# 1. Functional testing

## Approach

Functional testing phase was focused on application behavior correctness. Test cases were designed to find run-time errors and memory leaks. Tests were manual.

Tools listed below have been used for investigation:

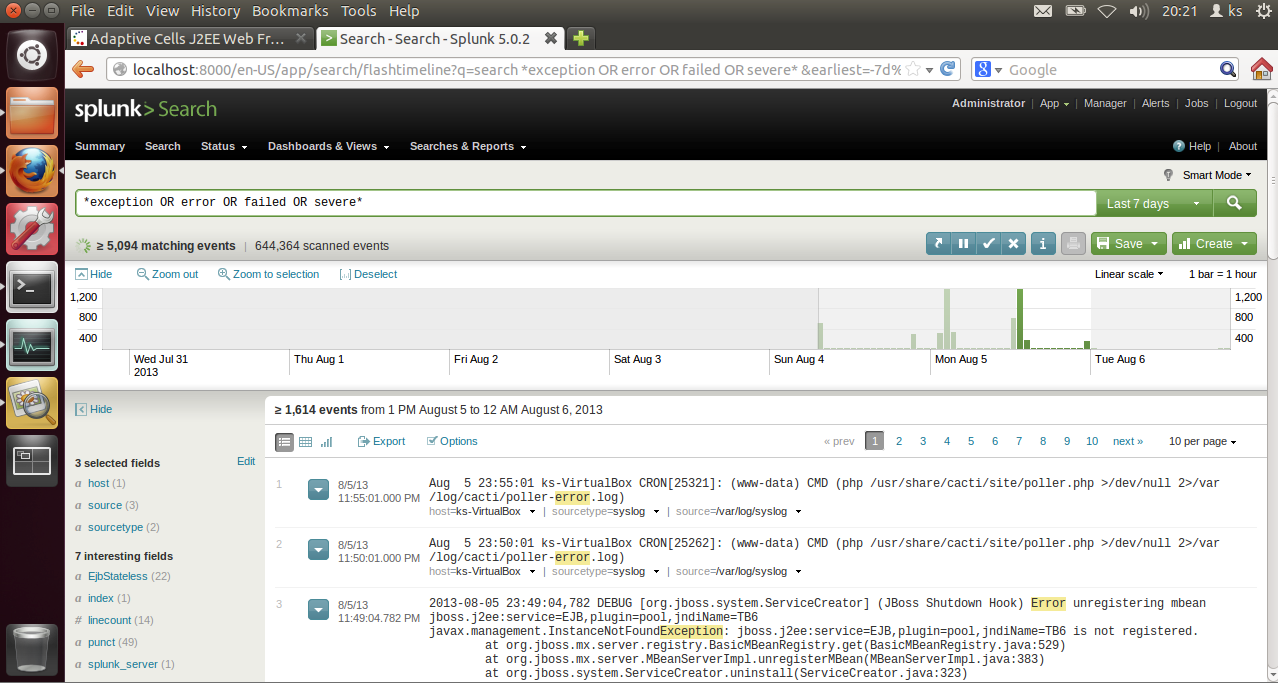
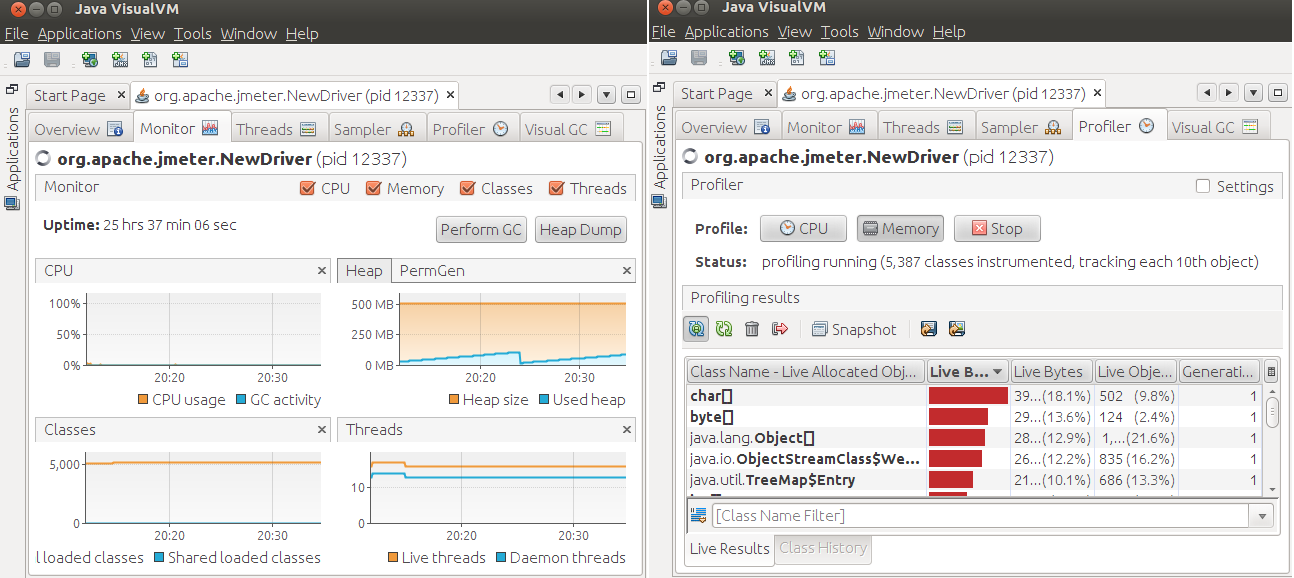
* Splunk - log mining tool for search, analysis and visualization

Figure 1: Splunk query results screenshot.

* Visualvm – JVM monitoring and profiling tool

Figure 2: Visualvm monitoring and profiling views.

Tests cases were executed against every AdaptiveCells/J configuration available (config1...10).

Jboss server and boot logs (JBOSS/server/default/log/server.log, boot.log and access.log) were indexed by Splunk during tests.

For each config, 2 types of test templates were used:

1. run-time exceptions search:

* start Jboss and let it warm up
* start and attach Visualvm
* run config(1...10) once
* repeat every 2 minutes for 3-4 times
* run config(1…10) multiple times
* detach Visualvm
* stop Jboss
* analyse logs using Splunk query that is searching for following keywords in logs:

“exception OR error OR failed OR severe OR ( sourcetype=access\_\* ( 404 OR 500 OR 503 ) )”

1. memory leak recognition:

* start Jboss and let it warm up
* start and attach Visualvm
* start Visualvm memory profiler tracking every object allocation and stack traces
* switch off profiler results automatic refreshing
* kick off garbage collection
* clean all profiler results

\*\*\* at this point object instances are garbage collected and only new allocations will be displayed in profiler results \*\*\*

* execute config(1…10) several times
* kick off garbage collection again
* results contains objects that were not cleaned up properly
* create heap dump
* focus on objects that survived most garbage collections – generations metric in profiler
* for suspicious objects find and investigate stack traces in profiler and find references to instances in heap dump

## Results

Before starting any testing executed ‘run-time exceptions search’ test case to find which exceptions already exists before application deployment. Tested: clean Jboss, with Visualvm attached, with memory profiler running and with G9.ear deployed. Identified numerous exceptions thrown by server before any config execution. Highlights:

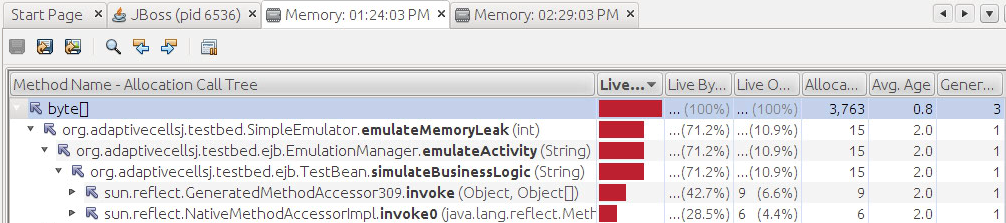
* javax.management.InstanceNotFoundException: jboss.j2ee:service=EJB,plugin=pool,jndiName=TB6 is not registered. thrown by org.jboss.mx.server.registry.BasicMBeanRegistry.get(BasicMBeanRegistry.java:529) – affects all TB (1 to 6) instances
* numerous java.sql.SQLException: Table already exist, java.sql.SQLException: Index already exist, java.sql.SQLException: Violation of unique constraint SYS\_PK\_48: duplicate value(s) for column(s) $$
* java.lang.ClassNotFoundException: org.jboss.mx.server.MBeanServerBuilderImpl Caused by: javax.management.JMRuntimeException: Failed to load MBeanServerBuilder
* java.lang.IllegalArgumentException: Property is not readable: propertyReplace for org.jboss.beans.metadata.plugins.AbstractPropertyMetaData

All of those types of exceptions were investigated for every test case run and excluded from further analysis if no new instance of a given type found.

### Config 1

No application specific run-time exceptions found.

Found memory leak. Identified suspicious object that was the biggest one on the heap and lived for 3 generations: byte[]

Figure 3: Visualvm memory profiler results – allocation call tree.

Suspected method: org.adaptivecellsj.testbed.SimulateEmulator.emulateMemoryLeak(int).

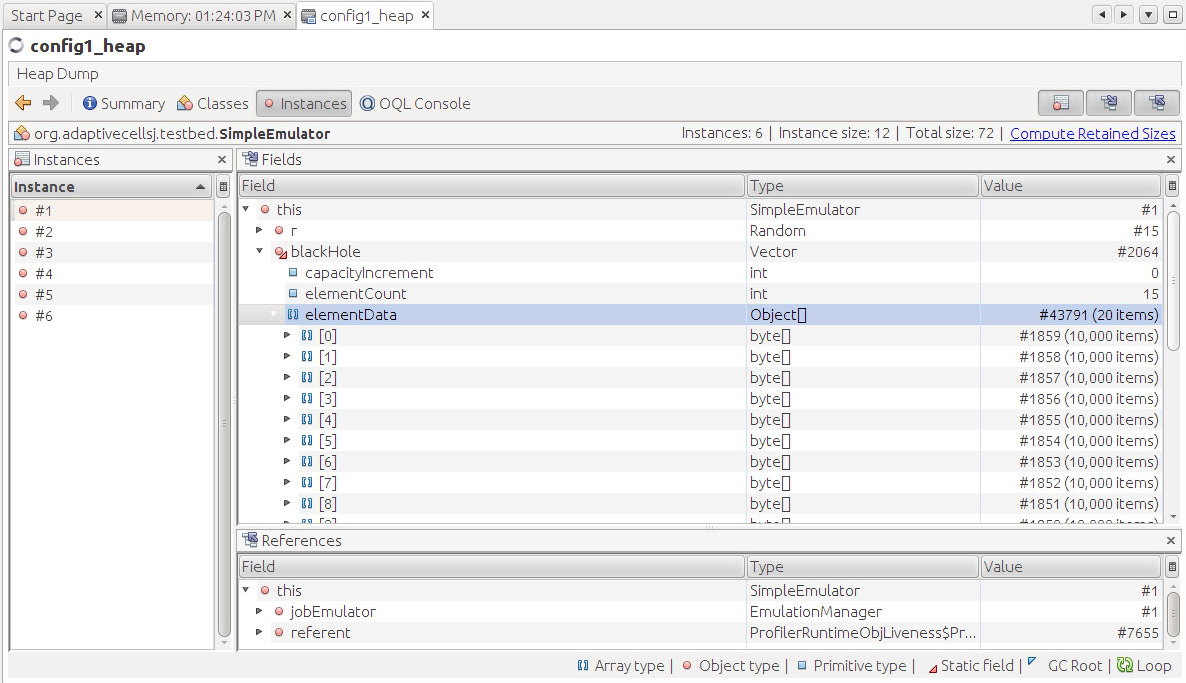
Therefore, checked for SimpleEmulator object references in heap dump. Found 6 instances:

Figure 4: Heap dump view in Visualvm

All of SimpleEmulator instances contains a reference variable to object Vector blackHole that is encapsulating references to several byte[]objects that were not garbage collected properly. Those are the leaked objects.

Exactly the same instances of SimpleEmulator and byte[] objects leaked in Config 2 and Config 7.

### Config 4

Found run-time exception thrown directly in the browser:

The Adaptive Cells EJBs have raised the exception:

null; nested exception is: java.lang.RuntimeException

You might have configured the cells to raise exceptions. This can be done adding an environment entry (namely configXexception) to the deployment descriptors of the cell EJBs.

Found the same exception in Jboss server.log:

TransactionRolledBackException in method: org.adaptivecellsj.testbed.ejb.TestBeanIF.simulateBusinessLogic(java.lang.String) throws java.rmi.RemoteException,java.lang.Exception,

caused by RuntimeException thrown from: org.adaptivecellsj.testbed.SimpleEmulator.emulateException(SimpleEmulator.java:69)

Exception occurrences in logs are aligned with Config 4 execution times from browser.

Exactly the same exception occurred for Config 9.

No memory leaks found.

Summary for all findings:

|  |  |  |
| --- | --- | --- |
|  | Run-time error | Memory leak |
| Config 1 |  | x |
| Config 2 |  | x |
| Config 3 |  |  |
| Config 4 | x |  |
| Config 5 |  |  |
| Config 6 |  |  |
| Config 7 |  | x |
| Config 8 |  |  |
| Config 9 | x |  |
| Config 10 |  |  |

Table 1: Functional testing findings.

Only Configs: 3, 5, 6, 8, 10 will be included in further testing.

Raw data and detailed results from testing can be found:

* pre config execution results: <http://bit.ly/11Ku8kD>
* full results: <http://bit.ly/191s1uS>
* raw data: <http://bit.ly/11LGol1>

# 2. Load testing

## Approach

Load testing was targeted to characterize application performance trends during average load.

To simulate and monitor realistic user load following tools were used:

* Jmeter - Java application designed to load test functional behavior and measure performance

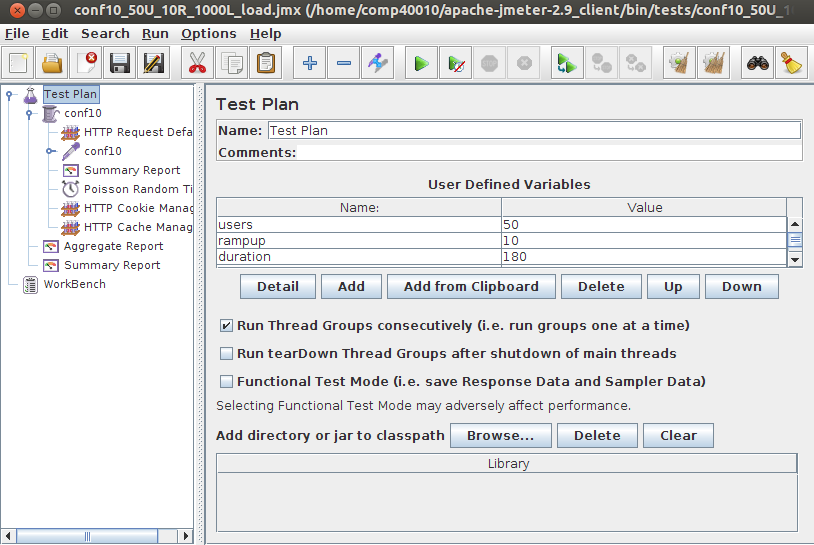


Figure 5: Jmeter test plan.

* Cacti – GUI for RRDtool that logs operating system performance data (CPU, Memory, Processes, Load average, Network in/out) in time series

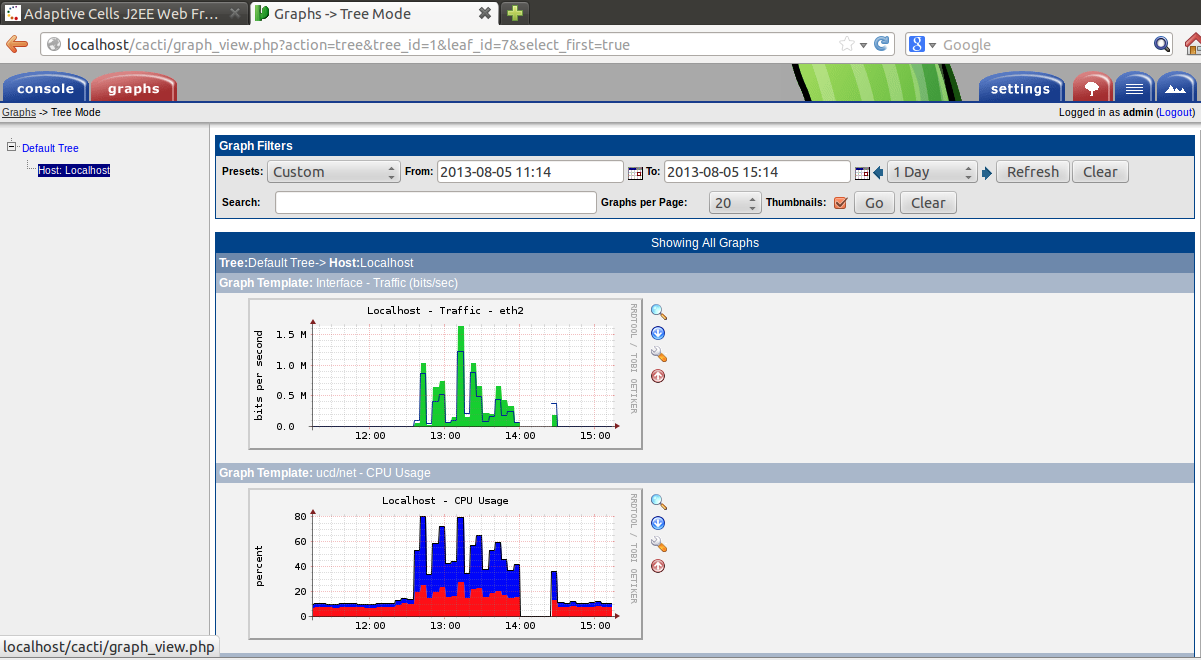


Figure 6: Cacti graphs.

To simulate realistic user load used distributed testing mode in Jmeter. Traffic was spread between 3 client hosts accessing 1 Jboss server.

Visualization of physical and network architecture:

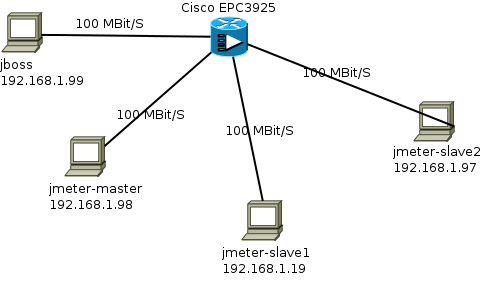


Figure 7: Physical and network architecture of testing infrastructure.

Detailed specification of nodes in table below:

|  |  |
| --- | --- |
| Node | Description |
| Application server – 192.168.1.99 (jboss) | Jboss 5.0.1 GA  2x Intel Core i7 2.9 GHz, 2GB RAM, 100Mbit/s LAN, Ubuntu 12.04.2 LTC x86 |
| Jmeter server – 192.168.1.19 (jmeter-slave1) | Jmeter 2.9  1x Intel Core i5 2.67 GHz, 1GB RAM, 100Mbit/s LAN, Ubuntu 12.04.2 LTC x86 |
| Jmeter server – 192.168.1.97 (jmeter-slave2) | Jmeter 2.9  1x Intel Core i5 2.67 GHz, 1GB RAM, 100Mbit/s LAN, Ubuntu 12.04.2 LTC x86 |
| Jmeter client and server – 192.168.1.98 (jmeter-master) | Jmeter 2.9  2x Intel Core i7 2.9 GHz, 2.5GB RAM, 100Mbit/s LAN, Ubuntu 12.04.2 LTC x86 |
| Routing and switching | Cisco EPC3925 Cisco Residential Gateway |

Table 2: Testing infrastructure specification

Jmeter distributed architecture used for testing:

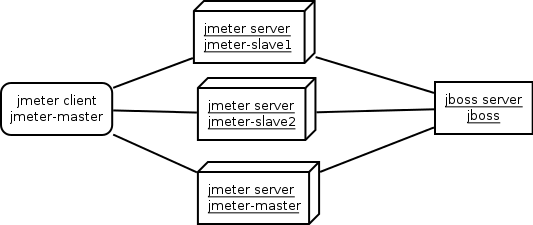


Figure 8: Jmeter testing architecture.

Every configuration was tested using Test Plan design showed on Figure 5. Explanation of test parameters that was used:

* Users – number of threads (concurrent users accessing the system) – 50, 100 and 200
* Ramp-up – period of time to start all specified threads – 10 seconds for average load simulation and 2 seconds for peak load simulation
* Poisson Random Timer – timer that simulates user pauses when accessing the system – this happens due to typing or reading. Timer pauses for minimum 500 milliseconds + random value in 200 milliseconds range. Random value added to respect various speed of typing, reading etc.
* HTTP Cookie and Cache Manager – test elements to simulate real browser behavior – cookie usage and caching. Max number of elements in cache set to 1000.

For traffic visualization refer to <http://bit.ly/1bbGgOv>. Used test plan accessing all of the configs.

Test plan generations have been automated: <http://bit.ly/16z8JNb>. Automation script takes a Jmeter test plan as a reference and populates it with different values of number of threads.

Furthermore, test execution, application monitoring, test results gathering and graph generation also have been automated: <http://bit.ly/13FNQiU>.

Automation script executes following steps for each test plan in specified folder:

* generate a unique named output folder
* start Jboss server
* get Jboss pid and attach Visualvm to this process
* warmup Jboss
* start distributed Jmeter testing
* gather and generate graphs from testing results
* stop Jboss
* gather garbage collection logs for Jboss

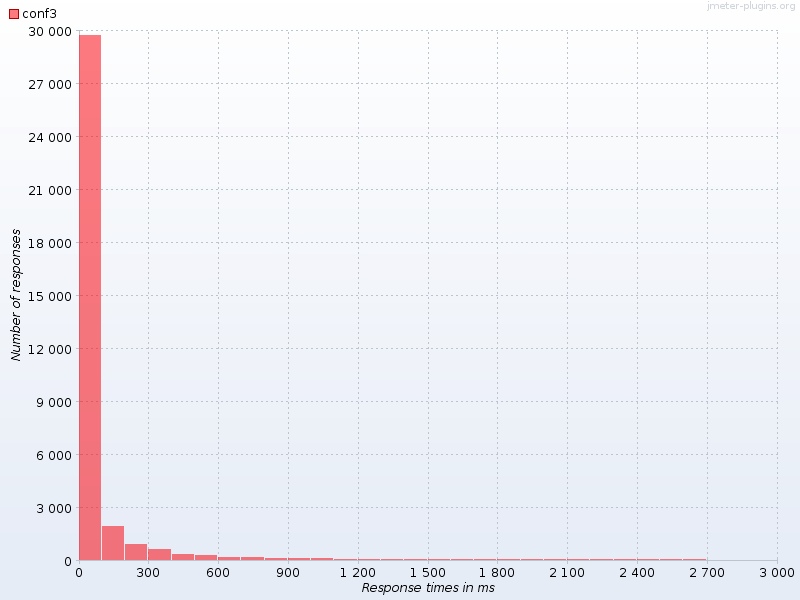
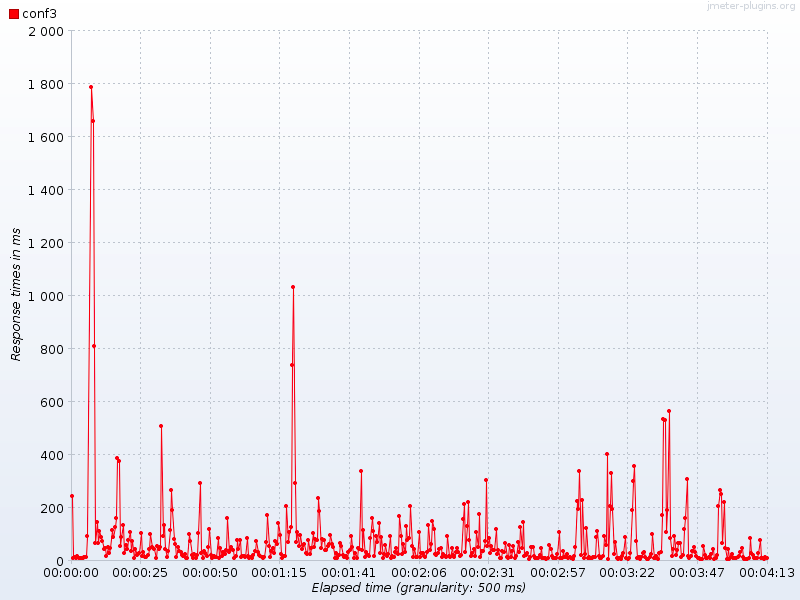
## Results

Following results documentation includes detailed explanation how results were interpreted for Config 3 with 50 concurrent users. The rest of the configs analysis will include only highlights that differentiate certain results from the others. For all configs analysis procedure looked pretty much the same as for Config 3.

### Config 3

#### Number of concurrent users: 50

During load test of application using 50 concurrent users the median response time was 13 ms. For peak simulation response time increased to 16 ms. A majority of response times were below 100 ms as showed on ‘Response times over time’ and ‘Response time distribution’ graphs:

Figure 9: ‘Response times over time’ and ‘Response time distribution’ graphs for Config 3 load testing with 50 concurrent users

As far as CPU is concerned, Jboss was using around 40% for load simulation and spiked couple of times up to 60-80% during peak simulation.

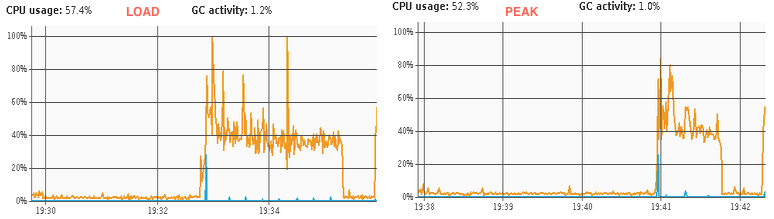


Figure 10: Jboss CPU usage for Config 3 load testing with 50 concurrent users

Operating system overall CPU didn’t exceed 30% - with around 10% being used by operating system itself.

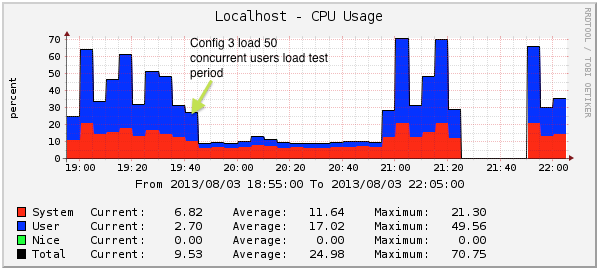


Figure 11: Operating system CPU monitoring for Config 3 load testing with 50 concurrent users

Jboss heap was used in a range from 120MB spiking up to 250 MB during both tests. Graphs show proper garbage collection process where heap comes back to the previous memory usage level:

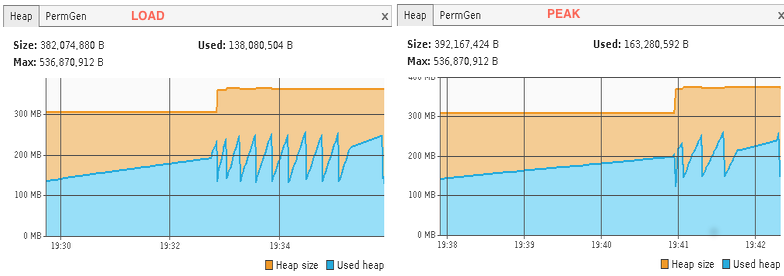


Figure 12: Jboss heap memory usage for Config 3 load testing with 50 concurrent users

Operating system memory had 600 MB free during the test:

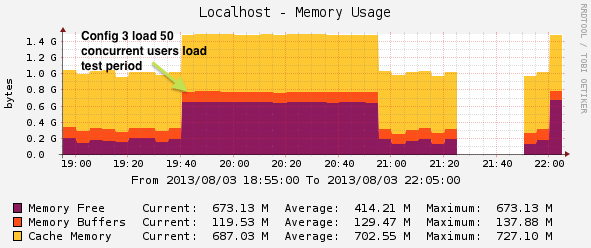
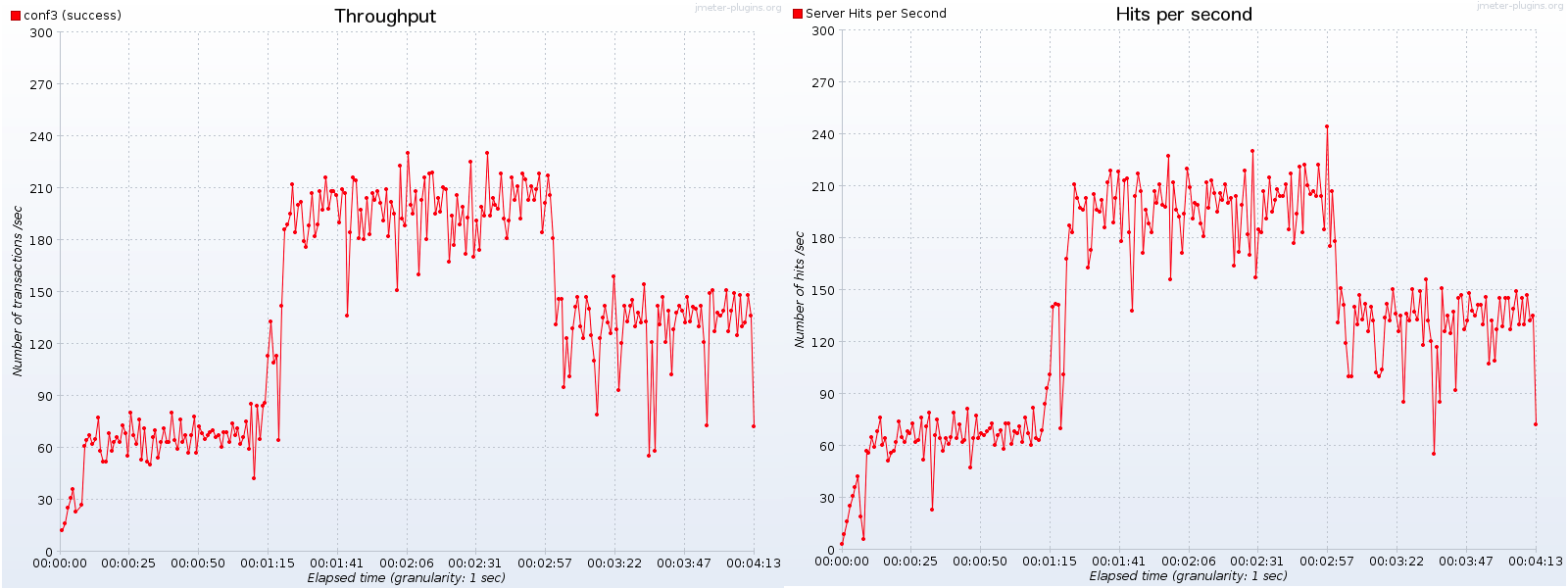


Figure 13: Operating system memory usage for Config 3 load testing with 50 concurrent users

Average throughput for load and peak simulations was 135 and 82 transactions per second respectively. Following graphs represents Throughput distribution over time showed along with hits per second graph during load simulation.

Figure 14: Throughput with hits per second visualization for Config 3 load testing with 50 concurrent users

Network traffic didn’t exceed 0.2 Mbit/s for inbound and 0.1 Mbit/s for outbound traffic.

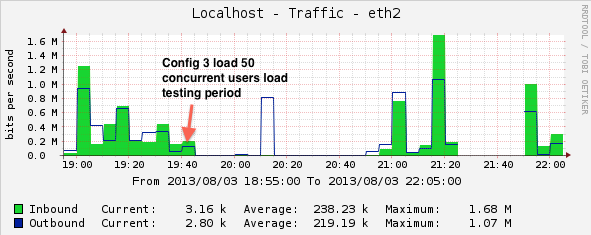


Figure 14: Network traffic for Config 3 load testing with 50 concurrent users

Detailed results: <http://bit.ly/1300Y42>

Raw data results: <http://bit.ly/17x6SFC>

### Config 5

### Config 6

### Config 8

### Config 10