Die Last und Lust mit Performancetests
Apache JMeter und Gatling im Vergleich

Frank Pientka @fpientka
Agenda

- Why performance testing?
- Java performance testing tools (selection)
  - Apache JMeter
    - JMeter main elements?
    - Have a (test-)plan!
    - Further steps, extensions, challenges
  - Gatling
    - Main elements
    - Test simulations
- Taurus – best of both worlds
- Comparison
- Testing right! - Resume
Why performance tests?

Bad, bad panda!
Come on. We want photos.

We’re aware of the problem and are fixing it.
Thanks for your patience.

Who?
When?
How often?
What went wrong while testing?

How secure & safe?

The Heisenbug!

Volkswagen St. (VW) share
WHEN WAS THE LAST TIME YOU HAD TO SOLVE A PERFORMANCE PROBLEM IN YOUR SOFTWARE?

DZONE’S GUIDE TO PERFORMANCE: OPTIMIZATION AND MONITORING, VOLUME III 2017
A typical web application

Diagram:
- Webserver
- Database
- LDAP server
- IDM

Connections:
- App
- User Interface
- IDM
- LDAP server
- Database

鳃
OpenSource Java performance testing tools (selection)

- JUnitPerf
- ab
- Gattling
- Taurus
- The Grinder
May I introduce Apache JMeter? 18 years of experience

- Since 1998 for **Apache Tomcat** loadtest (Stefano Mazzocchi)
- Recent version **3.2** (2017-04)
- Supported protocols
  - Web - **HTTP, HTTPS**, FTP
  - SOAP/XML-RPC
  - Database via **JDBC**, MongoDB, HADOOP
  - LDAP
  - JMS
  - Mail - SMTP(S), POP3(S) and IMAP(S)
  - Native commands or shell scripts

+ Highly extensible and configurable, tutorial, example templates, demos
- Limited reporting capacities, short outdated documentation, i18n
Versions distribution https://jmeter-plugins.org/stats/

2017: 80% = 3.1+3.2
JMeter testplan elements

Start

- Workbench
  - Test Recorder

- Listener
  - SummaryReport
  - Results Tree
  - Results Table

- ConfigTestElement
  - Request Defaults
  - Authorisation Manager
  - Cookie Manager

- Assertion

- Sampler
  - HTTP Request

- Timer
A JMeter web test plan example http://www.heise.de/
A JMeter JDBC test plan
A JMeter LDAP test plan
JMeter test plan recorder (HTTP Proxy Server) elements

Alternative: BlazeMeter’s Google Chrome extension, .har file to a JMeter test plan (.jmx file) converter
Typical steps

- **Create** Testplan
- **Test** Testplan
- **Run** Testplan and evaluate results
- **Document** Testplan and results
  - Extras/schematic.cmd Testplan.jmx → Testplan.html
  - TestPlanCheck --jmx <filename> [ --stats --tree-dump ]
  - jmeter -n -t <test JMX file> -l <test log file> -e -o <Path to output folder>
Five Ways To Launch a JMeter Test without JMeter GUI

1. Command line: `jmeter -n -t your_script.jmx`

2. JMeter Ant Task `ant extras/build.xml`

3. JMeter Maven Plugin `mvn verify jmeter:jmeter`

4. from Java code

5. Distributed Testplan as master and slaves
Run JMeter from a Java Class

```java
import org.apache.jmeter.engine.StandardJMeterEngine;
import org.apache.jmeter.save.SaveService;
import org.apache.jmeter.util.JMeterUtils;
import org.apache.jorphan.collections.HashTree;
import java.io.FileInputStream;

public class JMeterFromExistingJMX {
    public static void main(String[] argv) throws Exception {
        // JMeter Engine
        StandardJMeterEngine jmeter = new StandardJMeterEngine();

        // Initialize Properties, logging, locale, etc.
        JMeterUtils.loadJMeterProperties("/path/to/your/jmeter/bin/jmeter.properties");
        JMeterUtils.setJMeterHome("/path/to/your/jmeter");
        JMeterUtils.initLogging(); // comment this line out to see extra log messages i.e. DEBUG level
        JMeterUtils.initLocale();

        // Initialize JMeter SaveService
        SaveService.loadProperties();

        // Load existing .jmx Test Plan
        FileInputStream in = new FileInputStream("/path/to/your/jmeter/extras/Test.jmx");
        HashTree testPlanTree = SaveService.loadTree(in);
        in.close();

        // Run JMeter Test
        jmeter.configure(testPlanTree);
        jmeter.run();
    }
}
```
Continuous integration...more often

Jenkins

Performance Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin
JMeter distributed testing with Master-Slave

```
In bin/jmeter.properties "remote_hosts=localhost:1099,localhost:2010" add IP addresses of Slave-Clients, server.rmi.port=1099, server.rmi.localport=1099, client.rmi.localport=2010
On Master: jmeter -n -t script.jmx -R slave1,slave2,slave3.
'Remote Start All' distribute and execute test plan and receive results, SET SERVER_PORT=1099, set REMOTE_HOSTS=master:1099
'Remote Stop', control jmeter-server.log, and have check and assert elements, to handle errors
```
How to document test plans in HTML?

```
cd apache-jmeter-3.0\extras
schematic.cmd test-plan.jmx
ant -Dtest=test-plan run report
TestPlanCheck.bat --jmx testplan.jmx --stats --tree-dump
```

Cold/warm start!
Think Time!

© Materna GmbH 2017
www.materna.com
Apache JMeter Dashboard 3.x

http://jmeter.apache.org/usermanual/generating-dashboard.html
https://sense.blazemeter.com
Useful extensions http://jmeter-plugins.org

Custom Plugins for Apache JMeter™

This project is an independent set of plugins for Apache JMeter, the popular Open-Source load and performance testing tool.

Latest Update

Version 1.7.1 was released on the 12th of October, 2015. It is mostly a maintenance release that fixes issue with Ultimate Thread Group. See ChangeLog page for the list of all
The prerequisite is to run JMeter 2.12 or above with a Java 7. Follow installation procedure after download.

Choose on Your Taste

<table>
<thead>
<tr>
<th><strong>Standard Set</strong></th>
<th><strong>Extras Set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic plugins for everyday needs. Does not require additional libs to run.</td>
<td>Additional plugins for extended and complex testing. Does not require additional libs to run.</td>
</tr>
<tr>
<td>Download</td>
<td>Installation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Extras with Libs Set</strong></th>
<th><strong>WebDriver Set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional plugins that do require additional libs to run.</td>
<td>Selenium/WebDriver testing ability.</td>
</tr>
<tr>
<td>Download</td>
<td>Installation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Hadoop Set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop/HBase testing plugins.</td>
</tr>
</tbody>
</table>
Resume JMeter

- Reliable and extensible tool
- How to integrate JMeter in the CI-Build?
- How to document test plans and results?
- How to use JMeter with Selenium/JS/mobile Apps/Cloud?
- Takeover from BlazeMeter by CA Technologies
Gatling is

an open-source load testing framework based on Scala, Akka and Netty
High performance
Ready-to-present HTML reports
Scenario recorder and developer-friendly DSL
"Arm yourself for performance"
Gatling.io

- Since 2011 ASF 2.0 license
- Scala-Akka-Netty based HTTP Load Test Tool
- Interactive HTML reports
- Scenario recorder and developer-friendly DSL
- Recent version 2.2.5
- http://gatling.io/docs/2.2.3/cheat-sheet.html
Gatling

700,000 downloads

2012 2013 2014 2015 2016 2017

40,000 downloads/month in 2017
Gatling typical steps

Record browser activities
Edit recorded scenarios

Launch (Shell, Build, Jenkins)

Analyze
Gatling Testrecorder

`recorder.[bat|sh]`

Set Browser-Proxy
First Testsimulation with Scala

class MySimulation extends Simulation {
  val conf = http.baseUrl("http://localhost")
  val scn = scenario("Gatling")
    .exec(http("index").get("/"))
    .during(10 minutes) {
      exec(
        http("json").get("/json")
          .check(jsonPath("$.id")
              .saveAs("id"))
      )
    }
  setUp(scn.inject(atOnceUsers(5)))
    .protocols(conf)
}
class MySimulation extends Simulation {
    val httpProtocol = http
        .baseUrl("https://www.heise.de/")
        .inferHtmlResources()
    val scn = scenario("Gatling")
        .exec(http("request_0")
            .get("/")
            .check(status.is(200)))
    setUp(scn.inject(atOnceUsers(1))).protocols(httpProtocol)
}
Gatling's DSL Cheat sheet main elements

- Scenario definition Describe your users behavior
- Simulation configuration Tune your simulation (time, throttling)
- Injection profile Control how users are injected in your scenario
- Assertions conditions
- Assertions Check that your results match your expectations
- Feeder definition Inject data in your scenario (types, strategy)
- HTTP Action Define the HTTP requests sent in your scenario
- HTTP Checks Verifying server responses
- HTTP Protocol Mutualize your scenario's code and tune HTTP client
- JMS Checks Verifying server responses
- JMS Configuration Configure how Gatling connects to your JMS queues
- JMS Define the JMS requests sent in your scenario
- SSE (Server Sent Events) Define the SSE requests sent in your scenario
- WebSockets/SSE Checks Verifying server responses
- WebSockets Define the WebSocket requests sent in your scenario
- WebSockets Configuration Mutualize your code and tune WebSocket client
Run: `gatling.bat -s computerdatabase.BasicSimulation`

Simulation computerdatabase.BasicSimulation completed in 18 seconds

Parsing log files...
Parsing log files; done
Generating reports...

--- Global Information ---
- request count 3 (OK=5  KO=- )
- min response time 57 (OK=57  KO=- )
- max response time 416 (OK=416  KO=- )
- mean response time 165 (OK=165  KO=- )
- std deviation 141 (OK=141  KO=- )
- response time 50th percentile 34 (OK=34  KO=- )
- response time 75th percentile 238 (OK=238  KO=- )
- response time 95th percentile 394 (OK=394  KO=- )
- response time 99th percentile 410 (OK=410  KO=- )
- mean requests/sec 9.429 (OK=9.429  KO=- )

--- Response Time Distribution ---
- t < 500 ms 5 (100%)
- 500 ms < t < 1200 ms 0 (0%)
- t > 1200 ms 0 (0%)
- Failed 0 (0%)

Reports generated in src:
Please open the following files: `G:\gatling-charts-highcharts-bundle-2.2\results\BasicSimulation-1465177066553\index.html`
View /results/basicsimulation-1485177666450/index.html
Comprehensive web performance testing tool

Need some programming knowledge

Not so many experience and extensions
http://gettaurus.org/ getting started

- Install pip python get-pip.py
- pip install bzt
- **Create a Config**
  - quick_test.yml
  - existing_jmeter_script.yml
  - complex_yaml_script.yml
- Run The Test bzt quick_test.yml
Taurus Executor Types

- **JMeter**, executor type jmeter
- **Selenium**, executor type selenium
- **Gatling**, executor type gatling
- **Grinder**, executor type grinder
- **Locust**, executor type locust
- **PBench**, executor type pbench
- **Siege**, executor type siege
- **ApacheBenchmark**, executor type ab
- **Tsung**, executor type tsung
JMeter Testplan and Gatling Scenario execution: `bzt example.yml`

```yaml
---
scenarios:
  get-requests:
    requests:
      - http://localhost/1
      - http://localhost/2

execution:
  - concurrency: 10
    hold-for: 1m
    scenario: get-requests
  - executor: gatling
    concurrency: 5
    iterations: 10
    scenario: only-script
  - hold-for: 20s
    scenario: my_jmx_file.jmx # or `bzt my_jmx_file.jmx`

reporting:
  - final_stats
  - console
```
Who is the winner? JMeter or Gatling
https://www.openhub.net/p/jmeter
https://www.openhub.net/p/gatling
https://www.openhub.net/p/_compare?project_0=Apache+JMeter&project_1=Gatling

May 29, 2011 – Jan 25, 2017
Contributions to master, excluding merge commits
<table>
<thead>
<tr>
<th>Feature</th>
<th>JMeter</th>
<th>Gatling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maturity</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Protocols</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Performance</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>Extensions</td>
<td>+++</td>
<td>0</td>
</tr>
<tr>
<td>Changability</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Usability</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>
- Wide adoption (since 1998)
- big community
- many protocols (Subsystems)
- many extensions, report types
- Lots of documentation
- Active community (3.2)
  - XML machine readable, more than for human
  - Restricted scripting
  - low integration

- For Scala-Programmer (DSL)
- Modern, interactive HTML-Reports
- Good integration
  - Small community
  - Small number of extensions
  - Decreasing activity (3.0 2017?)
Resume

- Lack of HTTP/2 support will come 😊
- Performance tests are more important than ever
- For SPA/JavaScript-Apps combine with Selenium tests
- Choose your test tool wisely and know it well
- Taurus to get the best of both worlds!
- Don’t be fooled by your tools!
Wrap-up

- Performance-test are still important
- Test often test wisely
- Integrate in your release process
- Adopt Cloud testing
- Monitor your Performance test environment
- Don’t guess measure!
- Know your tools, but don’t fool yourself
Further info

- Gatling [http://gatling.io](http://gatling.io)
- JMeter: [http://jmeter.apache.org](http://jmeter.apache.org)
- JMeter extensions: [http://jmeter-plugins.org](http://jmeter-plugins.org)
- JMeter-Online-Reports: [https://sense.blazemeter.com/](https://sense.blazemeter.com/)
- JMeter-ec2 script: [https://github.com/oliverlloyd/jmeter-ec2](https://github.com/oliverlloyd/jmeter-ec2)
- Last- und Performance-Tests mit JMeter oder Gatling, Pientka [https://heise.de/-3648505](https://heise.de/-3648505)