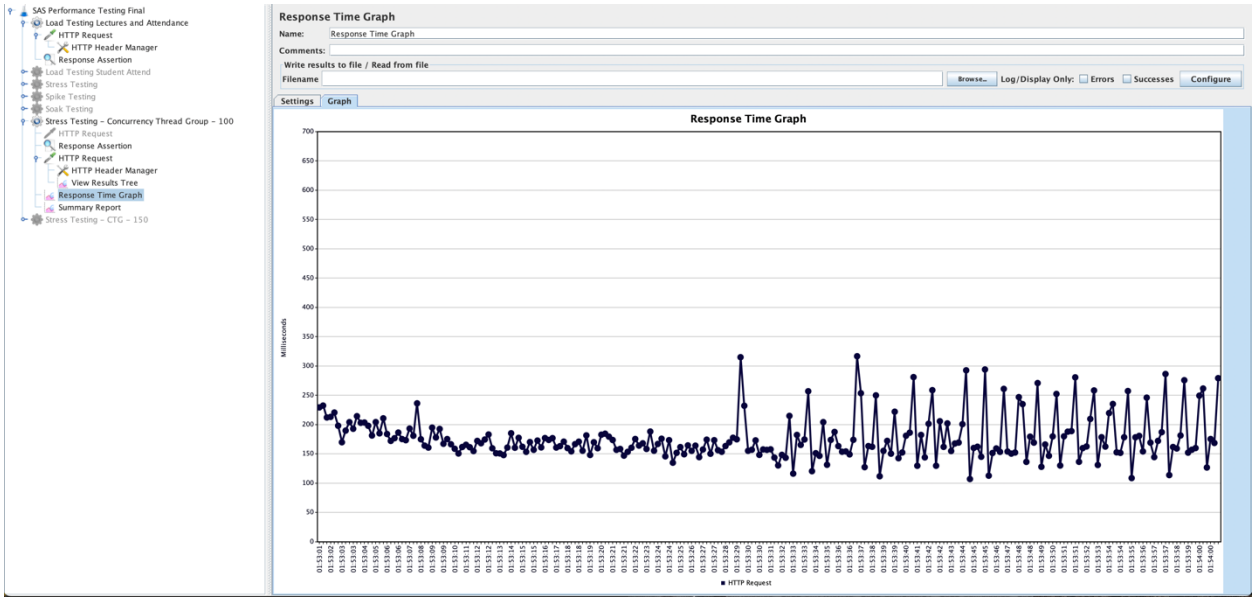



*Figure 7*

Because the system held up quite well in this test, we decided to try another stress test with 200 concurrent threads.

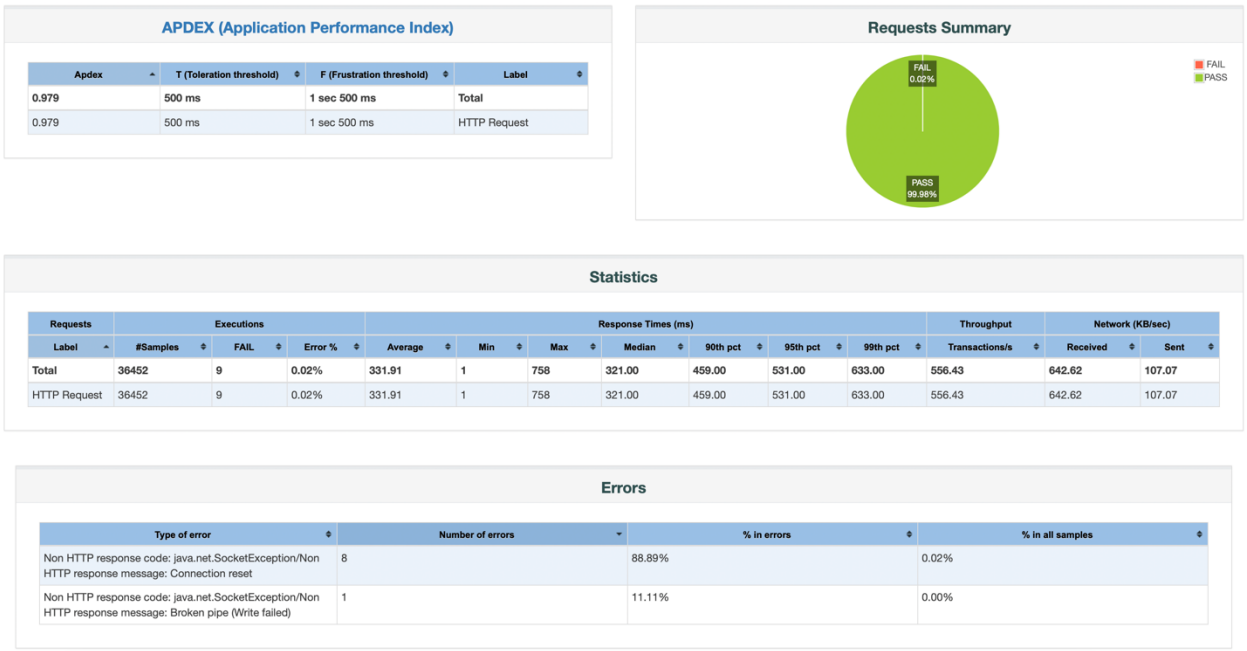
200 Concurrent Threads

We performed three tests in total with 200 concurrent threads, where each test differed slightly:

1. “Hold Target Rate Time” set to 1 minute, executed using the JMeter GUI.
2. “Hold Target Rate Time” set to 1 minute, executed through the command line.
3. “Hold Target Rate Time” set to 3 minutes, executed through the command line.

In test scenario 1, we were not able to get any results from the “Response Time Graph”, therefore tried scenario 2 and generated an html report from the test result file generated by JMeter.

**Scenario 2 - html report snippets**:



APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.979 | 500 ms | 1 sec 500 ms | Total |
| 0.979 | 500 ms | 1 sec 500 ms | HTTP Request |

Requests Summary



Statistics

| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| Total | 36452 | 9 | 0.02% | 331.91 | 1 | 758 | 321.00 | 459.00 | 531.00 | 633.00 | 556.43 | 642.62 | 107.07 |
| HTTP Request | 36452 | 9 | 0.02% | 331.91 | 1 | 758 | 321.00 | 459.00 | 531.00 | 633.00 | 556.43 | 642.62 | 107.07 |

Errors

| Type of error | Number of errors | % in errors | % in all samples |
|---|---|---|---|
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset | 8 | 88.89% | 0.02% |
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Broken pipe (Write failed) | 1 | 11.11% | 0.00% |

*Figure 8*

Figure 8 shows an overview of the results.
Figure 9 shows the response time percentiles, where we can see that our performance requirements are no longer met. The graph shows that 91,1% of the requests had a response time of 400ms or higher, while our requirements were that 99% of the requests should have a maximum of 400ms response time when tested locally.

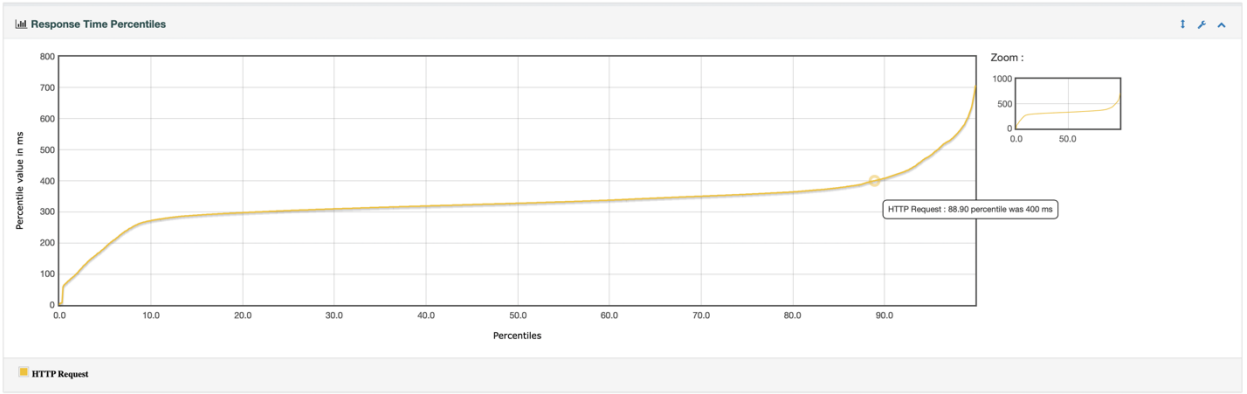



*Figure 9*

Figure 10 is the reason we decided to test again, but while holding the thread target rate for a little bit longer, as the response time over time shows a steady linear decline. However it is misleading because each point in the graph is a single minute, so we were curious to see what it would look like with two more data points.

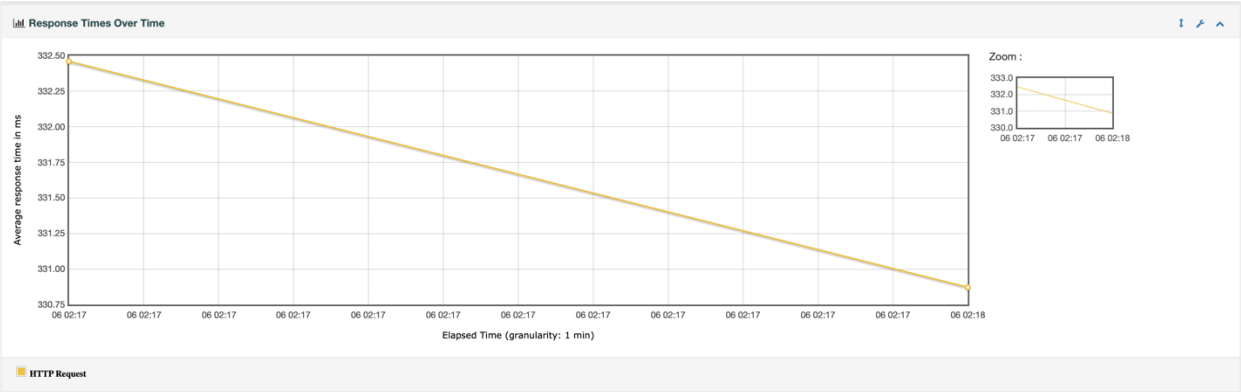
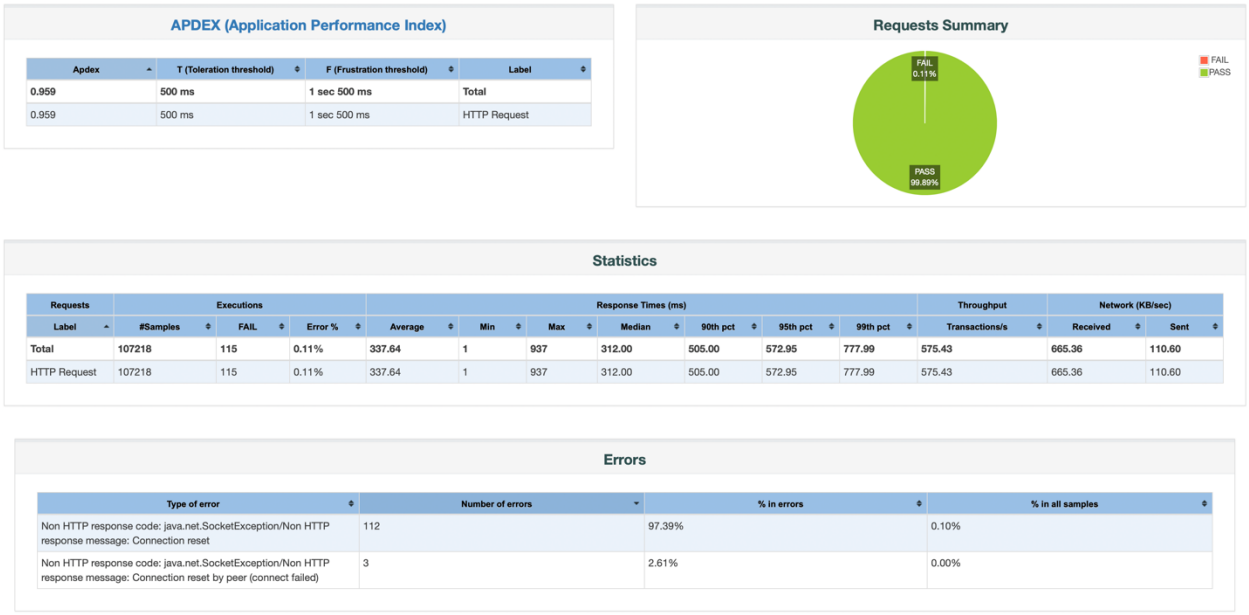



*Figure 10*

**Scenario 3 – html report snippets:**



APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.959 | 500 ms | 1 sec 500 ms | Total |
| 0.959 | 500 ms | 1 sec 500 ms | HTTP Request |

Requests Summary

FAIL 0.11%
PASS 99.89%

Statistics

| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| Total | 107218 | 115 | 0.11% | 337.64 | 1 | 937 | 312.00 | 505.00 | 572.95 | 777.99 | 575.43 | 665.36 | 110.60 |
| HTTP Request | 107218 | 115 | 0.11% | 337.64 | 1 | 937 | 312.00 | 505.00 | 572.95 | 777.99 | 575.43 | 665.36 | 110.60 |

Errors

| Type of error | Number of errors | % in errors | % in all samples |
|---|---|---|---|
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset | 112 | 97.39% | 0.10% |
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset by peer (connect failed) | 3 | 2.61% | 0.00% |

*Figure 11*

Figure 11 shows the test results overview.

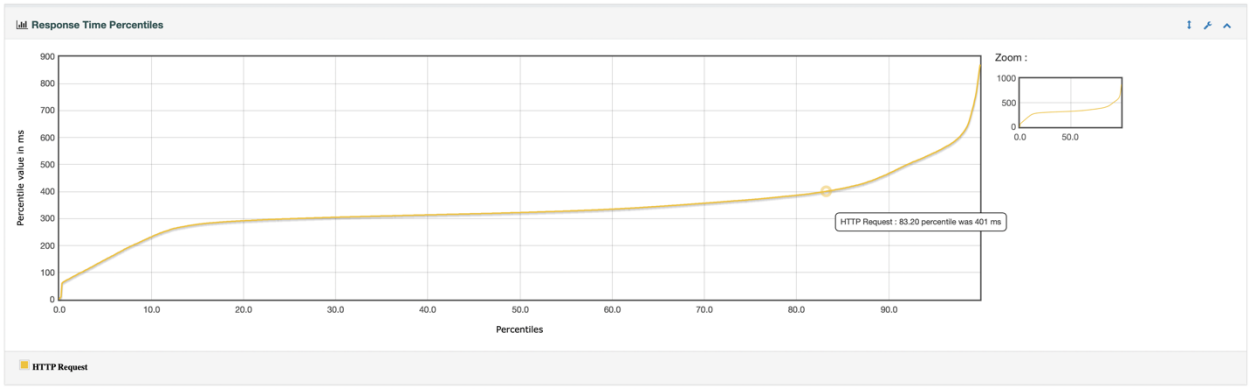



*Figure 12*

In Figure 12, we can see that running the test for two minutes longer, our performance requirements are now met by only 83,2% of the requests.

In Figure 13, we can see the response time during the second and third minutes:
First minute avg response time: 334ms
Second minute avg response time: 343ms
Third minute avg response time: 336ms

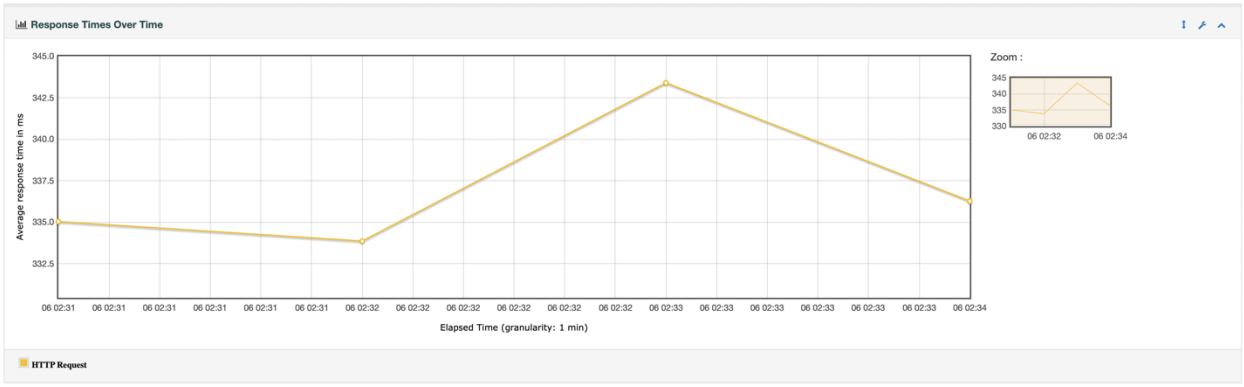



*Figure 13*

## Spike Test

*While stress testing the system, the test results indicated that the system could handle a load of more than six times the expected load while meeting the accepted performance requirements for 83,2% of the requests. We consider the stress test to have covered our spike test scenario, especially since we don't expect a spike in usage to be that high, and for more than a few seconds.*

For our spike testing, we used another thread group called "Ultimate Thread Group" where we could configure different groups of threads and have them run on different schedules.

In Figure 14, we can see the configuration of the Ultimate Thread Group, where we start with a normal load of 100 threads as we did in the load test, and we set that group of threads to run for 180 seconds. Then there is our spike thread group that starts running 65 seconds after the other one, with a 1000 threads, and it will run for 20 seconds, simulating a spike in users sending requests to our server.

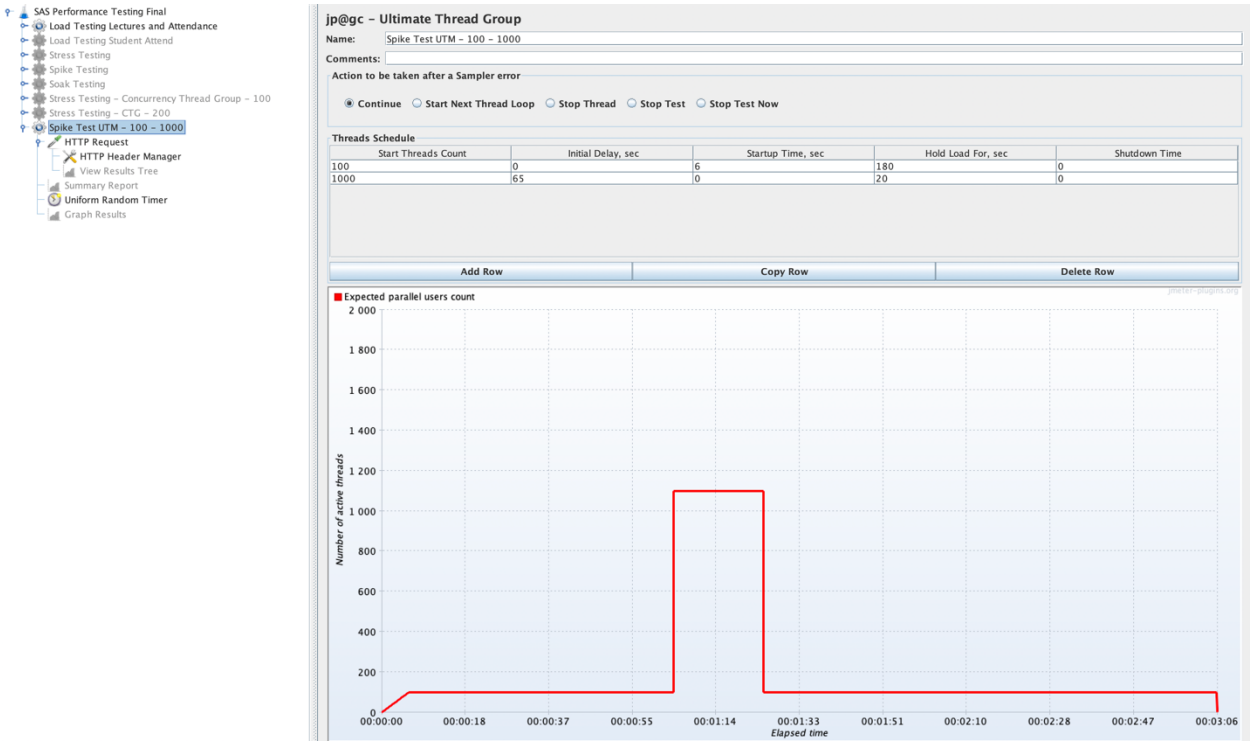
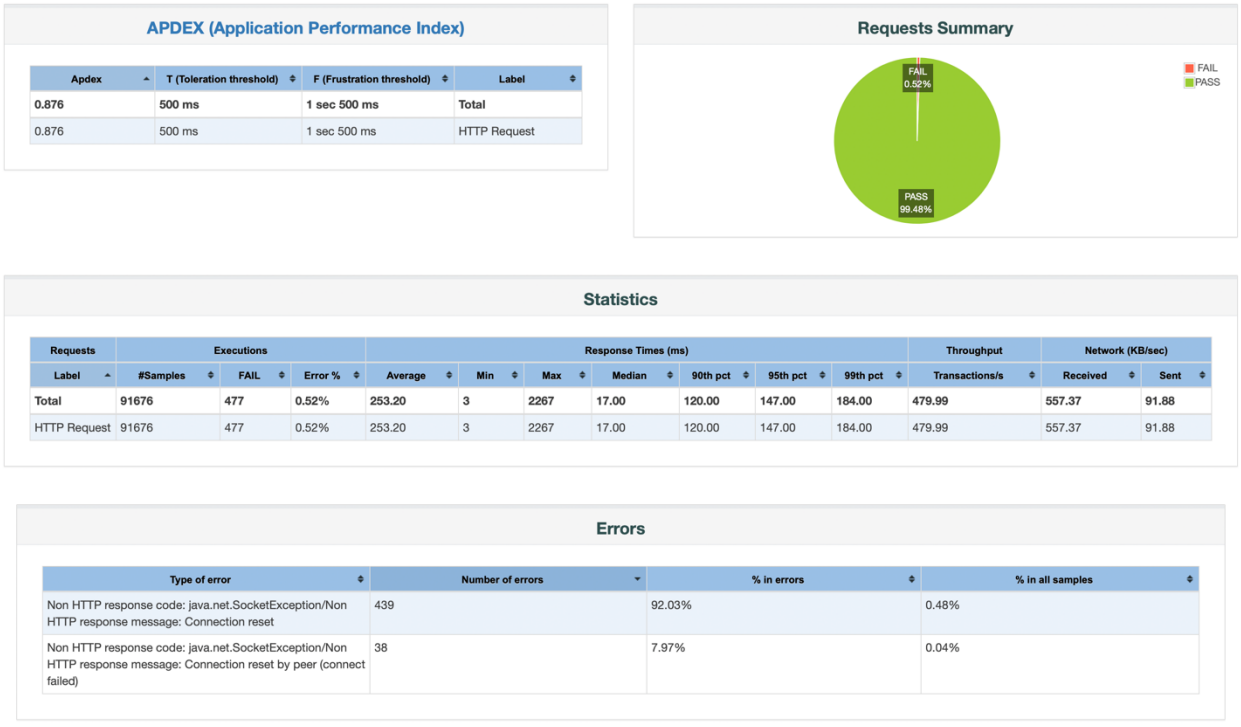



*Figure 14*

Figure 15 shows the overview of the test, from a generated html report of the test, and we can see that there were 91676 requests sent to our server, where 99,48% of them were successful. Average response time was 235ms.



APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.876 | 500 ms | 1 sec 500 ms | Total |
| 0.876 | 500 ms | 1 sec 500 ms | HTTP Request |



Statistics

| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| Total | 91676 | 477 | 0.52% | 253.20 | 3 | 2267 | 17.00 | 120.00 | 147.00 | 184.00 | 479.99 | 557.37 | 91.88 |
| HTTP Request | 91676 | 477 | 0.52% | 253.20 | 3 | 2267 | 17.00 | 120.00 | 147.00 | 184.00 | 479.99 | 557.37 | 91.88 |

Errors

| Type of error | Number of errors | % in errors | % in all samples |
|---|---|---|---|
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset | 439 | 92.03% | 0.48% |
| Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset by peer (connect failed) | 38 | 7.97% | 0.04% |

*Figure 15*

Figure 16 shows a steep increase during the second minute of the test, showing lower response time with the increase of threads.

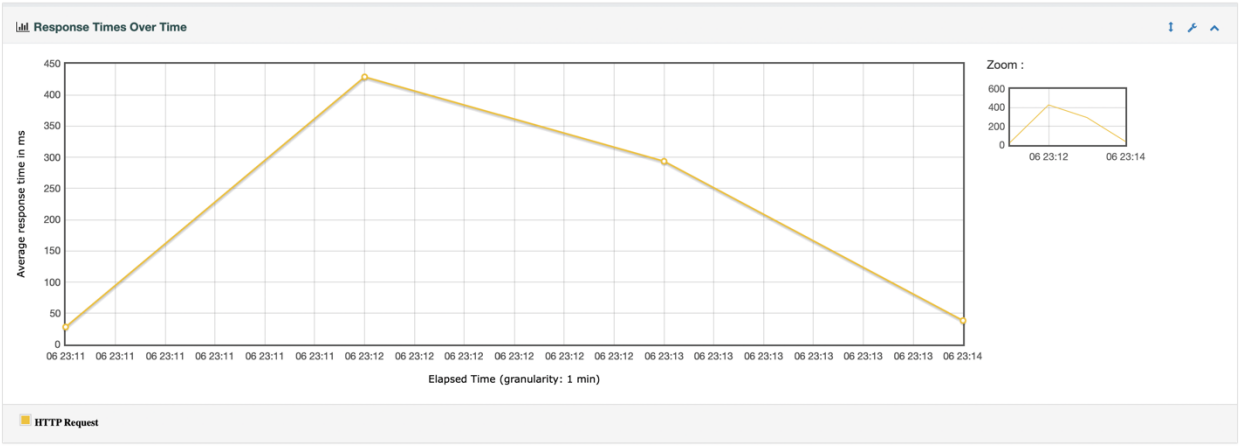



*Figure 16*

Figure 17 shows us that 14,1% of the threads had a response time that did not meet our accepted performance requirement. Interestingly 16,8% of the requests in the stress test were not meeting the requirements, showing us that with 200 concurrent threads we had a slower

response time than with this test, where supposedly we have 1100 concurrent threads running for a 20 second period.

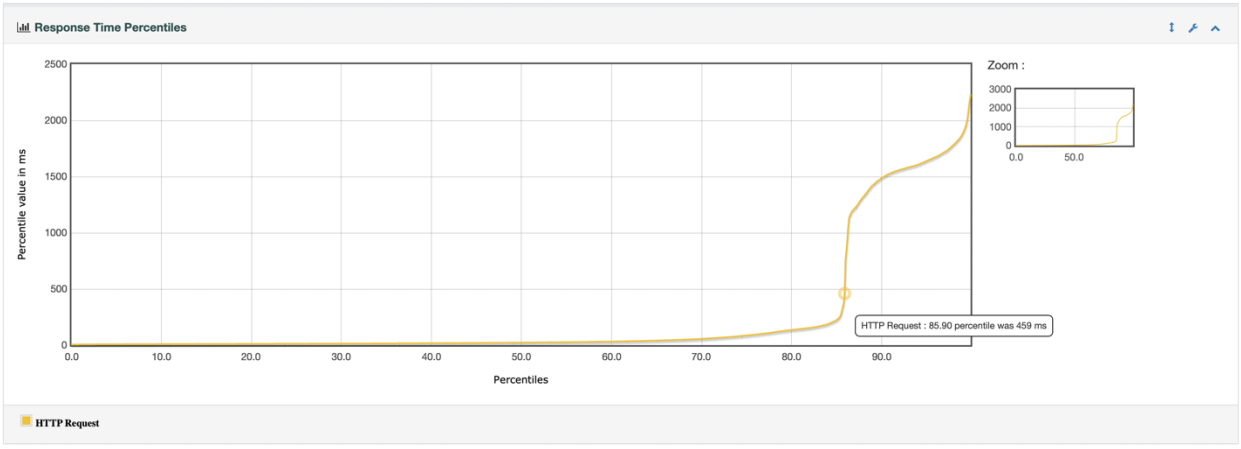



*Figure 17*

Figure 18, again shows a spike in how long it takes for the thread to receive a response.

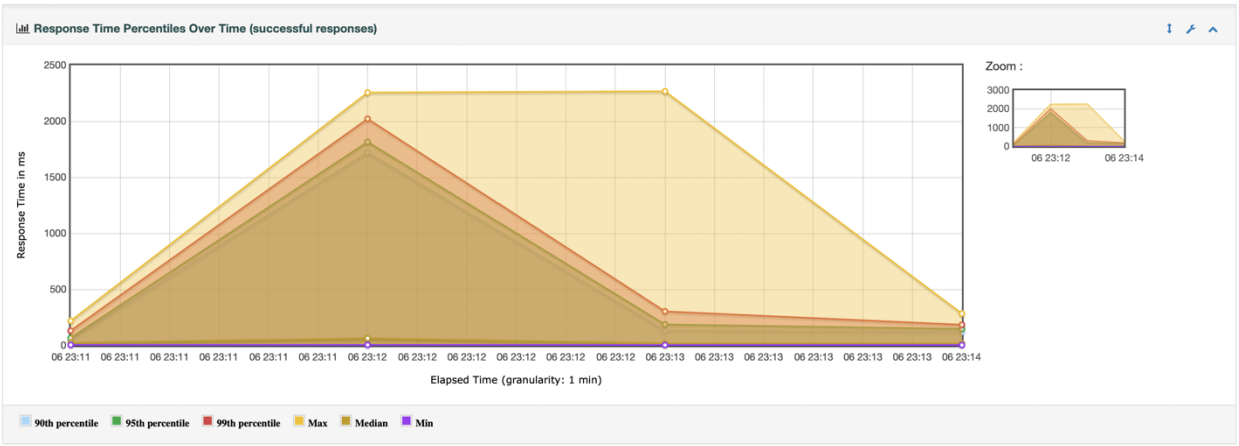



*Figure 18*

Looking at the graph here in Figure 19, we can see that we did not reach anywhere near 1100 concurrent threads, but our top thread count was 372 threads.

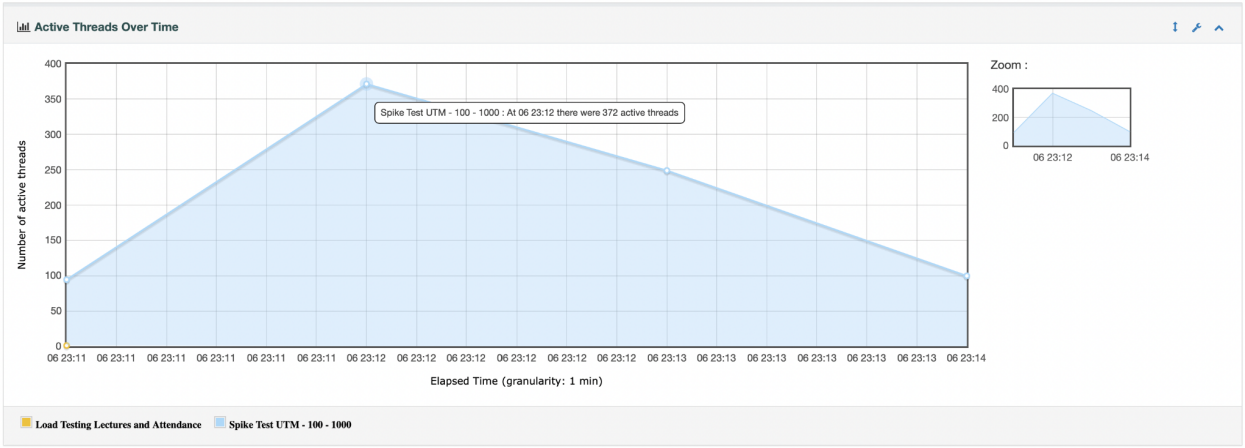

Active Threads Over Time
Number of active threads
400
350
300
250
200
150
100
50
0
Spike Test UTM - 100 - 1000 : At 06 23:12 there were 372 active threads
06 23:11 06 23:11 06 23:11 06 23:11 06 23:11 06 23:11 06 23:12 06 23:12 06 23:12 06 23:12 06 23:12 06 23:12 06 23:13 06 23:13 06 23:13 06 23:13 06 23:13 06 23:13 06 23:14
Elapsed Time (granularity: 1 min)
Zoom :
400
200
0
06 23:12
06 23:14
Load Testing Lectures and Attendance
Spike Test UTM - 100 - 1000

*Figure 19*