

# Work in Progress: Building a Distributed Generic Stress Tool for Server Performance and Behavior Analysis

Ada Casanovas  
Computer Architecture Department  
Technical University of Catalonia  
acasanovas@ac.upc.edu

Javier Alonso and Jordi Torres  
Computer Architecture Department  
Technical University of Catalonia  
Barcelona Supercomputing Center  
alonso@ac.upc.edu, torres@ac.upc.edu

Artur Andrzejak  
Zuse Institute Berlin  
andrzejak@zib.de

**Abstract**—One of the primary tools for performance analysis of multi-tier systems are standardized benchmarks. They are used to evaluate system behavior under different circumstances to assess whether a system can handle real workloads in a production environment. Such benchmarks are also helpful to resolve situations when a system has an unacceptable performance or even crashes. System administrators and developers use these tools for reproducing and analyzing circumstances which provoke the errors or performance degradation. However, standardized benchmarks are usually constrained to simulating a set of pre-fixed workload distributions. We present a benchmarking framework which overcomes this limitation by generating real workloads from pre-recorded system traces. This distributed tool allows more realistic testing scenarios, and thus exposes the behavior and limits of a tested system with more details. Further advantage of our framework is its flexibility. For example, it can be used to extend standardized benchmarks like TPC-W thus allowing them to incorporate workload distributions derived from real workloads.

**Keywords**—Benchmarking, Performance Analysis, Dependability.

## I. INTRODUCTION

The past decade has witnessed a significant evolution in the use of the Internet. Web Services span from a simple customer/provider paradigm to multi-tiered provider chains that constitute symbiotic business (and research) communities. In the early years of Internet users were tolerant to performance issues or faults. Nowadays these factors can no longer be neglected; in order to stay competitive, service providers must keep an eye on the quality of their services, or customers will run away to competition.

### A. Performance and quality of service

In the Internet community, feedback and reputation of a service is likely to spread without necessarily an actual intervention of the provider itself. Benchmarks display a clear and thorough study on how services behave, using several metrics to compare and evaluate their quality of service. They appear as key components of SLA (Service Level Agreements) which are contracts between customers and providers where the quality of the service (QoS) is expressed in terms of specific values

for availability, throughput capacity, latency and other metrics. Corporations like SPEC (Standard Performance Evaluation Corporation) or TPC (Transaction Processing Performance Council) have focused for years on the specification and design of standardized benchmarks [1][2]. Additionally, several tools have been designed with the same purpose: to put a given system under a magnifying glass and point out its flaws by emulating hundreds of users.

### B. Problem statement

System benchmarking tools attempt to simulate several typical deployment scenarios. For this purpose they provide a collection of different test models:

- The Stress Test aims to determine the limits of a system by using a constant burst load or increasing it until the server crashes.
- The Real Load Test uses an estimated load to test the system, focusing on the study of performance metrics under normal load to ensure the established QoS is met. HP's LoadRunner for example, uses a Controller to capture real request flow and reproduces it using load generators [3].
- Other methods are generally used, such as increasing the load step by step, using mathematical distributions, or altering it depending on feedback performance data from the server.

In the majority of above cases synthetic workload distributions are used, with the exception of products like LoadRunner. This approach provides an effective and fast way to generate infinite workload streams. However, the underlying mathematical models - while based on solid theoretical background and approximately accurate - cannot capture the complexity and unpredictability of a workload occurring in a real scenario. Consequently, when it comes to generating real load, e.g., reproducing a server output, existing tools provide only limited support, or none at all. In turn, the results of the evaluation possibly do not reflect the full spectrum of the scenarios encountered in a production environment.

### C. Paper contributions

The main goal of this paper is to present a new tool which is able to perform a stress test of a service under real load. In more detail, taking as input the traces of a real load or some concise description of a synthetic load, our tester generates a series of requests and issues these requests via a pool of distributed, emulated clients. Our design allows for a flexible configuration of the type of test as well as the type of service. Additionally it can be extended and tuned to fit the specific purposes of the user.

The paper has the following structure. Section II presents the related work. Section III describes the architecture of our framework and Section IV concludes the paper and presents the future work.

## II. RELATED WORK

There is a large variety of benchmarking tools. Currently, we can find tens of them available to evaluate multi-tier systems. Few of them which have gained the largest popularity in industry and academia are described below.

### A. Httpperf

Httpperf is a tool for UNIX-Like Operating Systems to measure web server performance [4]. Httpperf has a quick setup of stress tests. Furthermore, it implies a low overhead for the clients machines and exploits Unix System features to increase performance, offering rich output metrics. However, Httpperf cannot handle more than approximately 1020 concurrent connections, since it is the limit imposed by the maximum amount of file descriptors in Linux. Using Httpperf, we can not schedule requests as we want, to generate real workloads.

### B. LoadRunner

Mercury Interactive's LoadRunner is a load testing tool that analyzes system behavior and performance. It exercises the entire enterprise infrastructure by emulating thousands of users and employs performance monitors to identify and isolate problems. LoadRunner is distributed by Mercury Interactive company under a commercial license. The target systems are various, and most existing technologies are supported. There are two main types of scheduling, but both allow to customize a ramp up period, a duration period and a ramp down period. Real scenarios must be captured before reproducing them and not extracted from raw input. The schedule of requests can't be extracted from an input file and distributed in a custom manner.

### C. TPCW Java Implementation

TPC Benchmark W is a transactional web benchmark based on workload against an e-bookstore service [5]. It uses Emulated Browsers to send multiple requests for the same session. The TPCW implementation we have studied allows for a variety of interactions with the server. All data is stored in and retrieved from a remote database, the initial contents of the database are randomly generated. The sequence of requests is also random, nevertheless it follows a Markov chain, that

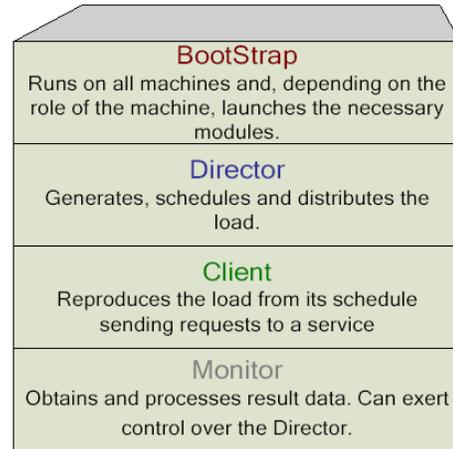


Fig. 1. Architecture overview of the tester tool

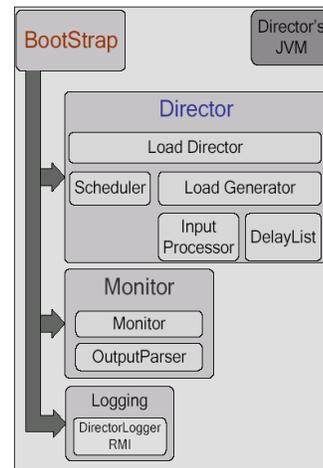


Fig. 2. Detailed Architecture of Director

is, the probability of a browser following a link depends on the page it is currently viewing. However, using TPC-W we cannot schedule requests. Metrics are printed and accessible only at the end of the execution.

## III. ARCHITECTURE OF THE FRAMEWORK

Obviously it is difficult to find a benchmark tool that allow to run experiments with real workload without paying licenses or modifying the code of the benchmark. Consequently, we developed a benchmarking framework (called "Tester") which allows us to add this capability to external benchmarks or develop an own client and add it as a plug-in to the framework.

The Tester's architecture comprises four components: Boot-Strap, Director, Client and Monitor. Figure 1 outlines their roles in the order of the deployment. As a choice of design, both the Monitor and the Director run in the same JVM, several Client JVMs can be distributed locally or remotely. In the following we briefly define the basic features of each component as well as the interactions between them.

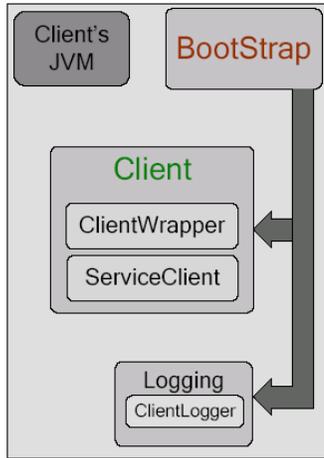


Fig. 3. Detailed Architecture of Clients

The Tester will first BootStrap all JVMs and determine their role, proceeding with the necessary steps to launch them. The Tester has been designed to run automatically with minimal configuration and without necessarily any user intervention during runtime. Therefore, all JVMs start the same execution whether they are Client or Director, local or remote. In order to achieve this, BootStrap locates all machines and determines which components have to be loaded. As shown in Figure 2 BootStrap launches the LoadDirector, Monitor and DirectorLogger in the single JVM where load generation and control is managed. Figure 3 presents the same process in the Client machine, where the ClientWrapper and the Client-Logger are started. The core components of the framework are the Director and the distributed Clients, both described in detail below. The final component is the Monitor, responsible for monitoring and decision making. It runs parallel to the Director, and collects logged data to analyze it during runtime. Metrics are extracted from log messages using a standard or custom OutputParser. In automatic mode, the monitor can additionally decide if the configuration used to pre-process the load must be modified.

#### A. Director

The mechanisms by which custom load input is processed and distributed are handled by three main components: the LoadDirector, the LoadGenerator and the Scheduler. The LoadDirector locates and communicates with the clients, supervising their execution and periodically sending them schedules for the requests they have to produce against the service. Finally the Scheduler is responsible for a consistent distribution of the load.

The LoadGenerator uses the DelayList module (a delay based schedule) to obtain the initial raw load (extracted from the trace file obtained from the system) and the InputProcessor to modify or add information to the load if necessary. Figure 4 presents a simple input file containing real traces and the details of the configuration file section where the DelayList

module behavior is defined. The first file is the workload representation from the real system, in this simple example we have only the timestamp and the number of requests processed by the system. In the second file, we define how the DelayList module interprets that information: reqScale multiply the number of requests of the file (we can change the workload) and timescale indicates the timestamp magnitude (default milliseconds).

#### B. Distributed clients

The ClientWrapper is the remote interface to a distributed client. It sends requests using the adequate ServiceClient to test server following a schedule. The ServiceClient represents a single test user. It is customary to the tested service. For example, in case of the TPC-W benchmark it corresponds to the Emulated Browser [5]. The output data is logged using the ClientLogger which dumps it to the client's local file system and can optionally forward log messages to the DirectorLogger that is running in the Director machine.

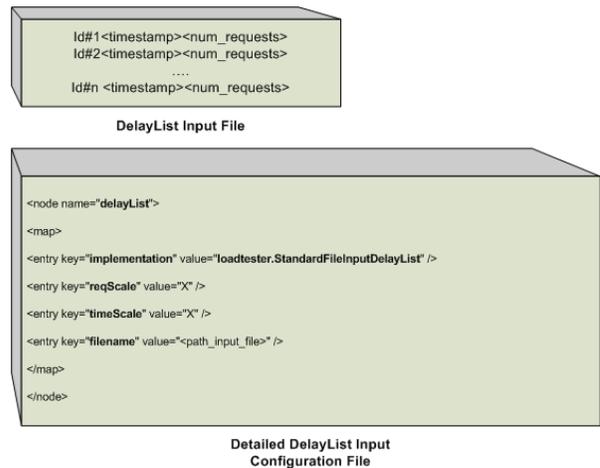


Fig. 4. Details of the input files

## IV. CONCLUSION AND FUTURE WORK

The Tester is a pure Java tool designed to collect input data representing a schedule of load with the purpose of testing a service. The main features of our design are the ability to collect the input, preprocess it, schedule it in order to distribute it among several JVMs and execute requests against any type of service using a service class programmed by our users. Other characteristics that define our project are the set of interfaces allowing users to plug-in their own versions for every key component of the tool, the Preferences Library for a rich customization of the scenarios, the bootstrap class used for client discovery, and the implementation of a TPCW Benchmark plugin. The Tester is an open project subject to modifications, we have implemented several interfaces to allow for users and developers to extend our initial implementation. The Preferences library offers the possibility to load a custom implementation of an interface and use it as a key component

of the scenario. We are currently working on a research project which aims to determine the behavior of a service under different circumstances. More specifically we intend to put a service under real load while injecting memory leaks and load bursts. Additionally we are working on improving the monitoring class to be capable of examining the service response and deciding whether we should inject load bursts, drop a client, or change the scale of the load. Using these new features of the Tester we will be capable of observing the service's behavior to determine its limits and to what extent memory leaks affect its performance.

During the process of building the Tester several new ideas and innovations have appeared. We offer a list of some of them we found interesting to comment.

- Allowing clients to interact with the LoadDirector and the Monitor through the servlets
- The creation of an implementation of the DelayList, the InputProcessor and the ClientWrapper to allow for multiple possibilities of building LTTasks.
- Adding a new feature modifying the LoadDirector, the Scheduler and the Monitor that customizes the distribution of the load depending on the current load and performance of the client machines.
- Building additional classes to allow for a fast creation of service clients similar to the ones we found on the tools we presented in the related work section.

#### V. ACKNOWLEDGMENTS

This research work is carried out in part under the FP6 Network of Excellence Core-GRID funded by the European Commission (Contract IST-2002-004265). Also this work has been supported by the Spanish Ministry of Education and Science (projects TIN2007-60625).

#### REFERENCES

- [1] Transaction processing performance council. <http://www.tpc.org>. 02.21.2009.
- [2] Standard performance evaluation corporation. <http://www.spec.org>. 02.21.2009.
- [3] Hewlett Packard. HP LoadRunner. *HP LoadRunner software - tips and tricks for configuration, scripting and execution - White paper*. <http://www.hp.com>. 02.21.2009.
- [4] D. Mosberger and T. Jin. *httperf A Tool for Measuring Web Server Performance*. HP Technical Report HPL-98-61, March 1998.
- [5] TPC-W in Java. <http://www.ece.wisc.edu/pharm/>. 02.21.2009.