

IMPORTANCE OF LOAD TESTS BEFORE APPLICATION RELEASE

By **Massimiliano Cavicchioli**



ZEND/PHP CONFERENCE & EXPO 8-11 OCTOBER 2007

DEFINITIONS & THEORY

Some definitions and some theoretical bases related to performance testing.



PERFORMANCE TESTING

In software engineering, **performance testing** is testing that is performed, from one perspective, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability and reliability.

Performance testing is a subset of Performance engineering, an emerging computer science practice which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort.



PURPOSES OF PERFORMANCE TESTING

Performance testing can serve different purposes:

- It can demonstrate that the system meets performance criteria.
- It can compare two systems to find which performs better.
- It can measure what parts of the system or workload cause the system to perform badly.



PERFORMANCE TESTING IS CRITICAL

It is critical to the cost performance of a new application, that performance test efforts begin at the inception of the development project and extend through to deployment.

The later a performance defect is detected, the higher the cost of remediation.

This is true in the case of functional testing, but even more so with performance testing, due to the end-to-end nature of its scope.



SOME MORE DEFINITIONS - BENCHMARK

In computing, a **benchmark** is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it.

Benchmarking is helpful in understanding how an application responds under varying conditions. You can create scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics and even the effect on the application of using a new release of the product.



SOME MORE DEFINITIONS – STRESS TESTING

Stress testing is a form of testing that is used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results.

Stress testing often refers to tests that put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered correct behavior under normal circumstances. In particular, the goals of such tests may be to ensure the software doesn't crash in conditions of insufficient computational resources (such as memory or disk space), unusually high concurrency, or denial of service attacks.



PERFORMANCE TESTING BASICS

Things to bear in mind when involved in performance testing practices.



PERFORMANCE SPECIFICATION

It is fundamental to detail performance specifications (requirements) and document them in any performance test plan.

Ideally, this is done during the requirements development phase of any system development project, prior to any design effort.

Performance specifications should ask the following questions, at a minimum:

- In detail, what is the performance test scope? What subsystems, interfaces, components, etc are in and out of scope for this test?
- For the user interfaces (UI's) involved, how many concurrent users are expected for each (specify peak vs. nominal)?
- What does the target system (hardware) look like (specify all server and network appliance configurations)?
- What is the Application Workload Mix of each application component? (for example: 20% login, 40% search, 30% item select, 10% checkout).
- What is the System Workload Mix? [Multiple workloads may be simulated in a single performance test] (for example: 30% Workload A, 20% Workload B, 50% Workload C)
- What are the time requirements for any/all backend batch processes (specify peak vs. nominal)?



MIMICKING PRODUCTION CONDITIONS

In performance testing, it is crucial (but often difficult to arrange) for the test conditions to be similar to the expected actual use.

This is, however, not entirely possible in actual practice. The reason is that production systems have a random nature of the workload and while the test workloads do their best to mimic what may happen in the production environment, it is impossible to exactly replicate this workload variability.



PRODUCTION CONDITIONS COMPLEXITIES

Loosely-coupled architectural implementations (e.g.: SOA) have created additional complexities with performance testing.

Enterprise services or assets (that share common infrastructure or platform) require coordinated performance testing to truly replicate production-like states.

Due to the complexity and financial and time requirements around this activity, some organizations now employ tools that can monitor and create production-like conditions (also referred as "noise") in their performance testing environments (PTE) to understand capacity and resource requirements and verify / validate quality attributes.



TASKS TO UNDERTAKE

Tasks to perform such a test would include:

- Decide whether to use internal or external resources to perform the tests, depending on inhouse expertise (or lack thereof)
- Gather or elicit performance requirements (specifications) from users and/or business analysts
- Develop a high-level plan, including requirements, resources, timelines and milestones
- Develop a detailed performance test plan (including detailed scenarios and test cases, workloads, environment info, etc)
- Choose test tool/s
- Specify test data needed and charter effort (often overlooked, but often the death of a valid performance test)
- Develop proof-of-concept scripts for each application/component under test, using chosen test tools and strategies
- Develop detailed performance test project plan, including all dependencies and associated timelines
- Install and configure injectors/controller
- Configure the test environment, router configuration, quiet network (we don't want results upset by other users), deployment of server instrumentation, database test sets developed, etc.
- Execute tests – probably repeatedly (iteratively) in order to see whether any unaccounted for factor might affect the results
- Analyze the results - either pass/fail, or investigation of critical path and recommendation of corrective action



PERFORMANCE TESTING CONCEPTS APPLIED TO LAMP

Applying valuable performance testing practice
on LAMP systems.



BASICS ON LAMP PERFORMANCE TESTING

As we all know LAMP is the acronym of Linux, Apache, MySQL and PHP.

Each one of these entities has its own performance limitations and we must always be careful, while benchmarking, to have clearly in mind the thin demarcation line between the layers.



LINUX PERFORMANCE TESTING

Linux is, most probably, the best (and most used) OS for the purposes of a small to medium Web application.

Performance tests, related to Web applications, that can be done in the sphere of a Linux OS should focus on the following areas:

- Memory Management
- I/O Subsystem
- Process Scheduler
- File System
- Network



APACHE PERFORMANCE TESTING

Apache is, most probably, the best (and most used) Web Server for the purposes of a Web application of any size.

Performance tests, in the sphere of the Apache Web Server, should focus on the following areas:

- Memory Consumption
- Serving of Static Pages
- Serving of Dynamic Pages
- MPM Mode (prefork, worker)



MySQL (DB) PERFORMANCE TESTING

Being the databases a fundamental part in the logic of any Web application, DBs performance testing is a must.

DB performance tests should focus on the following areas:

- Sustainable Data Amount
- Connections Management
- Caching Policies
- Concurrent User Load
- Scalability



PHP PERFORMANCE TESTING

PHP is, most probably, the best (and most used) Web Server for the purposes of a Web application of any size.

PHP performance tests should focus on the following areas:

- Major Flows Memory Usage
- DB Transactions Management
- Bandwidth Management
- Generated Sys Calls
- Scalability



LAMP PERFORMANCE TIPS

To optimize the work within the LAMP stack, many simple enhancements can be applied.

Each LAMP entity can be tuned, mostly through configuration improvements, to better serve the performance needs of a Web application.

Always remember though, that any change made to default configurations in the LAMP stack has (most probably) repercussions on all the stack operations.



PHP PERFORMANCE IMPROVEMENTS

Some tips to improve PHP performances on a production server:

- PHP compilation with flags best suiting the system hardware and with no debug data.
- Opcode Caching
 - Each PHP script is compiled only once for each revision.
 - Reduced File IO, opcodes are being read from memory instead of being parsed from disk.
- Engine Configuration Improvements
 - register_globals = Off (Also security related)
 - magic_quotes_gpc = Off
 - expose_php = Off (Also security related)
 - register_argc_argv = Off
 - always_populate_raw_post_data = Off
 - session.use_trans_sid = Off
 - session.auto_start = Off
 - session.gc_divisor = 1000 or 10000



CODE ACCELERATION/OPCODE CACHING A TEST

- From a test on Wordpress
 - Without acceleration
 - Requests per second: 2.28
 - Time per request: 437.766 ms
 - With acceleration
 - Requests per second: 8.11
 - Time per request: 123.377 ms
 - 4-fold increase in performance by flipping a switch



APACHE PERFORMANCE IMPROVEMENTS

Some tips to improve Apache performances on a production server:

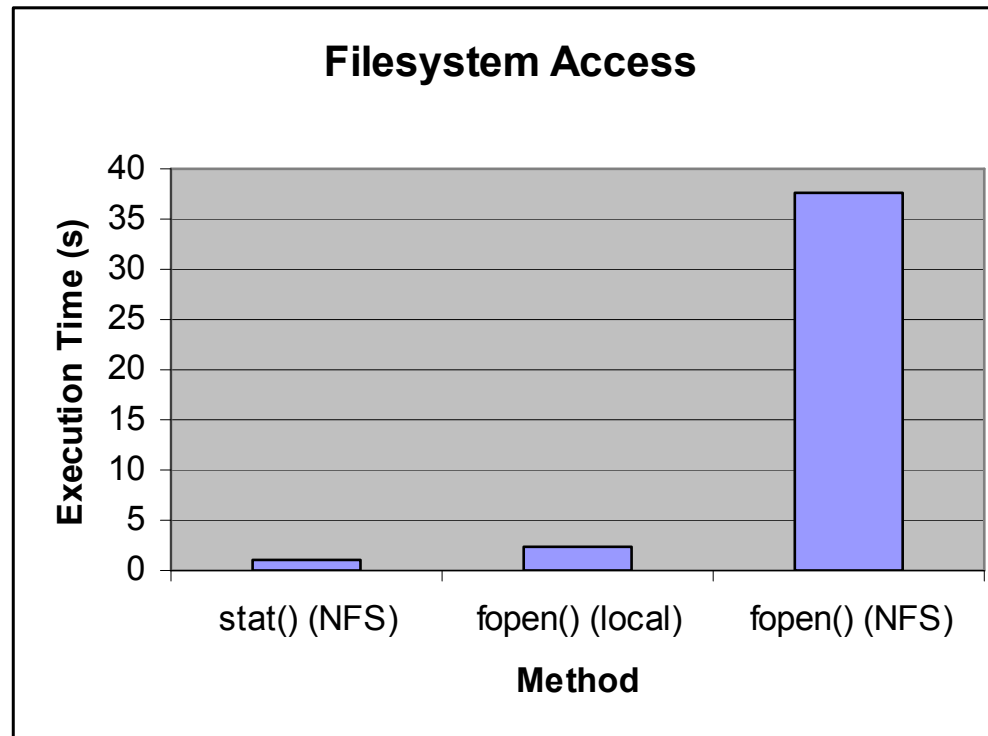
- Apache compilation with flags best suiting the system hardware and with no debug data.
- Static vs Dynamic Extensions
 - Using a static configuration of Apache, choose the modules you wish to incorporate with care — the more modules, the more memory you use for each process serving a request.
 - Using Dynamically loaded modules within Apache is a convenience, but using them can result in a performance hit, as the code is loaded when the module is required.
- Engine Configuration Improvements
 - Whenever possible disable **.htaccess** via **AllowOverride none**.
 - If **logs** are not strictly necessary disable them otherwise log everything to 1 file and break it up during the analysis stage.
 - Do not enable **ExtendedStatus**.
 - Do not enable **HostnameLookups**.
 - Keep **ServerSignature** off.
 - If the server is only serving dynamic requests, disable **KeepAlive**.



ARCHITECTURAL TIP

DO NOT SERVE PHP CONTENT FROM NFS

- Single point of failure
- In a test of 100,000 iterations...



PERFORMANCE TESTING TOOLS

Tools to perform the performance testing tasks



OPEN SOURCE – MAJOR TOOLS

On the regard of performance tests or stressing tests, there are three major players in the open source space:

- WebLOAD Open Source
 - <http://www.webload.org>
- OpenSTA
 - <http://opensta.org>
- JMeter
 - <http://jakarta.apache.org/jmeter>



MAJOR TOOLS COMPARISON

A thorough comparison of these major tools has been carried out by opensource-testing.org on the base of following criteria:

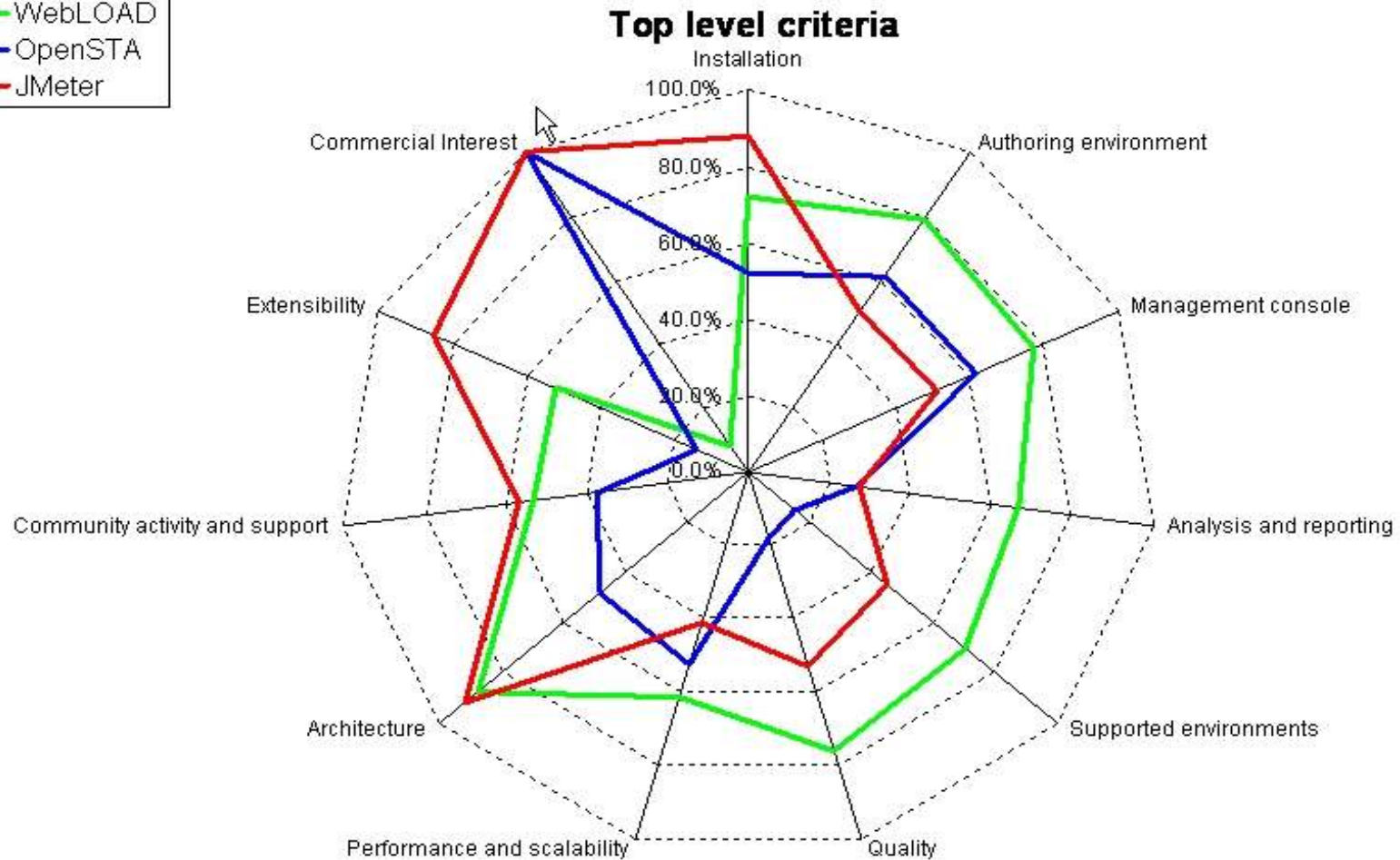
- Tool capabilities and design
- Community characteristics and maturity
- Business Models

The methodologies used to evaluate the above areas are the following:

- 11 high level criteria (listed in next slide).
- Detailed low-level criteria for each.
- Load Runner capabilities considered as 10 (maximum score).
- Reporting capabilities.
- Spider charts for comparison.
- Overall tool score.



11 CRITERIA COMPARISON



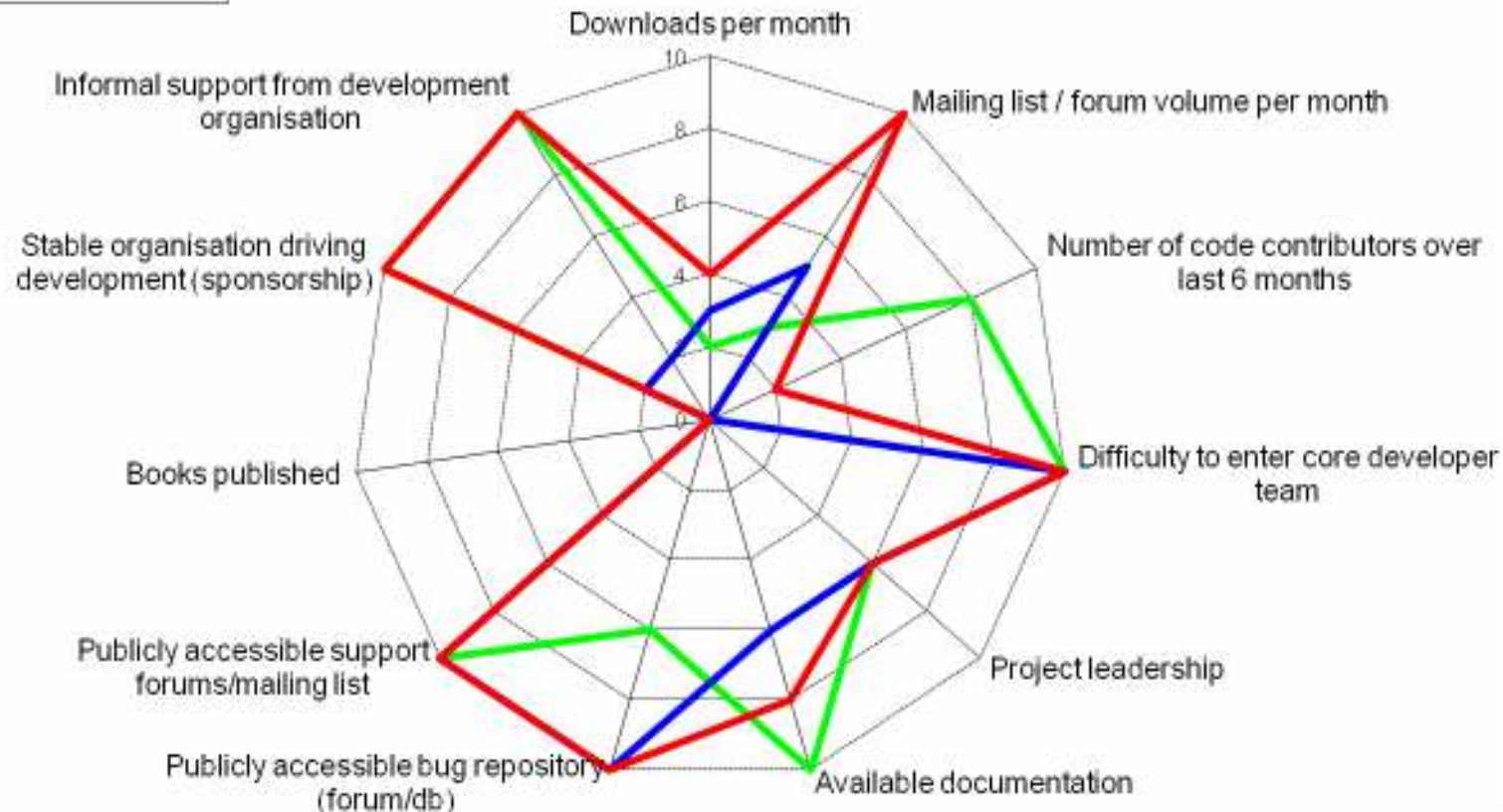
COMMUNITY ACTIVITY COMPARISON

WebLOAD

OpenSTA

JMeter

Community activity and support



TOOLS PRO AND AGAINST - WebLOAD

WebLoad strengths:

- Richness of authoring, controller and reporting features, particularly:
 - Scenario editing/customization.
 - Script debugging tools.
 - Stats collection.
 - Browser simulation.
 - Reporting output.
 - Protocol support.
 - Ease of use.
- Easy to get involved (open standards, SDK, developer documentation).
- Support forums and response times.

WebLoad weaknesses:

- Key features require commercial upgrade:
 - Commercial support (from Radview).
 - Wider protocol support.
 - Load distribution.
- Small community at present.
- Built-in help feature not active.



TOOLS PRO AND AGAINST - OpenSTA

OpenSTA strengths:

- Data input (static and dynamic).
- Commercial tech support helpdesk.

OpenSTA weaknesses:

- Few supported platforms.
- Proprietary scripting language.
- Poor reporting and analysis features.
- Supported protocols -HTTP only.
- No development activity.
- Very poor developer tools and documentation.
- No direct company backing or support.



TOOLS PRO AND AGAINST - JMeter

JMeter strengths:

- Widest range of supported platforms.
- Very active support and bug fixing community.
- Highly modular, good developer documentation.
- Lots of third party plug-ins.

JMeter weaknesses:

- Time consuming to write scripts (no capture/replay).
- No configuration during execution.
- Very poor stats collection.
- Not scalable in GUI mode.



SERVER SIDE UTILITIES TO MONITOR PERFORMANCE

Usually the benchmarking tools come with server side monitoring capabilities.

WebLoad for example can poll data from a server using different protocols:

- rstatd – OS related statistics
- UC Davis – OS related stats using UC-Davis SNMP probe
- apache-snmp – Web server statistics

Every Linux distribution ships also many server side utilities that will help gathering and analyzing data related to the server performances.

The most common ones are the following:

- **top**
 - provides a dynamic real-time view of a running system.
- **vmstat**
 - reports information about processes, memory, paging, block IO, traps, and cpu activity
- **lsof**
 - lists information about files opened by processes.
- **ps**
 - displays information about a selection of the active processes.
- **oprofile**
 - profiling system for systems running Linux 2.2, 2.4, and 2.6



Q & A

Ten minutes for questions.



THE END

Thanks for attending **IMPORTANCE OF LOAD TESTS
BEFORE APPLICATION RELEASE** session.

