LOAD & PERFORMANCE TESTING FOR J2EE

TESTING, MONITORING, ANALYSING AND REPORTING USING OPEN SOURCE TOOLS

Alexandru Ersenie
Senior Load & Performance Test Engineer
Edict eGaming GmbH, Hamburg
alexandru.ersenie@googlemail.com
http://www.alexandru-ersenie.com
AGENDA

PERFORMANCE BASICS
- Scope
- Metrics
- Factors on Performance
- Generating Load
- Performance Reports

MONITORING
- Monitoring types
- Reactive Monitoring/Reporting
- CPU Monitoring
- JVM GC and Heap Monitoring
- Other Performance Monitors

TOOLS
- Monitoring
- Reporting
- Analysis

3 TIER TUNING  BEST PRACTICES
SAMPLING/PROFILING  VIRTUALIZATION
LOGGING  TESTING IN THE CLOUD
SECURITY  NETWORKING

........
PERFORMANCE BASICS
PERFORMANCE

Short response time for a given piece of work

High throughput (rate of processing work)

Low utilization of computing resources

High availability of the computing system or application
1.1 REAL WORLD NUMBERS

**RESPONSE TIMES**

- 1 second of slower performance on pages could cost Amazon $1.6 billion in sales each year.
- 25% of users will leave a site if a page takes more than 4 seconds to load.

**THROUGHPUT**

- Facebook serves more than 2 million ‘Like’ buttons per second (June 2010).
- Facebook held a clear lead in total page views during March 2011, recording about 85 billion. This was more than three times as many as number two Google, which had about 25.6 billion.

**EFFICIENCY**

- Google uses a compressed, high performance, proprietary data storage file system, (...) designed to scale into the petabyte range (1000 terabytes).

---

http://performance-testing.org/performance-testing-statistics
http://en.wikipedia.org/wiki/BigTable
METRICS
### 1.2 METRICS

#### OPTIMISTIC

<table>
<thead>
<tr>
<th>SLA are <em>already</em> defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent users</td>
</tr>
<tr>
<td>10,000 logged in users</td>
</tr>
<tr>
<td>50,000 visitors</td>
</tr>
<tr>
<td>Transactions / Second</td>
</tr>
<tr>
<td>1,000 business transactions / second</td>
</tr>
<tr>
<td>4,000 web requests / second</td>
</tr>
<tr>
<td>Response Times</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Maximum 100 bytes per transaction</td>
</tr>
</tbody>
</table>

#### REAL WORLD PROJECTS

<table>
<thead>
<tr>
<th>SLA are <em>not</em> defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent users</td>
</tr>
<tr>
<td>Transactions / Second</td>
</tr>
<tr>
<td>Response Times</td>
</tr>
<tr>
<td>Storage</td>
</tr>
</tbody>
</table>

### Test to Achieve Defined Software Level Agreements

- How many transactions can the system handle?
- How many sessions can the system handle?
- What is the average response size?
- How much space will a transaction use in the database?
- What is the 90% value / business case?

### Test to Define Software Level Agreements

- Landing page total load time less than 8 seconds for 70% of the users.
- Transaction Response time less than 2 seconds for 90% of all transactions.
### RESPONSE TIMES

**What are the response times** that users are receiving when performing specific transactions in the system:

- Minimum / Maximum / Average response time
- 50 to 90 percent line

### THROUGHPUT

**How many users** can the system handle, and **how many transactions** can the system handle in a unit of time

- Transactions per second

As the number of users using the system increases, the throughput increases as well. The system is **busy satisfying user requests**

When the systems limit is reached, the throughput decreases, and waiting occurs, since the system is **busy managing itself** in order to satisfy all user requests.
FACTORS ON PERFORMANCE
1.3 FACTORS ON PERFORMANCE

SOFTWARE

Software design, architecture and configuration have great impact on performance:

PROGRAMMING

- Memory leaks
- Sync. errors
- Arithmetic errors
- Redundant operations
- Other

80% of all performance problems in JAVA

- Deadlocks
- Race conditions
- Buffer overflow
- Arithm. exceptions
- Dead code
- Redundant assign

CONFIGURATION

- JVM Config
  - -Xmx:2G
  - -XX:NewSize:1.8G
  - -XX:NewSize:1.8G
- ThreadPool
  - max-threadpoolsize="5"
- EJB container
  - Max-cachesize="512"; cache-timeout=3600
- JDBC
  - Max-connections="5"; jdbc-connection-timeout="5"
1.3 FACTORS ON PERFORMANCE

**HARDWARE**

**Single 3 Tier Server**

Shared resources:
- CPU
- Memory
- Network
- Disk

Virtualization

**OS / File System**

- Windows
  - 5000 cons
- Linux
  - 1024 Open files

**DB Server / Hardware**

- Oracle Standard
- Oracle Enterprise
- MSSql
- MySQL

- No partitioning
- Limited online operations
- Limited indexing
- Different locking implementation
- Different Index performance

- Default range for dynamic ports in Windows is 1024 to 5000
- Unix systems have a default maximum open files limit of 1024

**Shared resources:**
- CPU
- Memory
- Network
- Disk
GENERATING LOAD
1.4 GENERATING LOAD

WHAT DO WE NEED?

MONITORS

Web & Application Server

LOAD AGENT

LOAD AGENT

LOAD AGENT

LOAD AGENT

LOAD DISPATCHER

Test Results DB

Reporting Server

Reports
1.4 GENERATING LOAD

IMPLEMENTATION MODEL

- JMX; REST MONITORS
- Web & Application Server
- JMETER
- JMETER
- JMETER
- JMETER CONTROLLER
- MYSQL DB
- JASPER SERVER
- JASPER REPORTS

Alexandru Ersenie - Load & Performance Testing for J2EE
1.4 GENERATING LOAD

LOAD AGENTS
- Virtual Users
- Ramp up time
- Pause times
- Number of transactions
- Number of repetitions
- Increasing rate/repetition
- Server and Port

MONITORS
- Object usage
- EJB Resources
- JDBC Resources
- CPU Time
- HEAP Monitoring

REPORTING
- Process for DB Import
- Process for Maven
- Process for other formats

DISPATCHER
- Generate Load
- Collect Response Times
- Monitor Hardware Resources
- Monitor System Resources
- Process Results
- Import results into database
- Generate Reports

/start_test.sh

/users=200 ramptime=200 pausetime=2000 pausetimedev=500 transactions=100 repeats=1 loopsinrepeat=1 warmup=no' 'trace_objects=yes monitor_server=yes' 'generatereport=yes'
1.4 Generating Load

**functions**
- server_warmup
- start_test
- configure_server
- empty_logs
- monitor_resources
- generate_report

**script repo**
- HOMEPAGE
- LOGIN
- REGISTRATION
- PLACE ORDER
- EXPORT

**testplan repo**
- T_Homepage
- T_Login
- T_Registration
- T_Place_Order
- T_Export

**results**

**jmeter dispatcher folder**
- /includes
- /scripts
- /testplans
- /results

**mysql db**

**jasper reports**

Scripts call functions to control the test workflow.

Scripts start test plans by using functions.

Results are stored here.

Processed results are exported to a database.

Reports are generated from the DB.
PERFORMANCE REPORT
1.5 Performance Report

SUMMARY

METRICS / TRANSACTION

JVM CONFIGURATION

THROUGHPUT & RESPONSE TIMES

CPU USAGE

HEAP USAGE
MONITORING
2.1 Monitoring Types

**Active Monitoring**

- Allows real time monitoring of critical resources
- Allows real time interfering with the system:
  - Execute Garbage Collection
  - Generate Thread Dump
  - Analyze thread activity
  - Generate Heap Dump
- Enables monitoring the system's limits
- Allows a better understanding of how the system behaves depending on the load scenario, and the user behavior, thus enabling to determine what the system can perform
- Allows quickly modifying either the test configuration, or the software/hardware configuration, repeat the test and compare the results
- Allows parallel analysis of the system and the load scenario, thus enabling to identify root causes and behaviors quickly
2.1 Monitoring Types

Reactive Monitoring

- Allows reactive analyze and detailed interpretation of collected data
- Data collectors can easily be extended using a modular approach (function based):
  - Monitor JDBC; Monitor EJB; Monitor Thread Usage; Monitor Heap; etc

**ADVANTAGES**

1. Enables monitoring the system's behavior and resources over a prolonged period of time
2. Easily extendable by adding collectors
3. Storable information for future comparing
4. Tool independent
5. High definable granularity
6. Automated

**DISADVANTAGES**

1. High amount of data and complexity in analyzing the results
2. Requires high proficiency with tools, monitors, collectors for the purpose of extending the monitoring system
REACTIVE MONITORING
2.2 Reactive Monitoring

What are the response times for specific transactions:
- 50% 60% 70% 80% 90% line

What is the maximum throughput for specific transactions:
- No. of transactions
- Average Trans / Second
- Max Trans / Second

What is the resource usage for a load scenario:
- CPU
- HEAP
- JDBC
- EJB
- WEB

How efficient is the garbage collection for a load scenario:
- NO OF YOUNG GC
- NO OF OLD GC
- DURATION OF GC
- % IN GC
2.2 Reactive Monitoring

**RESPONSE TIMES**

<table>
<thead>
<tr>
<th>Request</th>
<th>Total Trans</th>
<th>Minimum</th>
<th>Average</th>
<th>Maximum</th>
<th>Std. Deviation</th>
<th>50 Percent</th>
<th>60 Percent</th>
<th>70 Percent</th>
<th>80 Percent</th>
<th>90 Percent</th>
<th>Average Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 - Heartbeat</td>
<td>510</td>
<td>2</td>
<td>10.7020</td>
<td>172</td>
<td>23.8698</td>
<td>4.0</td>
<td>4.0</td>
<td>5.0</td>
<td>7.0</td>
<td>17.0</td>
<td>2.999911767</td>
</tr>
<tr>
<td>1.2 - Login</td>
<td>510</td>
<td>15</td>
<td>69.6078</td>
<td>501</td>
<td>67.3849</td>
<td>51.0</td>
<td>59.0</td>
<td>69.0</td>
<td>85.0</td>
<td>138.0</td>
<td>2.992624061</td>
</tr>
<tr>
<td>2.1 - Join Session</td>
<td>510</td>
<td>14</td>
<td>31.7176</td>
<td>511</td>
<td>43.3257</td>
<td>20.0</td>
<td>23.0</td>
<td>26.0</td>
<td>32.0</td>
<td>47.0</td>
<td>2.991763850</td>
</tr>
<tr>
<td>2.2 - Start Game 1 Hands</td>
<td>51000</td>
<td>24</td>
<td>2400.27</td>
<td>52277</td>
<td>4712.59</td>
<td>893.0</td>
<td>1314.0</td>
<td>1867.0</td>
<td>2609.0</td>
<td>5319.0</td>
<td>32.42051570</td>
</tr>
</tbody>
</table>

**THROUGHPUT**

- Number of transactions per second
- Average transactions per second
- Response time increasing; throughput decreasing as users enter the system
- Average Response Time
- Maximum Response Time
## 2.2 Reactive Monitoring

### RESPONSE TIMES DRILL

<table>
<thead>
<tr>
<th>Transaction Report</th>
<th>Sorted by request name - alphabetically</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Total Trans</td>
</tr>
<tr>
<td>BC1 - 1.1 - /home</td>
<td>1500</td>
</tr>
</tbody>
</table>

### THROUGHPUT DRILL

[Graph showing throughput drill data]
2.2 Reactive Monitoring

RESOURCES

CPU

HEAP

JDBC

CONNECTIONS
2.2 Reactive Monitoring

GARBAGE COLLECTION

Heap Usage After GC

Clock Time (seconds)

Heap Usage After GC (MB/kev)
2.2 Reactive Monitoring

GARBAGE COLLECTION STATISTICS

[Image: Garbage collection statistics chart showing heap usage after GC, duration, cumulative allocation, creation rate, and user-defined categories.]

[Image: Additional metrics such as Eden, Survivor, Old, and Perm space usage, and GC activity summary with various collection times and counts.]
2.2 Reactive Monitoring

**OBJECT USAGE**

- See the behavior of objects in time
- Drill down on the object to see detailed information like **Instances** and **Bytes occupied**
2.2 Reactive Monitoring

**OBJECT USAGE - PRIMITIVES**

- See the behavior of objects in time
- Drill down on the object to see detailed information like **Instances** and **Bytes occupied**
Java Management Extensions (JMX)

Supplies tools for managing/monitoring:
- Applications
- System objects
- Devices
- Service oriented networks.

Resources are represented by objects called MBeans.

REST MONITORING

Monitoring and management data exposed by the application server (example: Glassfish).

LOAD GENERATOR

- Transactions
- Response Times
- Response Codes
- Response Sizes

SYSTEM

- Top
- Vmstat
- Iostat
- Jmap

COLLECTORS

MY SQL DB

SERVER LOGS

Garbage Collection Logs
ACTIVE MONITORING

CPU
INSERT MOVIE WITH RUNNING TEST AND MONITORING
2.3 Monitoring Types – CPU Monitoring

- Allows **real time monitoring** of CPU usage
- Allows identifying **hotspots** (spikes) and relate them in time using graphical timestamp representation
- Identify **Garbage Collections** and their effects on CPU Usage
- Allows creating “**Snapshots**” and future reactive analysis based on CPU usage patterns
- Allows identifying **outside contributors** (Database, JMS, I/O)
OPTIMAL BEHAVIOR

- Server resources properly configured – EJB Cache, EJB Pool, Thread Pool, JDBC Connection Pool
- Load configuration adapted for optimum usage of resources: Requests are being Processed, JDBC connections are used optimally
Depending on the configured Garbage Collection Strategy, on the Heap usage of your application, and the properly configuration of the Heap according to business scenarios and memory footprints, Garbage Collection can have a huge impact on throughput.

- Parallel Collections with only two CPU
- Stop the world strategy instead of CMS
- Poor Heap Configuration (too small, too big, wrong ratios of young to old)
- Manual Garbage Collections
2.3 Monitoring Types – CPU Monitoring

**DATABASE WAITS**

- **Long running queries** – Occupying the JDBC resource for too long. Requests are being queued, translating in decreased throughput and increased response times
- **Oracle Redo Log** failed checkpoints – Database stops servicing for satisfying internal needs
Number one cause for thread contention: wait times induced by *synchronized methods* (threads have to wait in a queue for acquiring the lock)
ACTIVE MONITORING
HEAP
2.4 Monitoring Types – HEAP Monitoring

**HEAP MONITORING**

- Allows **real time monitoring of HEAP usage**
- Allows **identifying memory leak trends**
- **Correlate Garbage Collection with CPU Usage:**
  - CPU usage
  - Time in GC
  - Overall system behavior on GC
- Determine **Garbage Collection rates and durations**
2.4 Monitoring Types – HEAP Monitoring

JVM HEAP STRUCTURE

PERMANENT GENERATION

- Internal representations of the JAVA classes
- Objects describing classes and methods
- Information used for optimization by the JIT compilers

YOUNG GENERATION

- Initially objects are allocated in the Young Generation
- When collected, objects are moved between survivors (default 32 collections)

OLD GENERATION

- Objects that have survived the maximum allowed number of collections
- Dead objects waiting to be collected

S1

S2

<table>
<thead>
<tr>
<th>1</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>32</td>
</tr>
</tbody>
</table>
2.4 Monitoring Types – HEAP Monitoring

JVM HEAP STRUCTURE

Visual GC

Refresh rate: Auto msec.

Spaces

Perm Old Eden

Graphs

- Compile Time: 5233 compiles - 1m 57.709s
- Class Loader Time: 30985 loaded, 432 unloaded - 37.664s
- GC Time: 406 collections, 5m 57.263s Last Cause: System.gc()
- Eden Space (900.000M, 900.000M): 170.513M, 294 collections, 11.263s
- Survivor 0 (300.000M, 300.000M): 0
- Survivor 1 (300.000M, 300.000M): 0
- Old Gen (2.002G, 2.002G): 1.009G, 112 collections, 6m 46.000s
- Perm Gen (512.000M, 332.547M): 199.513M

Histogram

Parameters

Tenuring Threshold: 4 Max Tenuring Threshold: 4 Desired Survivor Size: 283115520 Current Survivor Size: 314572800

Histogram

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
HEAP MONITORING
EXAMPLES
MEMORY LEAK?

**Used Memory** increases as the number of live objects surviving collections increases (particularly under load)

An **increasing trend** is not necessarily a memory leak. Objects can survive several collections (max threshold), and can be already dead, waiting for the old collector

Wait for a **full garbage collection** to see if the memory decreases

Test systems can be provided with the option of **manually triggering a garbage collection** to see if objects are being released after the test is over

In test systems **wait for several full garbage collections** and build a trend line in order to see if a memory leak is showing

---

2.4 Monitoring Types – HEAP Monitoring

- **Used Memory** increases as the number of live objects surviving collections increases (particularly under load)
- An **increasing trend** is not necessarily a memory leak. Objects can survive several collections (max threshold), and can be already dead, waiting for the old collector
- Wait for a **full garbage collection** to see if the memory decreases
- Test systems can be provided with the option of **manually triggering a garbage collection** to see if objects are being released after the test is over
- In test systems **wait for several full garbage collections** and build a trend line in order to see if a memory leak is showing
A full Garbage Collection goes over all generations, parsing the entire object structure to identify removable objects.

- Dead or marked for GC objects are removed.
- Space is reclaimed.
- Usually the old generation is at least twice the size of the young generation.
- Collection time for the old generation significantly increased (object tree parsed).
- Garbage Collection strategy of critical importance: Parallel or Concurrent GC.
2.4 Monitoring Types – HEAP Monitoring

**MEMORY LEAK**

- **Used Memory** increases as the number of live objects surviving collections increases (particularly under load)
- Objects that cannot be collected are moved to the old generation
- Old Garbage Collections cannot remove the objects; space cannot be reclaimed
- Total Heap used increases constantly
- Eventually the system performs only Garbage Collection
- The system is busy managing itself, instead of running your application
- Both throughput and response times affected
- Server restart is required
- The right time to trigger a HEAP DUMP and see what objects are leaking
ACTIVE MONITORING
OTHER MONITORS
2.5 Monitoring Types – Other Monitors

- Threads can be monitored, and blocking or long running threads can be identified

Caching shared data in a Hash Map

- User Profile
- Session Information
- File contents

Every request will want to acquire and hold the lock on the hash map, becoming a bottleneck

Brian Goetz: Threading lightly : Reducing Contention

Synchronized Methods

One thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object.

Java Tutorials - Concurrency
Most often contention problems appear only under load, due to concurrency effects (more requests on the same object/method). The right time to trigger a thread dump and see the root cause of contention.

**BEHAVIOR**

- Several threads in “Monitor” state, at the same time, indicate thread contention.
- CPU usage drops dramatically, for it is busy with solving the locks instead of satisfying business requests.
- Most often contention problems appear only under load, due to concurrency effects (more requests on the same object/method).

**ACTION**

- The right time to trigger a thread dump and see the root cause of contention.
2.5 Monitoring Types – Other Monitors

**THREAD DUMPS**

- Lists all existing threads and their state when the thread dump was triggered
- Allows quickly identifying blocked/blocking threads
- Threads are listed together with their stack traces, allowing identifying of:
  - What resource is being locked
  - Who locks the resource?
  - What is the locking thread doing?
2.5 Monitoring Types – Other Monitors

THREAD DUMPS

- Kill -3 PID: writes stack trace to jvm.log
- jvm.log can be imported into IBM Thread Analyzer
2.5 Monitoring Types – Other Monitors

EJB MONITORING

Actively monitor the number of EJB Beans in cache and other statistics like:

- Cache hits
- Passivation statistics
- Number of beans created
- Number of beans removed

JDBC MONITORING

Actively monitor the number of JDBC resources in use, and other statistics like:

- Connections created
- Connections removed
- Connections in pool

WEB MONITORING

Actively monitor the number of requests, and other statistics like:

- Error count
- Max request time
Database Call on each Transaction – In this case each transaction checks if the user is valid, by creating and using a new JDBC Connection / check;
TOOLS
3. Tools

LOAD
- JMeter
- Shell Scripts

MONITORING
- VISUAL VM
- CheckJMX
- REST Interface
- System Interfaces

PROCESSING
- PENTAHO
- Shell Scripts

REPORT DESIGN
- IReport

REPORTS
- Jasper Reports

HEAP ANALYSER
- MAT
- YourKit

THREAD ANALYSER
- VisualVM
- IBM Thread Analyzer

GARBAGE COLLECTION
- HP Jmeter

PROFILING
- YourKit
THANK YOU FOR LISTENING