FEDERAL UNIVERSITY OF PAMPA

Victor dos Santos Costa

LoadSun - Proposal of a Tool to Generate Workloads on Web Applications

Alegrete
2019

Victor dos Santos Costa

LoadSun - Proposal of a Tool to Generate Workloads on Web Applications

Term Paper presented in Software Engineering graduation course in the Federal University of Pampa as a partial requirement for obtaining a Bachelor's degree in Software Engineering.

Supervisor: Maicon Bernardino da Silveira

Co-supervisor: Elder de Macedo Rodrigues

Alegrete
2019

Victor dos Santos Costa

LoadSun - Proposal of a Tool to Generate Workloads on Web Applications

Term Paper presented in Software Engineering graduation course in the Federal University of Pampa as a partial requirement for obtaining a Bachelor's degree in Software Engineering.

Term Paper presented and approved on 26. nov....... of 2019
Committee members:

[illegible]

Prof. PhD. Maicon Bernardino da Silveira
Supervisor
UNIPAMPA

[illegible]

Prof. PhD. Elder de Macedo Rodrigues
Co-supervisor
UNIPAMPA

[illegible]

Prof. PhD. Fábio Paulo Basso
UNIPAMPA

[illegible]

Prof. Dr. Avelino Francisco Zorzo
PUCRS

ABSTRACT

The Quality of Service (QoS) of Web applications is usually measured by metrics such as response time, throughput, and availability. Low QoS translates into frustrated customers, which can lead to missed business opportunities. One way to assess application behavior is through performance testing, which allows us to evaluate how Web applications support the expected workload by emulating customer behavior at different load levels. Performance tests can prove difficult (not to say impossible) to be run without using some kind of automation tool. The main way to automate performance tests is through workload generators, which emulate synthetic customer behavior, by creating and managing virtual users. This term paper project presents a workload generator for Web-based applications, prioritizing the phases of analysis, design, implementation and evaluation. The proposal is called LoadSun, and its main focus is to be a lightweight tool that can be used by undergraduates and to be integrable with other modules such as a performance testing Domain-Specific Language and a performance testing monitoring tool. For the accomplishment of the analysis and the design an empirical research on performance testing tools was carried out, in the form of a systematic literature map, resulting not only in the proposed tool, but in the elaboration of a complete taxonomy of performance testing tools. LoadSun's presented design contains the necessary requirements, an abstraction of the architecture, and the major design decisions that were taken together with the reasoning behind them. An insight in the implementation of the tool and its peculiarities is given.
An experimental benchmark was conducted to evaluate and compare LoadSun with one of the industry's leading open-source performance testing tools. Finally, the results of the benchmark and the final conclusions are discussed, as well as new topics for future works.

Keywords: Workload generation, Software quality, Software testing, Performance testing.

RESUMO

A qualidade do serviço (Quality of Service - QoS) de aplicações web geralmente é medida por métricas como tempo de resposta, vazão e disponibilidade. Baixa QoS traduz-se em clientes frustrados, o que pode levar a perda de oportunidades de negócio. Uma maneira de verificar o comportamento das aplicações é por meio de testes de desempenho, que permite-nos avaliar como as aplicações web suportam a carga de trabalho esperada emulando o comportamento dos clientes em diferentes níveis de carga. Testes de desempenho podem provar-se difíceis (para não dizer impossíveis) de serem executados sem a utilização de algum tipo de ferramenta de automação. A principal maneira para se efetuar testes de desempenho é por meio de geradores de carga de trabalho, que emulam o comportamento dos clientes de forma sintética, criando e administrando usuários virtuais. Este trabalho apresenta uma proposta de ferramenta para geração de cargas de trabalho em aplicações baseadas na web, priorizando as fases de análise, projeto, implementação e avaliação. A proposta é chamada LoadSun, e seu foco principal é ser uma ferramenta leve que possa ser usada por estudantes de graduação e ser integrável a outros módulos, como uma Linguagem Específica de Domínio para teste de desempenho e uma ferramenta de monitoramento de testes de desempenho. A análise e o projeto basearam-se em uma pesquisa empírica, realizada sobre ferramentas de teste de desempenho, na forma de um mapeamento sistemático da literatura, resultando não apenas na ferramenta proposta, mas na elaboração de uma taxonomia completa de ferramentas de teste de desempenho. O projeto da LoadSun apresentado contém os requisitos necessários, uma abstração da arquitetura e as principais decisões de projeto que foram tomadas, acompanhadas do raciocínio por trás de cada uma delas. Também é fornecida uma visão sobre a implementação da ferramenta e suas peculiaridades. Foi realizado um benchmark experimental para avaliar e comparar a LoadSun com uma das principais ferramentas de teste de desempenho de código aberto do setor. Finalmente, são discutidos os resultados do experimento e as conclusões finais, além de novos tópicos para trabalhos futuros.

Palavras-chave: Geração de carga, Qualidade de software, Teste de software, Teste de desempenho.

LIST OF FIGURES

LIST OF TABLES

List of acronyms

AWS	Amazon Web Services
CR	Capture Replay
CS	Computer Science
DD	Design Decisions
DSL	Domain Specific Language
FR	Functional Requirements
GUI	Graphical User Interface
LESSE	Laboratory of Empirical Studies in Software Engineering
MBT	Model Based Testing
NFR	Non-Functional Requirements
PE	Performance Engineer
PTA	Probabilistic Timed Automata
QA	Quality Assessment
QoS	Quality of Service
RPS	Responses Per Second
RQ	Research Questions
RTT	Round Trip Time
SAM	Sequential Action Model
SE	Software Engineering
SLA	Service Level Agreement
SMS	Systematic Mapping Study
SPE	Software Performance Engineering
SUT	System Under Test
SWM	Stochastic Workload Model
UC	Use-Case

UML AD	UML Activity Diagram
UML UC	UML Use Case Diagram
UNIPAMPA	Federal University of Pampa
VU	Virtual Users

TABLE OF CONTENTS

1 INTRODUCTION

Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality (defined as having "value to some person") of a product or service. Myers e Sandler (2004) concluded that, when compared to other software development life cycle phases, software testing can represent up to 60% of the total development effort, consuming many critical resources like time and money. Although, in counterpart, empirical works provide many evidences suggesting that time invested in testing saves money in software projects (MYERS; SANDLER, 2004; PERRY, 2007; AMMANN; OFFUTT, 2016).

While the field has existed for over half a century, software testing is not without challenges, difficulties, and flaws. As software systems grow in both size and complexity, quality becomes significantly more difficult to ensure. These increasingly complex systems make existing testing issues more prominent and cause new issues to arise. Some of which are continuously attenuated by new and evolving automation technologies and methodologies.

Today's Web systems must support concurrent access by a huge number of users. An important component of testing these applications is load testing, which is frequently performed with the aim to ensure that a system satisfies a particular performance requirement under a heavy workload (it's important to highlight that in this context, workload refers to the rate of incoming requests to a given system). In fact, load testing involves evaluating performance of a system under normal and elevated load conditions. In such case, the SUT is not expected to process the overload without adequate resources, but to behave (*e.g.*, fail) in a reasonable manner (*e.g.*, not corrupting or losing data).

Performance tests measure and verify if the system is able to receive a large number of users without having its processing compromised, as maintaining the stability of the system means to keep for all users a good experience of interaction with the application (WOODSIDE; FRANKS; PETRIU, 2007).

In practice, load tests are rarely an integral part of the development process, as reported by Shams, Krishnamurthy e Far (2006). The main reasons (other than the lack of time or money) may be the common preconception that meaningful load tests are too difficult to create and maintain as well as short-sighted calculations showing that the benefits of load tests don't outweigh their costs.

Workload generation is one of the most fundamental aspects of performance evaluation. No system evaluation study can avoid confronting the problem of generation an amount of workload to test the system against, be it synthetic or natural. This is easily proved by observing that the performance counters to be evaluated in such a study are critically dependent on the workloads processed by the system being studied (FERRARI, 1984).

This paper term project reports the initial phases of the development of a solution to generate synthetic workloads and the complementary studies that were carried out to support this development.

1.1 Motivation

For modern Web applications, availability (the capacity to remain available) and short response times—even if accessed by large numbers of concurrent users, is critical. Cheung e Lee (2005) show that response times above one second cause dissatisfaction, which may make users look for a competitor solution.

Software performance tests are unlikely, if not impossible, to execute manually. Making it necessary to use tools that automate the process of generating workloads. A synthetic workload generation is essential to systematically evaluate performance properties of application systems under controlled conditions, in load tests or benchmarks.

In order to deliver significant results, performance testing demands the combination of good automation tools and an experienced performance engineer.

To meet the need for experienced engineers, testing activities are commonly found in undergraduate courses in the computer science area, but they only represent a small portion of the total hourly load when compared to the other activities in the software development process. Requiring undergrads to quickly-and sometimes, incorrectly, adapt and learn to use new tools.

The use of tools in the performance testing process, though common, is laborious and has a rigid entry barrier and a steep learning curve. In this way, there is a need for a tool that generates synthetic workloads with quality and in a simple way. Allowing the integration with other supplementary tools to access a systems performance under load.

The main motivation of this work was the importance of software load testing for the development of high quality and reliable Web-based systems and also the lack of easy to use open source tools that assist in the workload generation process.

1.2 Objectives

This study aims to propose the design and implementation of a workload generation tool for Web application systems. The focus of the tool is to generate synthetic workloads with a great number of simultaneous virtual users, that should interact and issue requests to a Web application. To achieve the main objective, some specific objectives were defined:

- **Research:** Conduct empirical research through the scientific literature to find and study proposed and applied workload generation tools and methods;

- **Design:** Acknowledge the necessary requirements of a workload generation tool;

Apply a simplified architecture: converting software characteristics such as flexibility, scalability, feasibility, re-usability, and security into a structured solution that meets the technical and educational expectations;

- **Develop:** Implement a prototype that can realize the conceived requirements, with the following characteristics:

  Open source: software with source code that anyone can inspect, modify, enhance, use and redistribute;

  Modular: allow for simultaneous execution with other performance monitoring and analysis tools;

  Efficient: the tool should generate heavy workloads with the minimum possible resource utilization;

- **Evaluate:** Access empirically if the tools meets the necessary characteristics to be a meaningful contribution;

- **Publish:** Release the tool and write a term paper to report the results of this study.

1.3 Research Synthesis

Based on the objectives presented in Section 1.2, to formalize this study, the following research synthesis was elaborated:

Table 1 – Research synthesis.

Subject	Performance Testing
Topic	Workload Generation
Research Question	Would the state of practice benefit from a open source workload generation tool for Web-based applications to integrate a performance testing solution?
Hypothesis	The performance testing practice would benefit from a open-source workload generation tool that is easy to learn, lightweight and integrable.
Main Goal	To develop a workload generation tool for Web applications.

1.4 Contribution

A Systematic Mapping Study (SMS) was carried out to identify and consolidate related works, thus providing empirically supported knowledge that can assist and guide decision making processes as well as directing future research in the area.

The conduction of the SMS resulted in a taxonomy on software performance testing tools, that should help reduce the gap between practice and research in this body of knowledge, especially when it comes to the terms used and the approaches implemented in each one.

This term paper project designed and proposed an open source, simple and easy to use tool that can generate automatized synthetic workloads in Web applications and can be integrated with other tools to easily access the performance of software on the Web.

The proposed workload generation solution will also serve the purpose of being an independent module composing a more complete solution developed by the Laboratory of Empirical Studies in Software Engineering (LESSE)[1] in order to be a free, open source solution that covers all the performance testing activities such as load generation, scripting through Model Based Testing (MBT), performance monitoring, and automated analysis of results. It's focus being the use in undergraduate student performance testing disciplines that are offered at Federal University of Pampa (UNIPAMPA) in Software Engineering (SE) and Computer Science (CS) courses.

The aforementioned SMS, taxonomy, and design of the tool will also contribute to the body of knowledge in the form of scientific publications, some of which have already been sent for publication and some still in planning.

1.5 Organization

This document is organized as follows:

- **Chapter 1: Introduction -** Introduced the body of knowledge, presented the motivations and the main objectives of this study;
- **Chapter 2: Methodology -** A compilation information based on the analysis of the proposed problem and the steps required to arrive at the design and implementation;
- **Chapter 3: Background -** Provides context for the information discussed in this paper;
- **Chapter 4: Related Work -** This chapter empirically analyzes previous work with a SMS;
- **Chapter 5: LoadSuns's Design -** Provides details of the design of the proposed monitoring tool;
- **Chapter 6: LoadSun's Implementation -** Explains and discusses LoadSun's implementation peculiarities;
- **Chapter 7: Experiment: LoadSun's Benchmark -** Defines the benchmark protocol and analyze it's main results;
- **Chapter 8: Conclusions and Future Work -** Presents the conclusions, indicates possible future work and a schedule indicating all the work that has already been done.

[1] LESSE's website is available at: <http://lesse.com.br/>

2 METHODOLOGY

This chapter will describe the scientific methodology that supports this study. Section 2.1 introduces what it is and why it is important. In Section 2.2 the research is classified accordind to Prodanov e Freitas (2013), and the research design is shown in Section 2.3.

2.1 Introduction

Minayo, Deslandes e Gomes (2015) consider research as the basic activity of Science in its inquiry and construction of reality. It is the research that feeds the teaching activity and updates it in front of the reality of the world. Therefore, although it is a theoretical practice, research links thought and action.

Scientific research is the accomplishment of a planned study, the scientific aspect of the investigation is characterized by the method of approach of the problem. Its purpose is to answers questions by applying the scientific method. Research is always part of a problem, of an interrogation, a situation for which the repertoire of available knowledge does not generate an adequate response.

The most important aspect must be the emphasis, the concern in the application of the scientific method rather than the emphasis on the results obtained. The criteria for the classification of research types vary according to the given approach, aims, fields, methodologies, situations and objects of study.

2.2 Research Classification

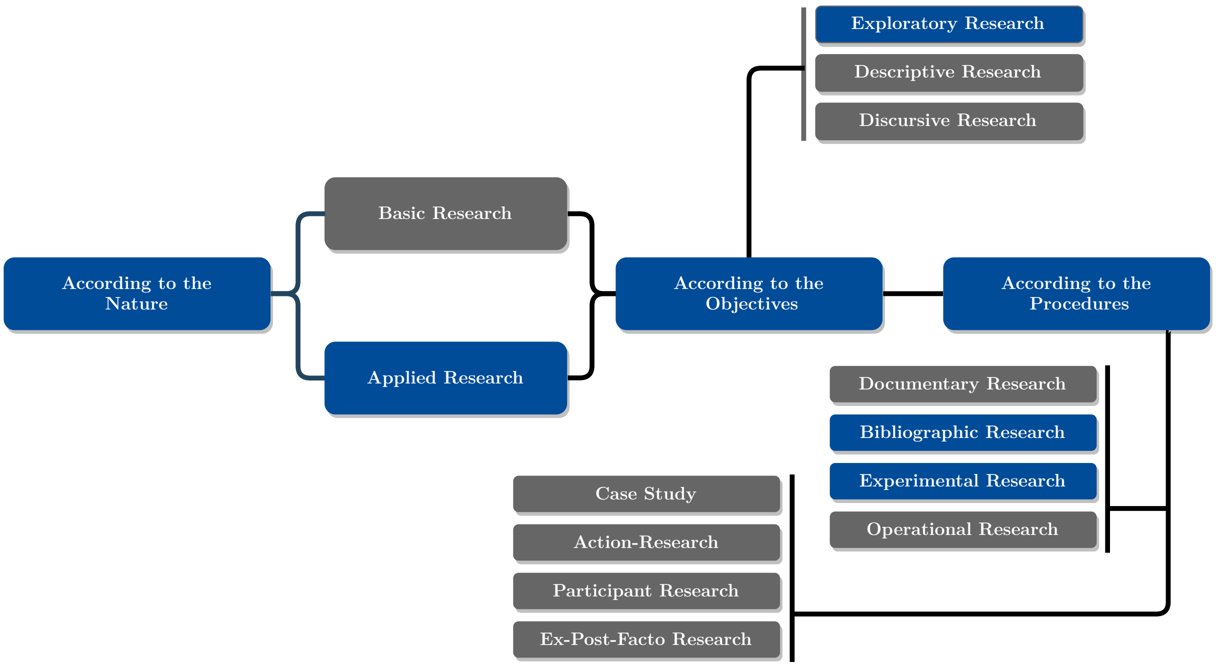
This research has been classified according to Prodanov e Freitas (2013) classification scheme. Figure 1 shows the classification of this research according to its nature, objectives and procedures, the highlighted terms classify this research and are described in this section.

From the nature point of view this is an **Applied Research**, which aims to generate knowledge for practical application directed to the solution of specific problems. Involving local truths and interests.

This research is classified as an **Exploratory Research** from the objectives point of view because one of it's purpose is to provide more information about the subject that is going to be investigated, allowing its definition, that is, to facilitate the delimitation of the research theme; guide the setting of objectives and the formulation of the hypotheses or discover a new type of approach to the subject. This type of research assumes, in general, the forms of **Bibliographical Researches** which this study applies in a SMS.

According to the Procedures this research also fits in the **Experimental Research** classification in which an object of study, in this case the proposed tool, was determined, the variables that would be able to influence it were selected, and the forms

Figure 1 – Research Classification



Source: adapted from Prodanov e Freitas (2013).

of control and observation of the effects that the variable produces on the object were defined along with another object for comparison.

2.3 Research Design

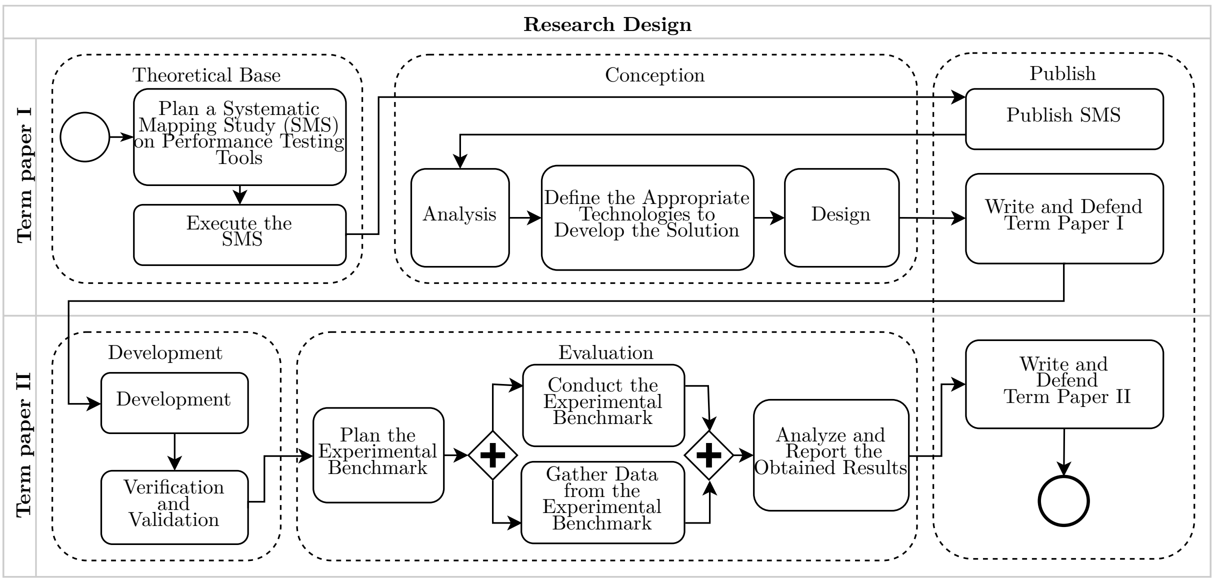
To conduct this study a research design was developed. In this design, the activities were grouped in five (5) phases: theoretical base, conception, publish, development and evaluation. Which are described in this section and can be observed in Figure 2.

The theoretical base group seeks to build an empirical based knowledge for the development of this work and includes the activities of planning and conducting an SMS which are described in Chapter 4. The publication activities group encompasses the writing and publication of the SMS and term papers I and II.

In the conception phase, the analysis was performed and the essential requisites were defined and documented, as well as the definition of the appropriate technologies to develop the solution. From those results a solution was designed and can be seen in Chapter 5. Once this is done, the development and implementation of a testable prototype will begin in the development phase, so that the testing can be performed and the prototype evaluated.

The prototype evaluation activities are contained in the evaluation group and include the planning of a controlled experimental benchmark, the execution of the benchmark parallel to the data collection and, afterwards, a report with the results obtained will be developed and included in term paper II.

Figure 2 – Research Design


Research Design
Term paper I
Term paper II
Theoretical Base
Plan a Systematic Mapping Study (SMS) on Performance Testing Tools
Execute the SMS
Conception
Analysis
Define the Appropriate Technologies to Develop the Solution
Design
Publish
Publish SMS
Write and Defend Term Paper I
Development
Development
Verification and Validation
Evaluation
Plan the Experimental Benchmark
Conduct the Experimental Benchmark
Gather Data from the Experimental Benchmark
Analyze and Report the Obtained Results
Write and Defend Term Paper II

Source: the author.

2.4 Chapter Summary

This chapter provided an idea of what the methodology is and how this research can be classified. In addition, the established research design was presented, so that the lane that deals with term paper I provided an understanding of what processes have been performed so far, and the lane term paper II indicates what is planned to be executed.

3 BACKGROUND

In this chapter, the definitions of the terms used in this study are provided. Section 3.1 introduces the general context of software testing, while software performance engineering and software quality is explained in Section 3.2. Section 3.3 defines performance testing and its main “types”. Some background on performance testing tools is given in Section 3.4, along with their main activity, workload generation, in Section 3.5.

3.1 Software Testing

Software testing is a broad term encompassing a variety of activities along the development cycle, aimed at different goals and is an essential activity in SE. In the simplest terms, it amounts to observe the execution of a software system to validate whether it behaves as intended and identify potential bottlenecks.

The purpose of testing a program is to find errors in it, and the purpose of finding errors is to fix them. The main benefit of testing is that tests result in improved quality, so it is widely used in industry for quality assurance: indeed, by directly inspecting the software in execution, it provides a realistic feedback of its behavior and as such it remains an inescapable complement to other analysis techniques.

Beyond the apparent straightforwardness of checking a sample of runs, however, testing embraces a variety of activities, techniques and actors, and poses many complex challenges. Indeed, with the complexity, pervasiveness and criticality of software growing ceaselessly, ensuring that it behaves according to the desired levels of quality and dependability becomes more crucial, and increasingly difficult and expensive (BERTOLINO, 2007).

If you attempt to test everything, the costs go up dramatically and the number of missed bugs declines to the point that it’s no longer cost effective to continue. If you cut the testing short or make poor decisions of what to test, the costs are low but you’ll miss a lot of bugs. The goal is to hit that optimal amount of testing so that you do not test too much or too little.

As you can not possibly test everything an important skill to develop is how to minimise a large number of tests into manageable tests set, and make wise decisions about the risks that are important to test and which are not (PATTON, 2005).

3.2 Software Performance Engineering

Software performance is a pervasive quality difficult to understand, because it is affected by every aspect of the design, code, and execution environment. By conventional wisdom performance is a serious problem in a significant fraction of projects. It causes delays, cost overruns, failures on deployment, and even abandonment of projects, but such failures are seldom documented.

A highly disciplined approach known as Software Performance Engineering (SPE) is necessary to evaluate a system's performance, and to improve it.

Woodside, Franks e Petriu (2007) defines SPE as "the entire collection of Software Engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements."

Two approaches are found in the literature, the most common is purely measurement-based; it applies testing, diagnosis and tuning late in the development cycle, when the system under development can be run and measured (*e.g.* (ARLITT; KRISHNAMURTHY; ROLIA, 2001; AVRITZER et al., 2002; BARBER, 2004; BARBER, 2003)), performance problems detected then, tend to be corrected by just adding additional hardware. The model-based approach, pioneered under the name of SPE by Smith (1990), creates performance models early in the development cycle and uses quantitative results from these models to adjust the architecture and design with the purpose of meeting performance requirements.

Like other Software Engineering activities, SPE is constrained by tight project schedules, poorly defined requirements, and over-optimism about meeting them. Nonetheless adequate performance is essential for product success, making SPE a foundation discipline in software practice.

3.3 Performance Testing

With the growing demand to serve a greater number of customers, there has been an increasing number of Web services and applications to allow multiple users to access simultaneously the system resources. Gao, Tsao e Wu (2003) define performance testing as the activity to validate the system performance and measure the system capacity. To maintain it's availability and performance as expected, it is necessary to simulate user activities under several conditions. In this context, through software performance testing, it is possible to develop effective strategies to maintain system performance at an acceptable level.

These tests aim at verifying whether the system performance is in line with design expectations, subjecting it to a certain load at a given computing environment.

In a general view, performance testing is a qualitative and quantitative evaluation of a SUT under realistic conditions to check whether performance requirements are satisfied or not. As a quality control task, the performance testing must be performed on any system before delivering it to customers. On a testing road map, performance testing comes at the end of the test plan, after functional testing including conformance, integration and interoperability takes place (DENARO; POLINI; EMMERICH, 2004). However, the evaluation of performance is particularly important in early development stages, when important architectural choices are made and sometimes, overlooked.

There are many types of performance testing depending on what you want to test your software against, some that are worth mentioning include (KHAN; AMJAD, 2016; PUTRI; HADI; RAMDANI, 2017):

- **Load testing:** In load testing the application has been checked under anticipated user load. The main aim of this testing is test all the bottlenecks before delivering the software to the customer;
- **Stress testing:** In stress testing the Web application checked under extreme workloads to see the breaking point of an application;
- **Endurance testing:** In endurance is has been checked that the software can handle the expected load over a long period of time;
- **Spike testing:** When sudden large spikes in the load generated by users then spike testing is used;
- **Volume testing:** In this type of testing a large number of data has been stored in the database and the overall Web applications behaviour is observed. The main objective of this type of testing is to check the correctness of the application under varying database volumes;
- **Isolation Testing:** Isolation testing is an uncommon thing for a performance testing, but it involves repetition in the execution of the test that resulted in system issues. Hence, isolation testing is often used to isolate and confirm the fault domain;
- **Configuration Testing:** A configuration test is conducted to determine the impact of configuration changes for component performance and system behavior. A common example is experimenting with different methods of load balancing, or whether virtual or physical is the better system environment setting.

3.4 Performance Testing Tools

Performance testing tools address applications and systems design problems by testing their scalability and reliability. Unlike functional testing tools, performance and load testing tools establish a performance baseline and then attempt to find out performance bottlenecks by adding up workload.

A performance testing tool adds up workload by generating multiple instances of Virtual Users (VU)s and setting up those VUs to interact with the SUT in different ways, which determine the different user profiles of the SUT.

A wide variety of tools for this purpose can be seen in Section 4 which reports the execution of an SMS on performance testing tools.

Those tools may cover specific tests which include the aforementioned spike tests, stress tests, endurance tests, load tests, or others. The main objective of performance testing applications is to automate or even, enable the testing of a multitude of interactions with a particular system at the same time. As those types of test may prove difficult, or even, impossible to execute in a manual manner.

Some tools help the Performance Engineer (PE) to establish benchmarks for a target system. A target might be a server, a Web application, a group of servers, or a whole network (as in cloud-based applications). Performance testing tools allow for the system to be monitored in real-time. Results should supply root cause analysis and trace bottlenecks. Additionally, performance testing tools can provide an analysis and calculate Service Level Agreement (SLA) (LEE; BEN-NATAN, 2002) compliance to offer an overall view of system resilience.

3.5 Workload Generation

Workload generation is the primary task of any performance testing tool, without an adequate workload the defects are not identified and then, become impossible to be corrected. Performance tests can last for hours, days, and even weeks, a test that does not find flaws or defects results in valuable time being wasted.

All performance analysis techniques, *i.e.* the techniques that can provide us with the values of a system's performance counters, require one or more workload models to be built. This is not only the case of analytic and simulation modeling techniques, but also of both types of measurement approaches mentioned by Ferrari (1984): the one in which the SUT is driven by an artificial (synthetic) workload, that is supposed to represent a real (current or future) workload, as well as the one which uses a sample of the system's natural workload.

To reach reliable conclusions about SLA and capacity requirements, based on the results of a performance test, the synthetic workload used must be representative of real workloads. A synthetic workload is said to be representative of a real workload if both workloads result in similar performance when submitted to a system (FERRARI, 1984).

The workload generation task must always be performed externally to the SUT, since it must generate thousands or even millions of VUs capable of interacting with the system, resulting in an enormous consumption of computational resources. Another reason for this is that workloads should simulate as closely as possible a real interaction scenario with the system, that is, including network latency, thinking time, error prone users, and others.

In most tools, load generation includes configurations of common variables to define the workload model, such as: the number of VUs, which indicates the number of synthetic users that will interact with the system in total; ramp-up time, which dictates the time that users take to start accessing the system; ramp down time, indicating the time the VUs take to leave the system; think time, which defines the time between each action of the VUs, simulating the response time of a real user; among others.

Most of the different test types presented in Section 3.3 can be achieved merely by configuring the workload model variables in different ways, *e.g.* to perform a spike test we set up a large number of VUs so that they access the system all at once in a very short ramp up time.

3.6 Chapter Summary

This chapter served the purpose of presenting the main areas of knowledge addressed in the rest of this study. Section 3.1 has introduced the term and what it means for software engineering as well as the role it plays in the development cycle.

In Section 3.2 the definition of SPE is detailed and the way it helps achieve performance goals is exemplified. Afterwards, Section 3.3 explains what performance tests are in the context of SE, the main reasons for this activity and its most common types.

It has been explained in Section 3.4 that performance testing tools are indispensable for running this type of test, followed by the description of workload generation, the main task the tools implement, in Section 3.5.

The following chapter presents the protocol formulated to perform the systematic mapping study on performance testing tools.

4 SYSTEMATIC MAPPING STUDY

This section reports the SMS that was conducted in order to propose and design the LoadSun tool (Chapter 5).

4.1 Protocol

A SMS identifies, selects and critically appraises research in order to answer a clearly formulated question (BARN; BARAT; CLARK, 2017). The SMS should follow a clearly defined protocol or plan where the criteria is clearly stated before the mapping is conducted. It is a comprehensive, transparent search conducted over multiple databases that can be replicated and reproduced by other researchers. It involves planning a well thought out search strategy which has a specific focus or answers a defined question. The mapping identifies the type of information searched, critiqued and reported within known time frames.

This SMS is built on the guidelines for performing SMS in Software Engineering proposed by Kitchenham (2007).

4.1.1 Scope and Objective

With the purpose of provide an empirical reference for professionals and researchers who search for new tools or tools that have certain particularities in the development and execution of performance testing, the objective of this study is to identify and characterize existing performance testing tools in the literature. In addition, we aim at identifying the academic and open source tools and finding out their quality attributes. In this sense, the description of the goal is described according to the GQM (Goal, Question, Metric) paradigm (KOZIOLEK, 2008) and can be observed in Table 2.

Table 2 – Objective according to the GQM paradigm.

For the purpose of:	*identify / characterize*
With respect to:	*performance testing tools*
From the viewpoint of:	*performance test engineers and researchers*
In the context of:	*performance testing environment*

Source: Adapted from Koziolek (2008).

4.1.2 Question Structure

The research questions (RQs) are structured based on the Population, Intervention, Comparison and Result (PICOC) (KITCHENHAM, 2007) criteria: **P**opulation: published research on software; **I**ntervention: performance testing; **C**omparison: general comparison of the retrieved tools; **O**utcome: performance testing tools; **C**ontext: both academic and industrial context.

4.1.3 Research Questions

The following Research Questions (RQ) are defined.

RQ1. What are the tools that support performance testing? Our goal is to find out which quality attributes are associated with these tools, their reported strengths and limitations.

RQ2. What characterizes a performance testing tool? In order to answer this question, the following sub-questions are needed:

RQ2.1. What are the elaboration approaches of the test scripts interpreted by the performance load generators?

RQ2.2. What performance testing monitoring approaches are applied?

RQ2.3. What are the persistence strategies of metrics data collected by performance testing monitors?

4.1.4 Search Process

Formal literature research was conducted using only databases that: (i) have a search engine capable of using keywords; and (ii) contain computer science documents. The selection includes the following bases: Association for Computing Machinery (ACM) Digital Library[1], Engineering Village[2], IEEE Xplore[3], ScienceDirect[4], SCOPUS®[5] and SpringerLink[6]. To define the search string the terms and synonyms presented in Table 3 were used, as well as, the Boolean operator "OR" to select alternative terms and synonyms, and the Boolean operator "AND" to add more terms to the string. The resulting string can be seen in Figure 3.

Table 3 – Search string definition.

Terms	Synonyms
Performance Test	Load Test, Stress Test, Soak Test, Spike Test, Workload Test, Automation Test
Tool	Generator, Injector, Monitor, Analyzer, Framework, Suite, Environment, Plug*in
Software	Application, System

[1] ACM: <https://www.dl.acm.org>
[2] Engineering Village: <https://www.engineeringvillage.com>
[3] IEEE: <https://www.ieeexplore.ieee.org>
[4] ScienceDirect: <https://www.sciencedirect.com>
[5] SCOPUS®: <https://www.scopus.com>
[6] SpringerLink: <https://www.link.springer.com>

Figure 3 – Search String.

```
(("Performance Test" OR "Load Test" OR "Stress Test" OR "Spike Test"
OR "Soak Test" OR "Workload Test" OR "Automation Test") AND (Tool OR
Plugin OR Plug-In OR Framework OR Generator OR Monitor OR Injector OR
Suite OR Analyzer OR Environment) AND (Software OR System OR
Application))
```

4.1.5 Inclusion and Exclusion Criteria

IC1. The publication should report the use of a tool that supports performance testing;

IC2. The publication should propose a tool to support performance testing;

EC1. Duplicated studies;

EC2. The publication is not related to performance testing in the software area. *e.g.* performance testing of an engine;

EC3. The publication is written in a language other than English;

EC4. The publication is only available in the form of abstract, slide show, poster or short paper;

EC5. The publication is not available for download;

EC6. The publication does not report or propose a performance testing tool.

4.1.6 Quality Assessment Criteria

The purpose of using Quality Assessment (QA) criteria is to evaluate the power from selected studies to answer some research question. The QA criteria is used in two stages: the former stage being the individual evaluation of each researcher, to reduce the probability of bias; the latter stage where the researchers should reach a consensual note about the publications in a "divergent state" in the quality measurement grade.

Each of the cited QA criteria is evaluated by each researcher, according to the following degree: Yes (Y) = 1; Partial (P) = 0.5; No (N) = 0. So the total score ranging the five questions can result in: 0-1.0 (very bad); 1.5 or 2.0 (regular); 2.5 or 3.0 (good); 3.5 or 4.0 (very good); and 4.5 or 5.0 (excellent). Each of the criteria and their possible evaluations are described below:

QA1: Does the publication make a contribution to the software performance testing field?

Y: A contribution is explicitly defined in the publication;

P: A contribution is implied;

N: No contributions could be identified.

QA2: Does the publication characterize a software performance testing tool?

Y: The publication proposes and demonstrates the use of a tool;

P: The publication proposes or demonstrates the use of a tool, never both;

N: No, the publication does not propose or demonstrate the use of a tool.

QA3: Does the publication apply any type of empirical evaluation?

Y: The publication explicitly applied an evaluation (for example, a case study, an experiment or proof of correctness);

P: The evaluation is a "Toy" example;

N: No evaluations could be identified.

QA4: Does the publication present some type of analysis, showing results?

Y: The publication presents some type of analysis or shows the results obtained;

P: The publication presents only a summary of the results;

N: No form of analysis or result were presented.

QA5: Does the publication describe the techniques used in load generation and monitoring?

Y: The publication explicitly describes load generation and monitoring techniques;

P: The publication describes either load generation techniques, or monitoring techniques, never both;

N: The publication does not describe any load generation or monitoring techniques.

4.1.7 Selection Process

The selection process is divided in five stages, which are performed by two researchers. The process steps as well as the researchers involved are described below:

(1) *Initial selection*: The search strings were generated using the selected keywords and synonyms adapting for each of the databases particularities. The initial selection encompassed all papers up to 2019 (exclusive), resulting in a total of 1673 studies. An initial selection was performed by researcher one, according to criteria EC1, EC2 and EC4 (see Section 4.1.5);

(2) *Eliminate redundancies*: at this stage, researchers one and two worked together on a pre-analysis of articles to eliminate redundancies. After the removal of duplicates, 1160 different papers remained.

(3) *Intermediate selection*: at this stage, researchers one and two read separately the title and the abstract (reading the introduction and conclusion when necessary) of each study. Here, the researchers decided to select or reject an article following IC1, IC2, EC1 - EC6 (see Section 4.1.5);

(4) *Final selection and elimination of discrepancies*: At this stage, all other studies were completely read by researchers one and two, who applied the same criteria for the intermediate selection. In case of disagreement/divergence, a third researcher would read the studies and discuss whether or not the study should be included in the final selection. This resulted in the inclusion of 146 papers;

(5) *Quality assessment*: Based on the quality criteria (see Section 4.1.6), we assessed the power of the studies to answer our research questions. The quality criteria were evaluated independently by the two researchers; therefore, reducing the probability of erroneous and/or biased results. Then researchers agreed in a consensual note on the publications that received a divergent grade. Papers that achieved at least a total score of 3 (good) and received a Yes (Y) response in QA2 were selected for data extraction. The final selection was composed of 53 papers that reported a total of 38 performance testing tools.

4.1.8 Data Extraction Strategy

To extract the relevant data from the selected publications, we produced a form that would help to answer the RQs and also to check the QA criteria. The following data were extracted for each study: title; year of publication; authors; name of the tool presented; type of license supported by the tool (commercial, academic, open-source); language or types of script supported; supported classes and types of metrics in respect to performance monitoring; reports generated on the tests performed; architecture and organization of data persistence.

An important issue during data extraction was solved in a way that both researchers acted as data extractors and also as data verifiers, thus reducing the probability of errors and/or bias in data extraction[7]

The data presented here were manipulated using the Thoth[8]. This tool assisted in the whole process of this mapping, supporting the classification and extraction of data, the selection and qualification of the papers and also aided in visualizing the results.

[7] All artifacts used in the SMS are available at the Google Drive repository: <https://drive.google.com/open?id=1ZMkMWL7EyDSAHdiAtaXpXmUWwoVXO7YG>

[8] Available at: <http://lesse.com.br/tools/thoth/>

4.1.9 Data Analysis Strategy

The data was tabulated to show: The list of published tools, its licensing and their source in Table 4 (addressing **RQ1.**); The list of published tools, supported input approaches of each and its categorization in Table 5 (addressing **RQ2.1.**); The list of published tools, their quality attributes including monitored metrics and its categorization in Table 6 (addressing **RQ2.2.**); The list of published tools, and the persistence strategies of each tool in Table 6 (addressing **RQ2.3.**).

4.2 Systematic Mapping Results

In this section we discuss the answers to our RQs (see Section 4.1.3). In each case, we indicate the utility of these results for researchers and practitioners.

4.2.1 RQ1. What are the tools that support performance testing?

The purpose of this question is to map the tools used or proposed by scientific studies that support some kind of performance testing. In total, thirty eight (38) performance testing tools were identified through our SMS. Table 4 lists these tools, their license type and the studies where they were found. Most of the tools were cited only once or twice, while some of then have been heavily referenced (11 and 9 times) showing a clear preference and greater adoption of these tools, namely LoadRunner and Apache Jmeter, the former being a commercial tool, while the latter is open source.

4.2.2 RQ2. What characterizes a performance testing tool?

In order to find any problems in software, the main characteristic of a performance testing tool is that it should generate a certain workload on a target system (SUT). These problems may be related to scalability, reliability, or any system bottlenecks, and this can occur in a variety of ways.

Each tool can have unique characteristics in its implementation. However, despite adopting distinct features and strategies, it is perceived that tools developed for this purpose make use of an already consolidated architecture.

Users of these tools may need to select a tool for a specific purpose, and selecting the most appropriate one may become a problem based on a lack of information about them. Therefore, we propose in this work a new taxonomy based on this extensive literature mapping, represented by feature model in Figure 5 The groups and elements of our taxonomy are presented and explained collectively in section 4.3, including the classification of 38 tools found in the literature during the execution of this mapping. We believe this taxonomy will assist others in the process of identifying, categorizing, developing, and deploying new tools or features for performance testing and monitoring tools.

Table 4 – Tools and references.

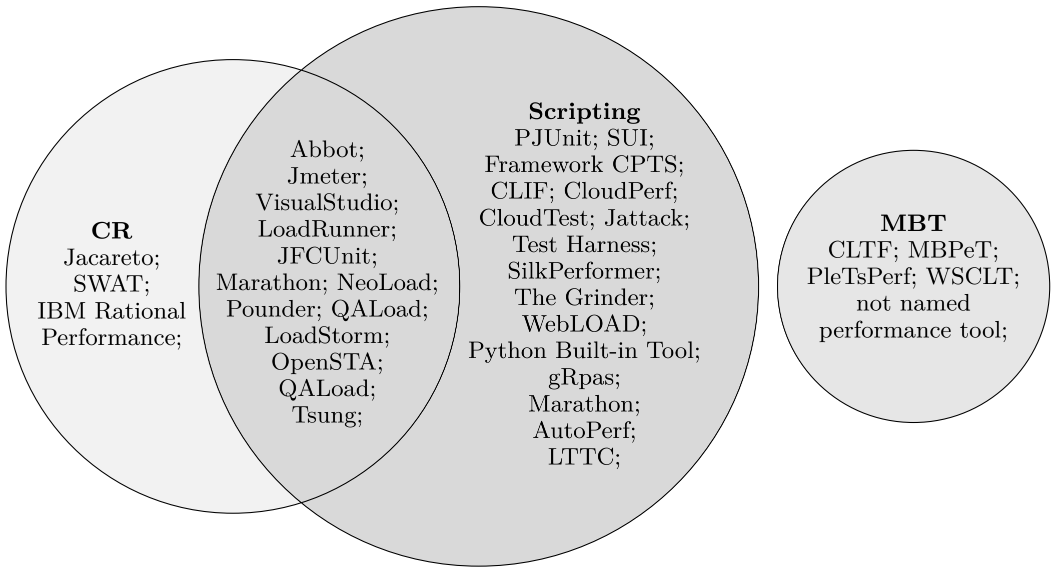
References	Tool Name	License Type
(JOVIC et al., 2010)	Abbot	Open-Source
(KIRAN; MOHAPATRA; SWAMY, 2015; PUTRI; HADI; RAMDANI, 2017; AGNIHOTRI; PHALNIKAR, 2018; APTE et al., 2017; ZHANG et al., 2011; SINGH; SINGH, 2012; WU; WANG, 2010; PODELKO, 2016; KRIŽANIĆ et al., 2010)	Apache JMeter	Open-Source
(APTE et al., 2017)	AutoPerf	N/D
(ZHANG et al., 2011)	Framework CPTS	Commercial
(KIM; KIM; CHUNG, 2015; DILLENSEGER, 2009)	CLIF load injection framework	Open-Source
(ZHOU; ZHOU; LI, 2014)	Cloud Load Testing Framework (CLTF)	N/D
(MICHAEL et al., 2017)	CloudPerf	Commercial
(PODELKO, 2016)	CloudTest	Commercial
(CUCOS; DONCKER, 2005)	gRpas	N/D
(JOVIC et al., 2010)	Jacareto	Open-Source
(AMIRANTE et al., 2016)	Jattack	Open-Source
(JOVIC et al., 2010)	JFCUnit	Open-Source
(DEVASENA; KUMAR; GRACE, 2017)	Load Testing Tool for Cloud (LTTC)	N/D
(ZHANG et al., 2011; NETTO et al., 2011; KHAN; AMJAD, 2016; CHUNYE; WEI; JIANHUA, 2017; LI; SHI; LI, 2013; YAN et al., 2011; PU; XU, 2009; KALITA; BEZBORUAH, 2011; PODELKO, 2016; HAMED; KAFRI, 2009; RODRIGUES et al., 2014)	LoadRunner	Commercial
(PODELKO, 2016)	LoadStorm	Commercial
(JOVIC et al., 2010)	Marathon	Open-Source
(ABBORS et al., 2013)	MBPeT	Academic
(PODELKO, 2016; KRIŽANIĆ et al., 2010)	NeoLoad	Commercial
(KIM; CHOI; WONG, 2009)	PJUnit	Open-Source
(RODRIGUES et al., 2015; RODRIGUES et al., 2014)	PLeTsPerf	N/D
(JOVIC et al., 2010)	Pounder	Open-Source
(FAN; MU, 2013)	Python Built-in Tool	Open-Source
(KRISHNAMURTHY; ROLIA; MAJUMDAR, 2006)	Session-based Web Application Tester (SWAT)	N/D
(KIM; KIM; CHUNG, 2015), (PODELKO, 2016)	Silk Performer	Commercial
(BRUNE, 2017)	Simulating User Interactions (SUI)	Academic
(KAMRA; MANNA, 2012)	Test Harness	Commercial
(ZHANG et al., 2011; KRIŽANIĆ et al., 2010)	The Grinder	Open-Source
(RODRIGUES et al., 2014)	Visual Studio	Commercial
(KRIŽANIĆ et al., 2010; HABUL; KURTOVIC, 2008; YAN et al., 2011), (PU; XU, 2009)	WebLOAD	Commercial
(YAN et al., 2014; YAN et al., 2012a; YAN et al., 2012b)	WS-TaaS	N/D
(MAâLEJ; HAMZA; KRICHEN, 2013)	WSCLT	N/D
(STUPIEC; WALKOWIAK, 2013)	Not Named Tool	N/D
(ZHANG et al., 2011; YAN et al., 2011; PU; XU, 2009)	IBM Rational Performance Tester (RPT)	Commercial
(YAN et al., 2011; PU; XU, 2009)	QALoad	Commercial
(APTE et al., 2017)	Tsung	Open-Source
(PU; XU, 2009)	Etest	N/D
(PU; XU, 2009)	OpenSTA	Open-Source
(YAN et al., 2011; PU; XU, 2009)	WAS	N/D

4.2.3 RQ2.1. What are the elaboration approaches of the test scripts interpreted by the performance load generators?

The goal of this research question is to explore different kinds of approaches for workload input definition and elaboration, and determine whether these types of input could be classified into different categories. Three main categories were observed: Model-

Based Testing (DALAL et al., 1999), Capture and Replay (MEMON; SOFFA, 2003), and Manual Scripting. The dispersion of tools within these categories is shown in Figure 4 and Table 5 specifies which kind of model and/or scripting language each tool supports. Choosing a tool whose model or scripting language is best known by the test engineers can result in a smaller learning curve in its use, and fewer errors when creating test scenarios. The tools that stood out most in this area were JMeter and LoadRunner, the tools were shown to support a greater number of different input types, which could explain why they were the most mentioned in **RQ1**.

Figure 4 – Input approach Venn diagram.



Source: the author.

4.2.4 RQ2.2. What performance testing monitoring approaches are applied?

Performance monitoring is an ongoing process of data collection and analysis to compare how well a project, program, or policy is being implemented in relation to the expected (KRIŽANIĆ et al., 2010). This task is fundamental in the software development life cycle and is also part of the preventive software maintenance cycle. Performance monitoring tasks are facilitated with the employment of monitoring tools. Most performance testing tools have dedicated features for monitoring, while others utilize a dedicated tool for monitoring.

Performance monitoring tools typically provide analysis of specific metrics and notifications about critical changes in the system. The selection of an appropriate tool for monitoring should be given in relation to which metrics one wishes to collect to analyze the performance requirements of the application being tested.

To answer this research question in detail, the data were classified according to the monitoring approaches found in the primary studies resulting from this SMS and in accordance with the metrics selected in subsubsection 4.3.3.1. The first approach refers

Table 5 – Tools and workload input approaches.

	CR	MBT					Scripting Language															
Tool Name	**Capture Replay**	**PTA**	**SAM**	**SWM**	**UML AD**	**UML UC**	**.Net**	**BeanShell**	**C**	**C#**	**C++**	**Clojure**	**Groovy**	**Java**	**JavaScript**	**JSON**	**Jython**	**Python**	**Ruby**	**Scala**	**SCL**	**XML**
Abbot	✓													✓								
Apache JMeter	✓							✓					✓		✓					✓		
AutoPerf																						✓
CLIF																						✓
CLTF			✓																			
CloudPerf																						✓
CloudTest															✓							
gRpas																						✓
IBM RPT	✓																					
Jacareto	✓																					
Jattack															✓							
JFCUnit	✓													✓								✓
LTTC														✓								
LoadRunner	✓						✓		✓					✓	✓							
LoadStorm	✓															✓						✓
Marathon	✓																		✓			
MBPeT		✓																				
NeoLoad	✓														✓							
Not Named Tool				✓																		
OpenSTA	✓																				✓	
PJUnit														✓								
PLeTsPerf					✓	✓																
Pounder	✓													✓								
Python Built-in Tool																		✓				
QALoad	✓										✓			✓								
SWAT	✓																					
Silk Performer										✓												
SUI														✓								
Test Harness														✓				✓				
The Grinder												✓					✓					
Tsung	✓																					✓
Visual Studio	✓								✓													
WebLOAD											✓											
WSCLT		✓																				
Etest* Framework																						
CPTS*																						
WAS*																						
WS-TaaS*																						

∗It was not possible to find information on the input approach of these tools in the literature.
Probabilistic Timed Automata (PTA); Sequential Action Model (SAM); Stochastic Workload Model (SWM); UML Activity Diagram (UML AD); UML Use Case Diagram (UML UC).

Table 6 – Monitored metrics and Persistence strategies.

	Monitored Metrics											**Persistence Strategies**						
	Application							**SUT Resources**										
Tool Name	**Response time**	**Hits per sec.**	**Responses per second**	**Transactions per second**	**Transaction success rate**	**#Virtual Users**	**Total test time**	**CPU**	**Memory**	**I/O**	**Network**	**Proprietary files**	**Databases**	**Html**	**JSON**	**Plain Text**	**JDBC Drivers**	**XML**
Abbot	✓							✓	✓									
Apache JMeter	✓	✓	✓	✓	✓	✓	✓	✓			✓				✓		✓	✓
AutoPerf	✓					✓		✓			✓					✓		
CLIF load injection framework	✓				✓			✓	✓		✓						✓	✓
Cloud Load Testing Framework (CLTF)	✓																	
CloudPerf	✓							✓	✓	✓	✓		✓					
CloudTest	✓	✓		✓		✓	✓	✓		✓					✓			
Framework CPTS	✓	✓	✓	✓	✓	✓		✓			✓		✓		✓			
gRpas	✓						✓	✓		✓								✓
IBM Rational Performance Tester	✓	✓	✓		✓			✓	✓	✓	✓		✓					
Jacareto	✓							✓	✓									✓
Jattack	✓				✓			✓							✓			✓
JFCUnit	✓							✓	✓				✓		✓			✓
Load Testing Tool for Cloud (LTTC)	✓					✓		✓	✓	✓	✓							
LoadRunner	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓		
LoadStorm	✓	✓			✓		✓	✓	✓	✓	✓							✓
Marathon	✓							✓	✓									
MBPeT	✓				✓	✓		✓	✓	✓	✓							
NeoLoad	✓			✓		✓		✓	✓		✓				✓			
Not Named Tool	✓							✓										
OpenSTA	✓						✓	✓										
PJUnit	✓										✓				✓			✓
Pounder	✓							✓	✓						✓			✓
Python Built-in Tool		✓					✓	✓	✓		✓							
QALoad						✓	✓					✓						
Session-based Web Application Tester (SWAT)	✓							✓		✓	✓							
Silk Performer	✓				✓	✓	✓	✓	✓		✓							
Simulating User Interactions (SUI)	✓							✓	✓	✓	✓				✓			✓
Test Harness	✓							✓			✓				✓			
The Grinder	✓			✓				✓	✓	✓	✓							
Tsung	✓			✓		✓		✓		✓	✓							✓
Microsoft Visual Studio	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓					✓
WAS	✓							✓			✓							
WebLOAD	✓					✓		✓	✓	✓	✓				✓	✓		✓
WS-TaaS	✓	✓		✓	✓	✓	✓	✓	✓	✓				✓				
WSCLT	✓			✓			✓	✓			✓							
PLeTsPerf*																		
Etest*																		

∗ It was not possible to find information on the monitored metrics or persistence strategies of these tools in the literature.

to metrics directly related to the application, such as: response time, hits per second, responses per second, transactions per second, transaction success rate, number of virtual users, and total test time. The second approach presents metrics related to the resources from which the SUT is hosted, which are classified as CPU, memory, I/O and network utilization.

All tools analyzed use metrics of the two approaches represented. In general, the tools monitored the SUT metrics more than the application itself. The reason is that, in the application metrics approach, the data obtained during and after the tests execution needed to be interpreted to be shown in a clear and objective way to the user. Meanwhile, the metrics related to the SUT are only data captured at certain moments in the workload execution and shown to the user.

The results obtained during this classification were represented in Table 6, where it is possible to visualize each monitoring metric that the tool in question has.

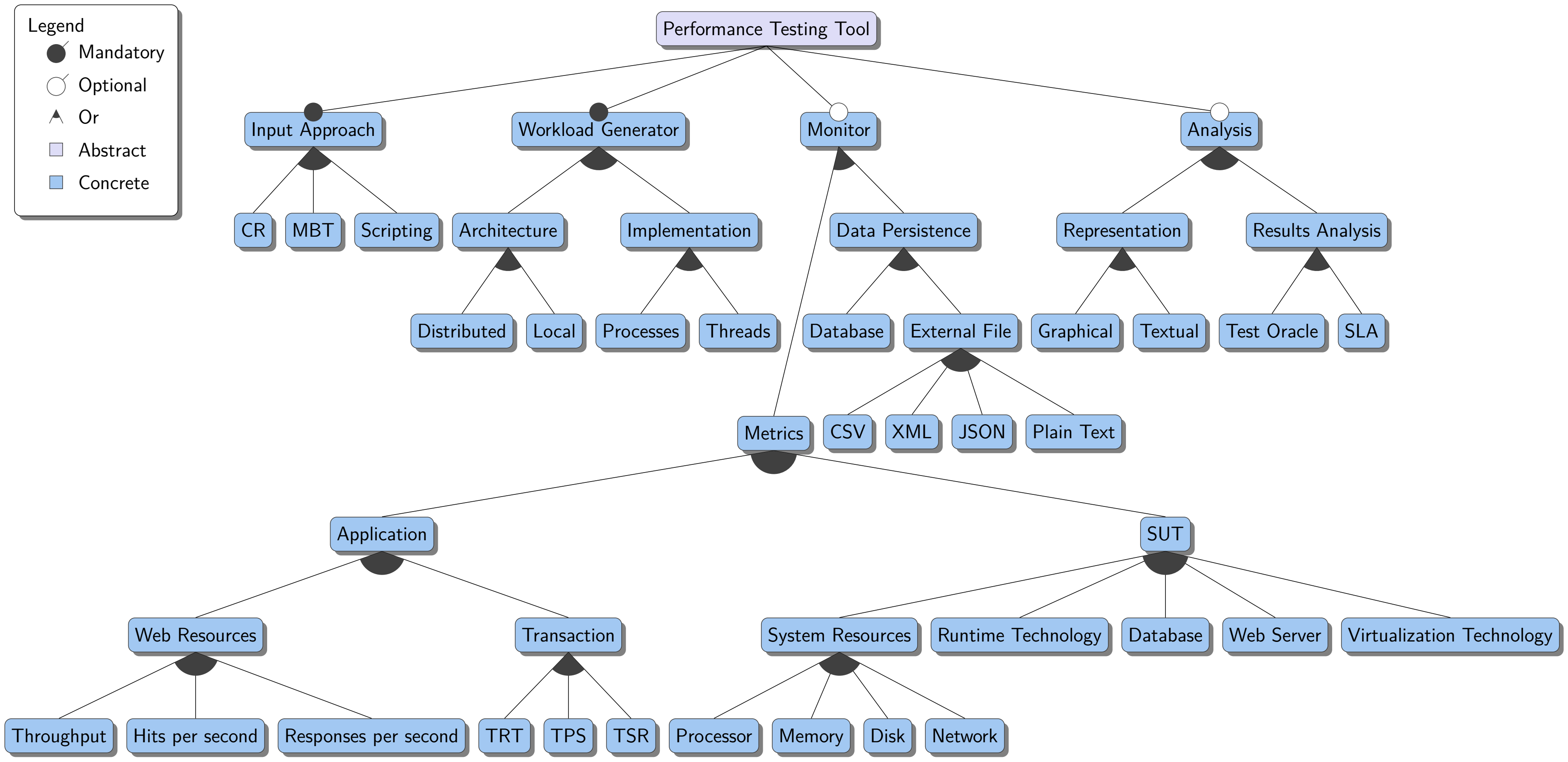
4.2.5 RQ2.3. What are the persistence strategies of metrics data collected by performance testing monitors?

To evaluate the performance of a system, it is necessary to monitor its behavior during workload execution. This results in a high-volume of data persisted for later analysis, thus making the implementation of a persistence layer necessary. As consequence, most persistence layers will use external files for persistence or underlying database management system. So the file types that have been observed as most common are XML and JSON, respectively (see Table 6). Finally, as for database management systems, no patterns or preferences were identified.

4.3 Taxonomy of Performance Testing Tools

A taxonomy is a scientific method of classification according to an established system in a specific domain, with the resulting catalog used to provide a framework for analysis. Any taxonomy should take into account the importance of separating elements of a group into subgroups that are unambiguous, and taken together include all possibilities (CLARKE; MALLOY, 2001).

Figure 5 – Taxonomy of performance testing tools represented by feature model.



Source: the author.

The main objective of our taxonomy (see Figure 5) is to reduce the gap between practice and research in performance testing tools, especially when it comes to the terms used and the approaches implemented in each one. This taxonomy provides means of comparison and evaluation of the tools features that can be useful in deciding which tool to use or how to design future systematic mappings and reviews. Not all features of the tools represented in the taxonomy were planned to be identified in our initial research research perspective, but are nevertheless identified and are represented in the taxonomy.

4.3.1 Input Approach

This section describes the input approaches that the tools support and/or provide means of elaboration. They were divided as follows:

4.3.1.1 Capture Replay

Capture Replay (CR), also known as Record and Playback, is a technique where a test engineer performs the tests manually once in the application. This is, by interacting with the Graphical User Interface (GUI) in “capture” mode, the tool stores this interaction and outputs a test script that can be “replayed” by the tool multiple times by several VU. The continuous modification of a GUI may render these types of tests obsolete, forcing the test engineers to re-capture those tests. However, modern CR tools do not rely solely on coordinates for test case execution but maintain extra information such as the handle, type, and label (if any) of the elements, enabling the replayer to locate the element when it has been moved (MEMON; SOFFA, 2003). Sometimes, this technique also employees manual script editing for the removal of random generated values, hard coded values and enhancements whenever possible.

4.3.1.2 Model-Based Testing

The MBT approach involves developing and using a data model to generate tests. The model is essentially a specification of the inputs to the software, and can be developed early in the cycle from requirements information. It can be especially effective for systems that are changed frequently, because testers can update the data model and then rapidly regenerate a test suite, avoiding tedious and error-prone editing of a suite of hand-crafted tests (DALAL et al., 1999) or even tests that were created using the CR technique. In our research we were able to find tools using PTA, SAM, SWM, UML AD, and UML UC as input.

4.3.1.3 Scripting

Manual script writing technique in which the test engineer manually writes a set of code statements, into a defined programming language, that will be executed by the load generator in the form of VUs.

4.3.2 Workload Generator

This group represents the often called “module” of workload generation. Its the core of many performance testing tools, it is responsible for interpreting the scripts and generating the correspondent workload in the SUT. It employs the creation and management of multiple VU, which can be executed locally or in a distributed manner, utilizing a master/slave approach.

4.3.2.1 Architecture

The architecture of a load generator deals mainly with the organization of its elements, which can be organized in a: (i) **Local** architecture when the VU are created and run in a single machine. This severely impacts the quality of the results obtained through performance testing, since they rely on the amount of workload that can be generated and maintained in a SUT. Limiting the load generation to one machine only, however broad it may be, limits the amount of load that can be generated, creating a bottleneck in the load generator itself; (ii) **Distributed** architecture load generators on the other hand, as the name implies, distribute the load of generating VU in a master/slave approach. This architecture enables having a local master controller that handles the test distribution and execution on the slaves, which are remote instances that will send the requests to the SUT. This architecture adds another layer of complexity onto load generators, as the test engineers will have to set up multiple computers and/or utilize cloud services, such as Azure[9], Amazon Web Services (AWS)[10] or Google Cloud[11].

Another difficulty when utilizing distributed architectures, is how to handle parameterized data in tests. This is because that first, if the load generator master controller does not handle the distribution of parameters, you will need to have separated files, and second, if the test engineer wants to update them, he has to go through each slave node to make the modifications.

4.3.2.2 Implementation

Characterizes the low level representation for load generators implementations. We were able to identify, albeit not in all cases, two different implementation approaches: (i) creating different **processes** for each instance of a VU, which do not share the same memory space, and are independent to each other. This is important for VU isolation, so that a problem within an instance of a VU does not affect the rest; (ii) The use of multiple **threads** for the VU execution, which shares the same memory address, lowering

[9] Azure: https://azure.microsoft.com
[10] AWS: https://aws.amazon.com
[11] Google Cloud: https://cloud.google.com

the communication cost between VUs. On the other hand, a problem within a VU will certainly affect the others and the reliability of the load generator itself.

4.3.3 Monitor

Refers to the monitoring modules present in some tools, the metrics they monitor as well as the data persistence approaches taken.

4.3.3.1 Metrics

A software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses. Metrics capture a value pertaining to systems at a specific point in time, like the number of users currently logged in to a web application or CPU usage. Therefore, metrics are usually collected once per second, per minute, or at another regular interval to monitor a system over time.

There are two important subcategories of metrics in our taxonomy:

(1) Application Metrics: indicates the top-level health of the system by measuring its useful output.

(i) **Web Resources**: These are vital performance counters for assessment of Web application ability to maintain the workload simulated. (a) *Throughput*: Shows the amount of server throughput during each second of the load test scenario run. Throughput measures the actual rate at which work requests are completed; (b) *Hits Per Second*: Shows the number of requests per second; (c) *Responses Per Second*: shows the number of HTTP status codes returned from the Web server during each second of the load test scenario run, grouped by status code.

(ii) **Transaction**: During load test scenario execution, VUs generate data as they perform transactions. This metric enables collecting data that shows the transaction performance and status throughout script execution, in which are presented as follows: (a) *Transaction Response Time (TRT)*: Different response time values under different load. Average response time, maximum, percentile, and so on; (b) *Transaction Per Second (TPS)*: Shows the number of transactions generated per second; (c) *Transaction Success Rate (TSR)*: Shows the number of transactions that passed, failed, or stopped.

(2) SUT Metrics: Most components of software infrastructure serve as a resource for monitoring systems. ***System Resources:*** Some resources are low-level, *e.g.* a server's resources include such physical components as processor, memory, disks, and network. Each one of them have a list of performance counters that could be used to measure the performance requirements of SUT, such as:

(i) **Processor**: Program execution threads consume processor (CPU) resources. Available performance counters measure how much CPU processing time threads and other executable units of work consume. These processor utilization measurements allow to determine which applications are responsible for CPU consumption. The processor performance counter are presented as follows: (a) *% Processor Time*; (b) *% Interrupt Time*; (c) *Processor Queue Length.*

(ii) **Memory**: A shortage of RAM is often evident indirectly as a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator. When observing a shortage of available RAM, it is often important to determine how the allocated physical memory is being used and count resident pages of a problematic process known as its working set. Instances of memory performance counters are shown as follows: (a) *Available Bytes*; (b) *Working Set*; (c) *Page Reads/Sec.*

(iii) **Disk**: Through the I/O manager stack, an operation system maintains physical and logical disk operations. A physical disk is the internal representation of specific storage device.It is important to be proactive about disk performance because it tends to degrade rapidly, particularly when disk-paging activity occurs. Examples of disk performance counters are presented, such as: (a) *Avg. Disk secs/transfer*; (b) *% Idle Time*; (c) *Disk Transfers/Sec*; (d) *Avg. Disk Queue Length.*

(iv) **Network**: Networking performance has become ever more important today with the proliferation of distributed and cloud applications. Network interface statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received. Networking bottlenecks are tricky to catch and analyze. Packet rates, collision rates and error rates do not always point to the cause of the problem. (a) *Bytes Total/Sec*; (b) *Server Bytes Total/Sec*; (c) *Connections Established.*

Runtime Technology: Application performance also depends on the architectural level monitoring and tuning. However, architectural design is built upon specific technologies. Each platform differs in which metrics and counters impact on the application performance. Common examples of runtime technologies are Java Platform Enterprise Edition (Java EE) or the .NET Framework.

Database: It is imperative to ensure optimal performance of the database as this is essential to any data-driven application. There are many factors affecting overall application performance that may come from the database side, such as: Poor database design; Poor logic used in queries; Database server machines dedicated to multiple applications.

Web Server: The function of a Web server is to service requests made through the HTTP protocol. Some Web servers even provide modules presenting information on server activity for automating the monitoring process.

Virtualization Technology: Virtualization platforms provide the service of creating a virtual (software) version of hardware. This adds another layer to complexity and computational efforts which also needs to be monitored and tuned for better results performance wise. Virtualization technology metrics can be very similar to those of System Resources, depending on the virtualization platform.

4.3.3.2 Data Persistence

The most common approaches for storing the data results from monitoring are the use of external files (like CSV, XML, JSON or even as plain text) or databases systems, for instance SQL or NoSQL.

4.3.4 Analysis

Refers to how the data results from the monitoring is processed and represented in performance testing reports.

4.3.4.1 Representation

How results are being presented to the test engineer, it could be a graphical and/or textual data representation using different techniques to generate a performance testing report. For instance, Word, PDF or HTML documents.

4.3.4.2 Result Analysis

What techniques, methods or approaches of automatic data analysis the tool applies in the measured data results. For instance, a test oracle or a SLA (LEE; BEN-NATAN, 2002).

4.4 Threats to Validity

In this section, the threats identified in the context of this study are described as suggested by Cook e Campbell (1979).

Construct Validity: This is a threat that affect the statements in this paper: provide an empirical reference that serves as a starting point decision making in selecting testing performance tools. In this sense, it is important to reassert that our analysis is built on well accepted guidelines for performing SMS in SE proposed by Kitchenham (2007), including a research. First of all, systematic mappings are known for not guaranteeing the inclusion of all the relevant works on the field. This can be explained by the limitation of the search mechanisms for set of keywords defined in this study and the lack of them in some of the relevant works. In order to avoid this bias, when known articles from the body of knowledge were not returned by the search, the search string was reformulated until the research encompasses all known articles.

Internal Validity: This type of threats is related to how we ensure that the performed analysis is valid to the problem statement. Likewise, in order to reduce possible bias, the stages of selection of the studies and data extraction were carried out by two researchers. The results found by each were tabulated and compared, so that any kind of bias could be identified, and when in disagreement, the authors could debate and a consensus was reached.

External Validity: The use of a well-defined and validated protocol assures that any other group of researchers could replicate this SMS using the same set of parameters would yield the same results. The only variable that could compromise this assumption is time, as new researches and tools emerge everyday. To minimize this threat the update of this SMS will be required in the future.

Conclusion Validity: This research found a relatively good number of focused papers, thus providing a statistical power to drive our conclusions. This could be affected by terminological problems in the search string, which may have led to the absence of some primary studies. For the minimization of these problems, the generated string was tested and the results were previously analyzed in a way that one could notice the relevance of the same. When necessary, the search string was modified and the process was redone. Finally, we reduced the threat of not indexing all available content on the web by using six (6) digital libraries.

4.5 Conclusions

In order to characterize and define some features from performance testing tools and software monitoring, this SMS aimed to analyze the reports from experiences and publications from the literature. In this sense, a SMS was carried out to identify and consolidate related work, thus providing a good document that can assist and guide decision making processes as well as directing future research to the area.

One of the main contributions of this SMS is the conception of a performance testing tool taxonomy. We hope our contribution will serve as the starting point for evolution of the research area, capturing the main tool features. Our goal was to provide empirically supported means for comparison and evaluation, thus a useful instrument for selecting the right tool for the right job.

As described by Engström e Petersen (2015), the popularization and availability of a taxonomy to the stakeholders is essential: 1) Considering the context of researchers and practitioners of performance testing, a good terminology is only achievable through peer reviewed scientific publications; and 2) by receiving further input to extend the taxonomy with new categories to reflect on its usage. Hence, this SMS allowed the description of the target problem addressed, the specification of research questions, and finally the answers to the questions elaborated. Initially, 1673 articles were found from 6 digital libraries. After applying the protocol, 53 primary studies were selected, thus used to answer the questions as a mean to accomplish the two essential criteria reported by Engström e Petersen (2015).

With the summarization, it was possible to verify that the number of papers published in the area. This number suggests a growth curve from 2014, reaching its apex in 2017. Thus, our conclusion is that the research area is in expanse, demonstrating an increase of interest by the academic community.

4.6 Chapter Summary

This chapter presented an SMS on performance testing tools. The developed protocol and it's execution was described in Section 4.1, its results, the tools found, and the answers to the research questions are discussed in Section 4.2.

The main contribution of this mapping is described in Section 4.3 in the form of a taxonomy of performance testing tools.

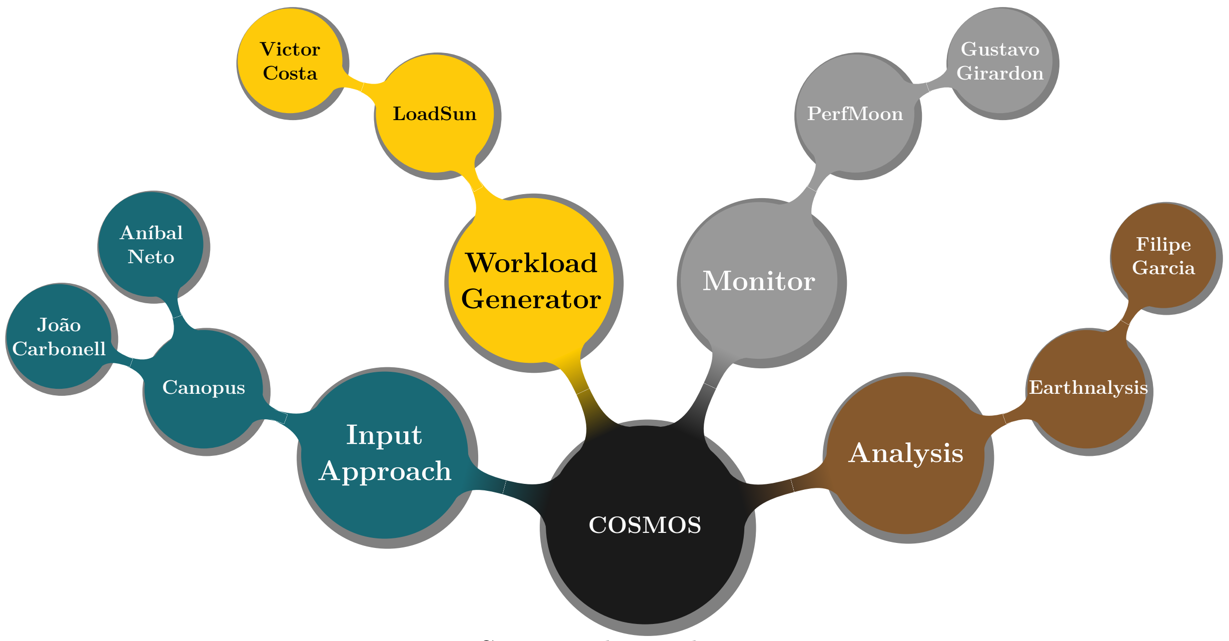
5 LOADSUN'S DESIGN

This chapter serves the purpose of describing the development of the solution and the thought process behind it.

5.1 Context

LoadSun is designed to work as a standalone module in a more complete solution for performance testing that is under development at LESSE. The complete tool is intended to be used by undergraduate students in performance testing learning in disciplines offered at UNIPAMPA, covering all the main aspects necessary to perform performance tests such as the creation of test scenarios and scripts, generation of synthetic workloads, performance monitoring and automated analysis of the results. Each module of COSMOS and its creators can be observed in Figure 6.

Figure 6 – COSMOS - LESSE's Performance Testing Solution.



Source: the author.

With the educational aspects and the idea of integration in mind the workload generation solution was designed to work in a simple way and with a low entry barrier, however, seeking to maintain the important characteristics of a workload generator, i.e., generating workloads at a low cost of computational resources and be adaptable to different types of workload models to support the different types of performance tests mentioned in Section 3.3.

5.2 Software Requirements

This section lists the requirements of the proposed solution, these were based on the needs found from the studies resulting from the SMS, as well as the prior knowledge and experience acquired during the author's graduation in SE. The requirements analysis was directly related to the design decisions described in Section 5.3 and the architecture described in Section 5.4.

- ***FR1. VUs should be able to make requests in the HTTP protocol:*** This requirement is necessary for the VUs to be able to interact with web-based systems.

- ***FR2. The solution must allow the configuration of workload models to cover the main types of performance testing:*** The configuration of workload models will allow the tester to define more realistic workloads, thus achieving better test results. It will also allow the tester to perform various types of performance testing[1].

  FR2.1. The solution must allow the definition of the amount of VUs that will access the SUT: The quantity of VUs is one of the main factors that characterizes a workload model, and the tester must be able to choose the load that best represents a realistic workload for his system.

  FR2.2. The solution should allow the configuration of the ramp up time interval: This requirement is necessary so that, according to the type of test, VUs gradually start accessing the system the according to the choice of the tester.

  FR2.3. The solution should allow setting the amount of VUs in each ramp up interval: To better control the access interval of the VUs in the system, the amount of VUs that would access the system at each interval must be set by the tester.

  FR2.4. The solution should allow the configuration of the ramp down time interval: Just like the ramp up time, the ramp down time must also be configurable so that the user can set intervals for the VUs to leave the system.

  FR2.5. The solution should allow setting the amount of VUs in each ramp down interval: The amount of VUs that will leave the system at each ramp down interval must be configured as well as the ramp up amount.

  FR2.6. The solution should allow setting the VUs think time: The inclusion of think time (which are timed intervals between one action and another) in VUs is of great importance so that each VUs represents a user of the system in a more realistic way, i.e., to click on a button and issue a request, an actual user would take at least the time to recognize that the screen has loaded and move the mouse to where the button is located.

[1] The types of performance testing are listed in section 3.3

- ***FR3. The solution must implement the option of informing the performance monitor that it started testing, as well as informing the amount of VUsers currently testing:*** An important aspect of the solution is that it should allow the user to execute the load generator together with the load monitor from the COSMOS performance solution, to integrate these two modules the load generator must send information that it started testing before generating the actual load on the system, as well as keeping the monitor informed about the quantity of VUsers currently in operation.

- ***FR4. The solution must implement the option of data parametrization through external files:*** It's important that the tool accepts external files with information about the parameters for the VUsers to utilize, this should be done through .csv files.

  FR4.1 The solution must allow for sequential selection of parameters. The load generator must allow the tester to set the parametrization to be in sequential order, from top to bottom.

  FR4.2 The solution must allow for random selection of parameters. The load generator must allow the tester to set the parametrization to be randomized within the parameters available in the parametrization files.

  FR4.3 The solution must allow for "sameAs" selection of parameters. The "sameAs" refers to the selection of parameters on the same line where random or sequential ones were selected. For example during login, the tester can choose to select a random login and a password that is the "sameAs" the random selected login.

- ***NFR1. It must be made available under an open-source license:*** In order to be used in the teaching process and so that other developers and researchers can evolve the tool in a collaborative way, the adopted license should be open-source.

- ***NFR2. The solution must generate load in a way that consumes the least possible computational resources:*** The lower the computational cost of generating the workload, the generation capacity of each instance of the load generator will be greater.

- ***NFR3. The solution must be possible to be integrated with other modules:*** In order to fully evaluate the performance of a system the solution must take into account the implementation and the communication protocols utilized by the other modules developed at the LESSE group, primarily the inputs generated by the test case scripting module.

A summary of the Functional Requirements (FR) can be seen in Table 7, and the Non-Functional Requirements (NFR) in Table 8.

Table 7 – Functional Requirements.

ID	Description
FR1.	VUs should be able to make requests in the HTTP protocol.
FR2.	The solution must allow the configuration of workload models to cover the main types of performance testing.
FR2.1.	The solution must allow the definition of the amount of VUs that will access the SUT.
FR2.2.	The solution should allow the configuration of the ramp up time interval.
FR2.3.	The solution should allow setting the amount of VUs in each ramp up interval.
FR2.4.	The solution should allow the configuration of the ramp down time interval.
FR2.5.	The solution should allow setting the amount of VUs in each ramp down interval.
FR2.6.	The solution should allow setting the VUs think time.
FR3.	The solution must implement the option of informing the performance monitor that it started testing.
FR4.	The solution must implement the option of data parametrization through external files.
FR4.1.	The solution must allow for sequential selection of parameters.
FR4.2.	The solution must allow for random selection of parameters.
FR4.3.	The solution must allow for “sameAs” selection of parameters.

Table 8 – Non-Functional Requirements.

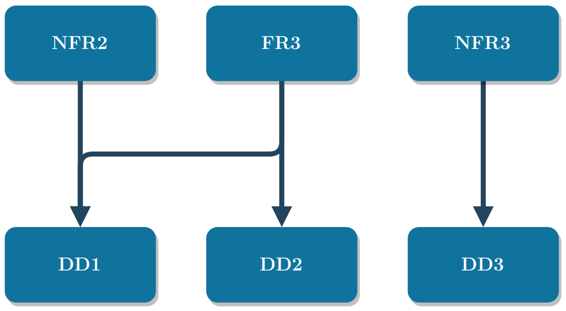
ID	Description
NFR1.	It must be made available under an open source license.
NFR2.	The solution must generate load in a way that consumes the least possible computational resources.
NFR3.	The solution must be possible to be integrated with other modules of COSMOS Performance Solution.

5.3 Design Decisions

This section discusses the Design Decisions (DD) that have been made for the development of LoadSun, so that they help us meet the requirements (described in section 5.2) as best as they can. Figure 7 maps which requirements influenced which design decisions.

- ***DD1. Programming Language:*** The decision of which programming language to use is important for LoadSun to keep a low computational cost and that NFR2 is reached with greater ease. This decision required analyzing different languages for the specific applications we want to give them, in this case, the generation of multiple VUs concurrently in multiple threads. Strong candidates for this service are the languages C++ and Go (created by google and launched as open source in

Figure 7 – Requirements and design decisions.



Source: the author.

2009), both languages are compiled and provide the necessary performance, C++ stands out a little more in this case, however, the fact of Go facilitates working with parallelism and the low difference in performance was significant in the decision to use ***Go*** in the development of LoadSun.

- ***DD2. Architecture:*** LoadSun's implements a simplified architecture, so that the tester does not have to worry with complicated test definitions and configurations. The designed architecture is best described in section 5.4 and a simplified version can be seen in Figure 8.

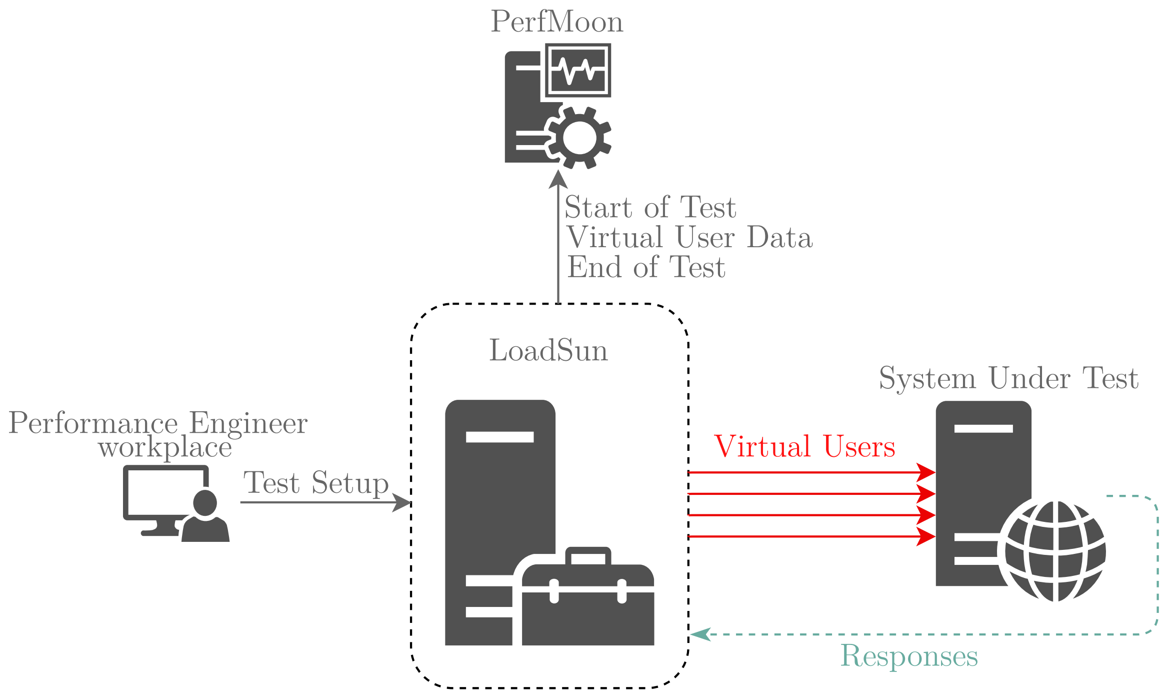
- ***DD3. Integration:*** The integration of LoadSun with the other performance modules is important so that everything works together without requiring changes from the test engineer, the modules of input generation and monitoring will be the main ones with which LoadSun will have to communicate, meaning it should be able to read the inputs in the form of workload models generated by Canopus in the ***Ecore*** format and a communication protocol with PerfMoon's monitoring module must be established so that it can exchange messages sending the response data received by each VU.

5.4 Architecture

The LoadSun's architecture was designed to function in a simplified way. It must communicate with the monitoring module of the COSMOS solution (PerfMoon), sending messages such as information about the start of the tests and the end of the tests, providing response time data, quantity of VUs currently testing the SUT, among other necessary data to perform the workload generation.

As can be seen in Figure 8 a simplified version of the architecture is presented, it shows the performance engineer creating the tests and LoadSun's VUs generating the actual load on the system. The responses coming from the SUT are processed in each VU and then the data is sent to PerfMoon so that it can decide the best way to manipulate and store it.

Figure 8 – Simplified Architecture.



Source: the author.

5.5 Chapter Summary

This chapter described the thought process behind LoadSun's design. The context where the tool is inserted was explained in Section 5.1. The requirements were presented in Section 5.2, along with two tables containing a summary of functional and non-functional requirements.

The main design decisions can be found in Section 5.3 together with their justifications. Finally, in Section 5.4 the designed architecture was presented and described.

6 LOADSUN'S IMPLEMENTATION

This chapter describes the implementation of LoadSun and the peculiarities of the Go programming language within this project.

6.1 Programming Language

Perhaps the most important part of the implementation of LoadSun[1] is the choice of programming language, the chosen one was Google's programming language called Go (or Golang). This choice was totally unbiased, since the author at the time of choice had no prior contact with the language.

Through research, it was concluded that Go had the best solution to the problem at hand, generate thousands of users concurrently with ease and a low resource consumption. What sets Go apart is the fact that it was build from the ground up to provide performance that is destined to compete with very powerful languages, such as C/C++, while supporting a relatively simple syntax that resembles dynamic languages such as JavaScript (ANDRAWOS; HELMICH, 2017). The Go runtime offers garbage collection. However, it does not rely on virtual machines to achieve that. Go programs are compiled into native machine code. When invoking the Go compiler, you simply choose the type of platform (Windows, Mac and so on) that you'd like the binary to run on when you build. The compiler will then produce a single binary that works on that platform. This makes Go capable of cross-compiling and producing native binaries.

Through *ad hoc* research it was possible to find experiments that provide evidence that Go outperforms, for instance, Java, especially when it comes to multi-threading tasks. (TOGASHI; KLYUEV, 2014)

Other factors that impacted the choice of Go as the main language were:

- Exceptionally simple and scalable multi-threaded and concurrent programming;
- Simplified C-like syntax that is as easy to read and write as Python;
- Great language for building networking services - Go has a solid team of engineers working on it (who are in the most part network engineers);
- The Go compiler compiles binaries instantly — as fast as a scripting language interpreter, which compiles on every OS without effort, a truly cross-platform compiler;
- Built-in unit testing;
- Performance is on the order of C - Go is blazing fast, but easier to write than Python, JavaScript, Ruby, or many other dynamic languages.

[1] Source code available at: <https://github.com/ProjetoDSL/LoadSun>.

Goroutines are functions or methods that run concurrently with other functions or methods. Goroutines can be thought of as "light weight threads" that run on threads (RAMANATHAN, 2017). They provide a simple way for concurrent operations — prepending a function with go will execute it concurrently. It utilizes channels for communication between Goroutines which aids to prevent races and makes synchronizing execution effortless across Goroutines.

Goroutines are extremely cheap when compared to threads. They are only a few Kb in stack size and the stack can grow and shrink according to needs of the application whereas in the case of threads the stack size has to be specified and is fixed.

The Goroutines are multiplexed to fewer number of OS threads. There might be only one thread in a program with thousands of Goroutines. If any Goroutine in that thread blocks say waiting for user input, then another OS thread is created and the remaining Goroutines are moved to the new OS thread. All these are taken care by the runtime and we as programmers are abstracted from these intricate details and are given a clean API to work with concurrency.

Now that we have talked about Goroutines, in the following is explained what their role is in the context of LoadSun.

6.2 Goroutines in LoadSun

The activity of the Goroutines focuses on the interaction with the SUT, *e.g.*, messages preparation, communication, state validation. The behaviour of a Goroutine may consist of simple HTTP request or may involve working with external files in more complicated manners. The complexity consists of data processing instructions, communication instructions and timing conditions. The set of testing related operations and their flow of execution include:

- Virtual User Creation - concerns the creation of a VU and its initialisation. At this step, the test process creates a new entry in the users repository and sets the initial status of that user;
- Virtual User Termination - at the termination of a user, the entry in the users repository has to be re-moved. Additionally, if the user utilized random or sequential reading of files, the associated indexes have to be removed as well;
- Data processing - different computations appear in the test process, e.g., data preparation, evaluation of SUT answers;
- Encoding - the data is encoded into the data format of the SUT, for our application it's only JSON marshalling. This operation only concerns the request which do not work directly on the raw-data of parameters;

- Send - the operation the load generator undertakes to send stimuli to the SUT. This operation also comprehends the underlying communication operations, e.g., transport and proxy configuration;
- Decoding - the inverse operation of encoding; a raw message, constituting the returned SUT information, is transformed into a structural entity which can be further investigated by the load generator;
- Filtering - the received messages are filtered upon rules which determine the type of the messages. In practise, the load generator has to verify more than one filter. If one of the filters matches the message, the load generator executes the actions triggered by that filter;
- Timer - is the operation to check if an event happens with a think time, a timer is set to wait for the determined think time before it does any other action;
- Logging - is the operation to produce log data when a relevant event happens in the load generator.

6.3 Chapter Summary

This chapter described the implementation of LoadSun. In Section 6.1 the choice of programming language and the line of thought behind this choice were described. Section 6.2 served the purpose of exemplifying Goroutines role in the implementation of LoadSun.

7 EXPERIMENT: LOADSUN'S BENCHMARK

This chapter presents the experimental benchmark used to evaluate LoadSun performance and validate its implementation.

The purpose of benchmarking in this context is to provide information on the performance of LoadSun's load generator and to test it against a selected competitor. Therefore, the test system defines the input stimulus to the system under test by providing precise definitions of scenarios and scripts. The tests specify how the traffic is to be provided to the SUT and define the measurements to be made from the SUT and how they are to be reported.

7.1 Benchmark Definition

The goal of this case study, is to create a benchmark for LoadSun's comparison against the selected competitor which implies the realisation of a representative workload and a test procedure capable of producing comparable results. Therefore, some definitions related to benchmarking are presented first.

A benchmark is a program, an application or a method used to assess the performance of a target system (hardware and/or software) (JOHN; EECKHOUT, 2018). This is realised by simulating a particular type of workload on a component or system. In a benchmark the test scenarios test the SUT services, the utility performance, one can experiment with different methods of loading data, transaction rate characteristics as more users are added, and even test the regression when upgrading some parts of the system. A benchmark is a type of performance test designed such that it can be used as a method of comparing the performance of various subsystems across different chip/system architectures (VOKOLOS; WEYUKER, 1998; DIN; PETRE; SCHIEFERDECKER, 2007). From a business perspective, a benchmark is the ideal tool to compare systems and make adequate acquisition decisions (MENASCÉ, 2002). Nevertheless, a benchmark serves also as a performance engineering tool in order to identify performance issues among versions of a system.

A benchmark is not easy to realise and often the tools have to deal with high performance capabilities. The list of requirements includes:

- Benchmark procedure - the method for defining the load and for measuring the performance, should be based on the number of virtual users. This can be measured either by the number of users a specific configuration can support, or the minimal system cost of a configuration that can support a specific number of users;
- Hardware resource limitations - the benchmark must measure the capacity of a system. This requires the test system to emulate the behaviour of a big number of users interacting with the SUT. Although the benchmark will define a traffic load corresponding to hundreds of simulated users, the method of traffic generation

Table 9 – Scenario.

Use-Case	HTTP Request	Think time (s)	Method	Path
1	Login	10	POST	/public/login
	Search Request	5	GET	/public/products/search
	Product Detail	15	GET	/public/product/
	Order Product	5	POST	/public/product/order
	Cart	5	GET	/public/cart
	Check out	15	POST	/public/checkout
2	Home Page	5	GET	/public/home
	New Products	10	GET	/public/products/new
	Best Sellers	10	GET	/public/products/bestsellers
	Product Detail	15	GET	/public/product/
	Search Request	10	GET	/public/products/search

should be defined in such a way that it can be implemented economically by both test systems;

- Realistic workload - the benchmark must be driven by a set of traffic scenarios and traffic time profiles, *i.e.*, the benchmark is executed for various load levels. The test system should generate realistic traffic and cover the majority of use cases of interest.

7.2 Use-Cases, Scenario and Scripts

The benchmark traffic has been defined by selecting scenarios from the most common use cases that are encountered the most in the real life deployments. For each Use-Case, its design objectives have been identified. Table 9 provides a summary of the use-cases selected for this case study.

7.2.1 Use-Case 1: Search and Checkout

This use-case regards the user login in his account, searching for a specific product, going through the details of that product to check if that's the product he wants, ordering the product, going to his cart and finally buying the product. To make it more realistic, it is important that the user takes the longest time during the steps of reading the product details and purchasing it, as he must inform the number of his credit card to finalize the purchase. The purpose of this use case is to be the most costly for both the SUT and the testbed environment, as it performs more system functions in addition to performing more POST requests with parametrizations.

7.2.2 Use-Case 2: Browsing

This use-case represents the most common user, as it encompasses just browsing through the website looking for products. The user first goes to the Home Page, then goes through the New Products page, switches to the Best Sellers, finds a product and goes through its details in the Product Detail page and finally he decides to search for a specific product in the requesting the Search Request page. This use-case will be executed the most, as it is more common for users to just browse the website pages than make an actual purchase.

7.2.3 Use-Case Representativeness

The benchmark traffic set consists of a scenario that belongs to the most common use cases that are encountered the most in the real life deployments. The representativeness of the selected scenarios has a considerable impact on the significance of the produced results. One simple rule is that the more complex the scenarios are, the more significant the results are. The representativeness of the chosen use cases is evaluated considering the following arguments:

- HTTP protocol coverage - Overall, any type of Request and Response defined by the HTTP protocol should be included in at least one use-case. Although we only used the GET, and POST methods, through our testing we concluded that POST produced the same footprint as the other most common methods like PUT and DELETE. This ensured a good coverage of the HTTP protocol with respect to test method types;

- Load generator coverage - The coverage of the functionalities of the Load generators itself are very important factors as well. Every feature was guaranteed to be present in at least one of the use cases to ensure complete coverage. This includes features such as parametrization through external files, fixed parametrization, random parametrization, user ramp up “ladder”, etc. As Apache JMeter does not support most of these features out of the box, some pluggins where installed to ensure total comparability;

- SUT application coverage - The SUT supports various types of features: product searching, logging in, checking product details, ordering products, checking the user shopping cart, etc. These types of interactions are recognised in many other applications. Therefore, the selected use cases ensure also a good coverage with respect to the interactions types between users and Web applications;

- Negative testing - Not only the positive interactions are treated, *e.g.*, successful logins, but also negative situations, *e.g.*, the nonexistence of a searched product. This ensures a good coverage of typical user behaviours as encountered in reality.

7.3 Tools for Load Generation

The tool selected to be compared to LoadSun in this benchmark was Apache JMeter version 5.2. This choice was based on the systematic mapping study (section 4.3) previously performed by this study. The following requirements were used to define this choice.

- Open source - JMeter is an open source software. This means that it can be downloaded free of cost;
- Cross platform - JMeter runs everywhere where Java runs, including, but not limited to Windows, MacOSX, Linux, FreeBSD, Solaris, AIX and HP-UX;
- Most popular - From our systematic mapping, it was concluded that JMeter is one of the most popular choices when it comes to performance testing, losing only to the paid solution, LoadRunner;
- Comparability - Being able to compare both tools equally was a major concern when selecting the tool. Although JMeter doesn't support all of LoadSun's features out of the box, it has plugins available for such features that allowed the tools to be compared equally.

7.4 Experiments

The experiments presented in this section serve two major goals:

(i) The first goal is to show a concrete execution of the benchmark with a complete analysis of the performance of the SUT, to ensure that both tools, with the same configuration, generate similar load on the SUT. Additionally, the benchmark is applied to the load generators resource consumption, to check which of the tools can generate the same load levels requiring less of the testbed resources;

(ii) The second goal is to check which tool can make the most amount of requests in the same time frame, and also check the resource utilization of both.

7.4.1 The Testbed Environment

The hardware used to run the Benchmark test system consists of a robust server providing enough processing power to produce the requested loads. The system is a rented server in the Google Cloud Platform and the current configuration can be seen in Table 10.

Table 10 – Testbed Hardware Configuration.

CPU	product	Intel(R) Xeon(R) CPU 1 Core @ 2.30GHz
	width	64 bits
Memory	size	3840MiB
PCI	product	Intel 440FX - 82441FX PMC [Natoma]
	width	32 bits
	clock	33MHz
Disk	product	Google Persistent Disk
	size	10GiB (10GB)
Connection	provider	Google Cloud
	download	661.96 Mbit/s
	upload	332.00 Mbit/s
OS	version	Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1047-gcp x86_64)

7.4.2 The SUT Environment

The SUT consists of software and hardware. While the hardware is the same, the software here has various parameters that can be tuned. The hardware configuration is described in Table 11, it is very similar to that of the Testbed Enviroment, as it was also a rented server in the Google Cloud Platform, the only differences was in the CPU which had 1 extra core so it could better handle the projected workload.

The implementation used as the SUT software within the case study is a proprietary implementation of a e-commerce system in PHP utilizing the Laravel Framework[1]. This application, was developed by the author of this study and Girardon (2019) of the LESSE research group that was in charge of implementing PerfMoon (COSMOS monitoring module). It uses the Apache server version 2.4.18 (Ubuntu) mod_fastcgi/mod_fastcgi-SNAP-0910052141, with the following modules: *core_module, watchdog_module, http_module, log_config_module, logio_module, fastcgi_module.*

7.4.3 Monitoring Software

During the test execution, all benchmark-related information needed for assessing the SUT and testbed system performance is collected and stored by Google's Stackdriver[2]. Stackdriver has total integration with the Google Cloud Platform, hence the main reason it was selected as the monitoring software of this case study. It monitors the system performance thought an agent that is installed in Google's Virtual Machines and generates many types of user configurable graphs. The visualisation tool generates a benchmark report for the evaluation of SUT's behaviour under well-known load conditions. This benchmark report contains various graphs to visualise collected metrics in different views: CPU usage, Memory usage, Disk usage, etc. Also various statistics (max, min, average,

[1] Available at: <https://laravel.com/>

[2] Available at: <https://cloud.google.com/stackdriver>

Table 11 – SUT Hardware Configuration.

CPU	product	Intel(R) Xeon(R) CPU 2 Core @ 2.30GHz
	width	64 bits
Memory	size	3840MiB
PCI	product	Intel 440FX - 82441FX PMC [Natoma]
	width	32 bits
	clock	33MHz
Disk	product	Google PersistentDisk
	size	10GiB (10GB)
Connection	provider	Google Cloud
	download	1274.13 Mbit/s
	upload	987.37 Mbit/s
OS	version	Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1047-gcp x86_64)

variation, etc.) are reported. All graphs related with system resources in the following sections are generated with the Stackdriver tool.

All metrics related to the load generation, such as: VUsers count, response time, requests per second, etc. were collected by the load generators themselves, the graphs related to those metrics in the following sections were generated in Google Sheets.

It is important to note that among each test run both machines were completely reset, and the necessary settings redone to ensure clean test runs without cache or other interferences.

7.5 Experiment 1: A Complete Benchmark Execution

The benchmark may serve as a comparison tool between different hardware and software configurations. The comparisons can be realised in different ways:

- **A** - comparison between different software configuration with same hardware. This approach is useful for the software developers as a debugging method. Along the project one may track the performance improvements. Additionally, the approach is useful also for comparing different tuning of the load generator software on a given platform;

- **B** - comparison between different hardware configurations but same software configuration. This approach can be used by hardware vendors in order to compare their hardware on a common workload basis;

- **C** - comparison between two different hardware configurations and different software configuration. This case reflects the situation when the software is tuned for a particular hardware configuration in order to take advantage of the underlying hardware resources.

For this experiment we will focus on the approach A, comparing different software configurations, specifically the impact on the testbed hardware using LoadSun and Apache JMeter. For this we will set Stackdriver to log the most important metrics in the testbed server, those being: CPU usage percentage, Memory usage percentage, Throughput (Received and Sent bytes) and Disk I/O.

To ensure that the load generated by both tools is equivalent and can be used as a parameter to compare resource consumption in the SUT, it is expected that both tools generate the same load levels when using the same traffic set configuration, and the testbed hardware is not a limiting factor.

7.5.1 SUT Sustained Load Comparison

This section explains the way how the benchmark is executed against the SUT. It also discusses how to interpret the various statistics and, the most important, how to determine if both loads sustained by the SUT are equivalent.

The purpose of this experiment is to compare the load generated in the SUT by both tools using the exact same traffic set configuration, if both tools are working correctly they should produce very similar results in both CPU and Memory usage.

The selected traffic set, the ratio of each Use-Case (UC) and the ramp up configuration are presented in Table 12 and Table 13. Use-Case 1 should make up 40% (360 VUs) of the load, while Use-Case 2 should amount to 60% (540 VUs).

Table 12 – Traffic set distribution.

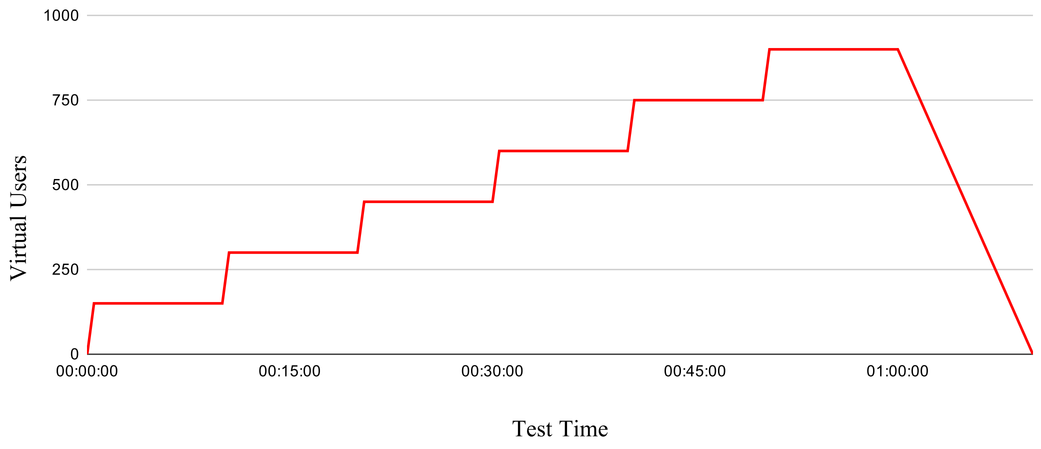
Use-Case	Total VUs	VUs	Distribution	Test Duration
1	900	360	40%	1h10min
2		540	60%	

Table 13 – Traffic set ramp up configuration.

VUs UC1	VUs UC2	Total VUs	Test Time	
60	90	150	0 min.	Test Start Ramp Up
120	180	300	10 min.	Ramp Up
180	270	450	20 min.	Ramp Up
240	360	600	30 min.	Ramp Up
300	450	750	40 min.	Ramp Up
360	540	900	50 min.	Ramp Up
360	540	900	60 min.	Ramp Down
0	0	0	70 min.	Test End

The test will run for 1 hour and 10 minutes, in this period it will start with 150 VUs and maintain this load for 10 minutes, then it will add another 150 VUs and so on, until 900 VUs have been kept for 10 minutes then it will start ramping down for another 10 minutes until it reaches 0 VUs and the test ends. For a better visualization of the expected VUs at each stage of the benchmark refer to Figure 9.

Figure 9 – Expected VUser Ramp Up.



The results of the experiment can be observed in Figure 10 for the SUT CPU usage and in Figure 11 for the comparison of the SUT Memory utilization. In the CPU utilization we can clearly see each of the ramp up steps from both tools. The CPU reported a 7-8 percent increase in CPU utilization for each 150 VUs we added until it started dropping during the ramp down stage.

Both tools resulted in very similar results in the SUT, confirming the experiment expectations.

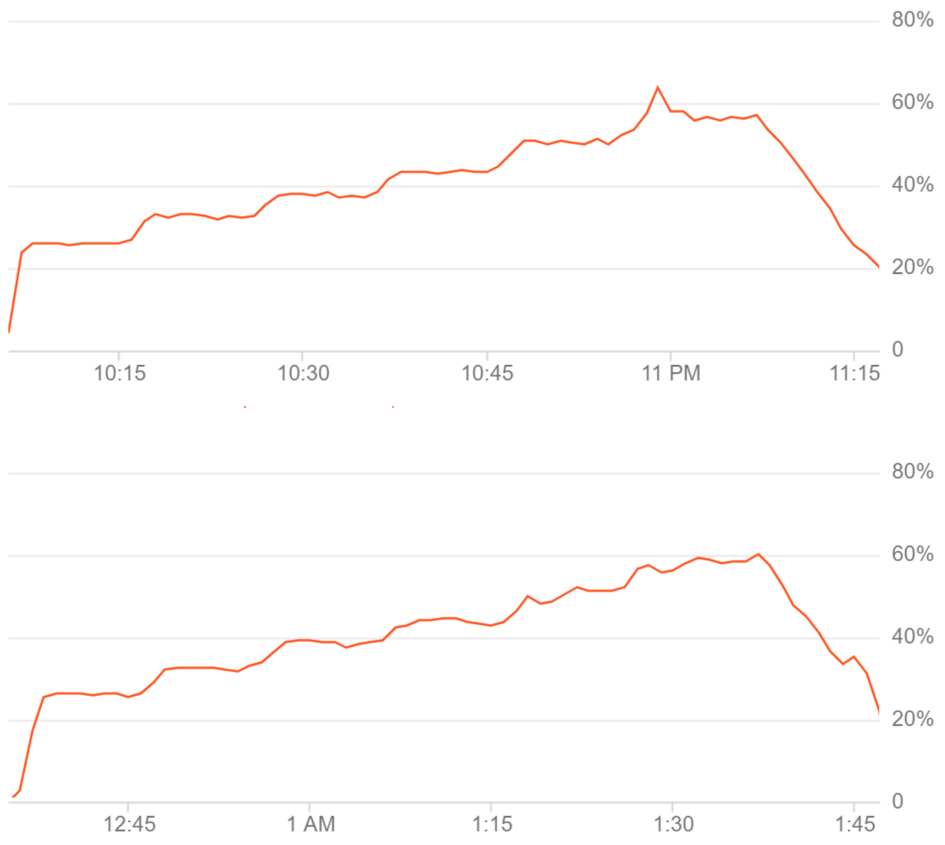
As for memory utilization percentage, both tools generated a very low impact on it. This might be the case because the SUT had 4GB of memory, and this might be too high of a number for a Web application server. The memory usage saw a .5 percent (0.5%) almost linear increase in utilization in both test runs. LoadSun actually generated a .5 percent (0.5%) higher load than JMeter, but these numbers are very low and should not be taken into account.

The most important conclusion to draw from this section is that with the same VU configuration, and no testbed bottlenecks, the load generated in the SUT was the same. It was also possible to conclude by observing CPU usage the impact of each ramp-up period, confirming that both load generators are working correctly.

7.5.2 Testbed Hardware Utilization Comparison

For this experiment, more important than the load generated in the SUT, are the consumptions of each resource in the testbed when using LoadSun or another workload generation tool, in this case Apache JMeter. This is important since a tool that uses less resources can theoretically generate more load with the same hardware.

Figure 10 – CPU usage of the SUT with LoadSun (above) and JMeter (bellow).



Since we want to identify which tool is “lighter”, we will directly compare four hardware utilization metrics. These are: CPU Usage Percentage, Memory Usage Percentage, Disk I/O, and Network Throughput.

7.5.2.1 CPU Consumption Comparison

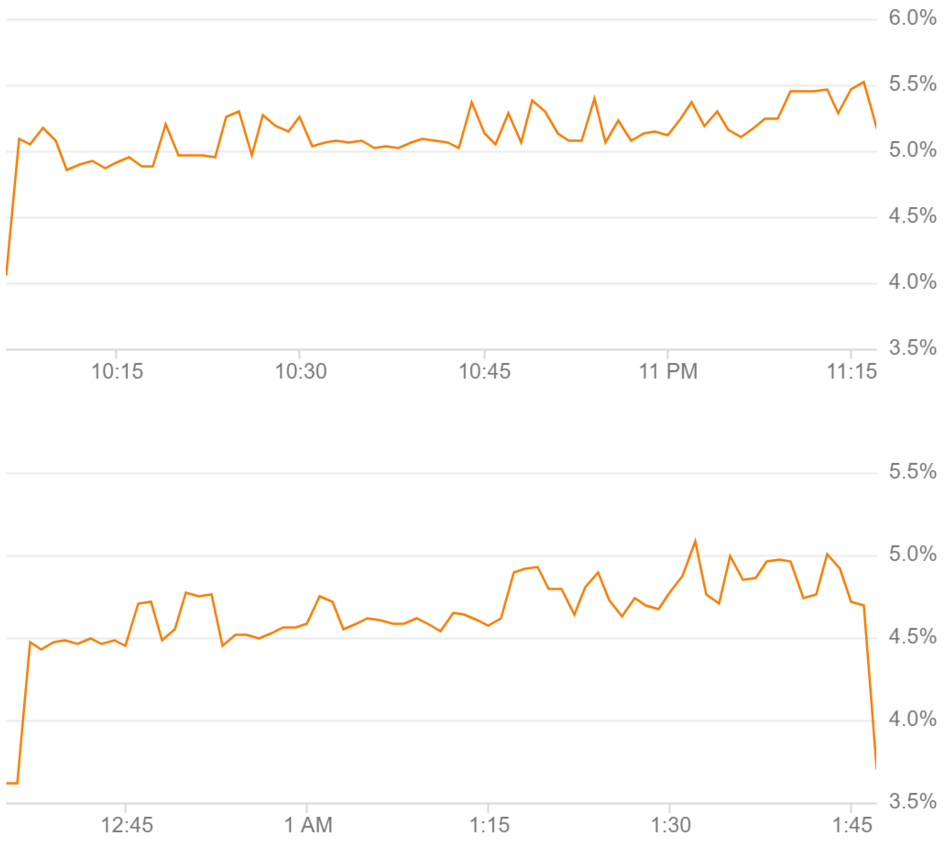
Firstly we want to compare the CPU usage on the testbed utilized by both tools, this is to compare which tool consumes less CPU resources to do the same type of test, generating the same load on the SUT.

Figure 12 shows the two CPU usage graphs in the testbed during the execution of the tests, in the top graph we can see the CPU usage while using the solution presented in this term paper, LoadSun, while in the bottom graph we can observe the consumption while using Apache JMeter.

During the execution of LoadSun it was possible to notice and compare different CPU usage increases as the number of virtual users increased in each ramp up. On average CPU processing went up 1.5% for each ramp up of 150 VUs. Showing only one abnormal 1.5% increase peak during the second ramp up. As this was a very low peak it was concluded that the metrics observed here indicated a stable and constant application.

While monitoring Apache JMeter it was possible to observe a more unstable be-

Figure 11 – Memory usage of the SUT with LoadSun (above) and JMeter (bellow).



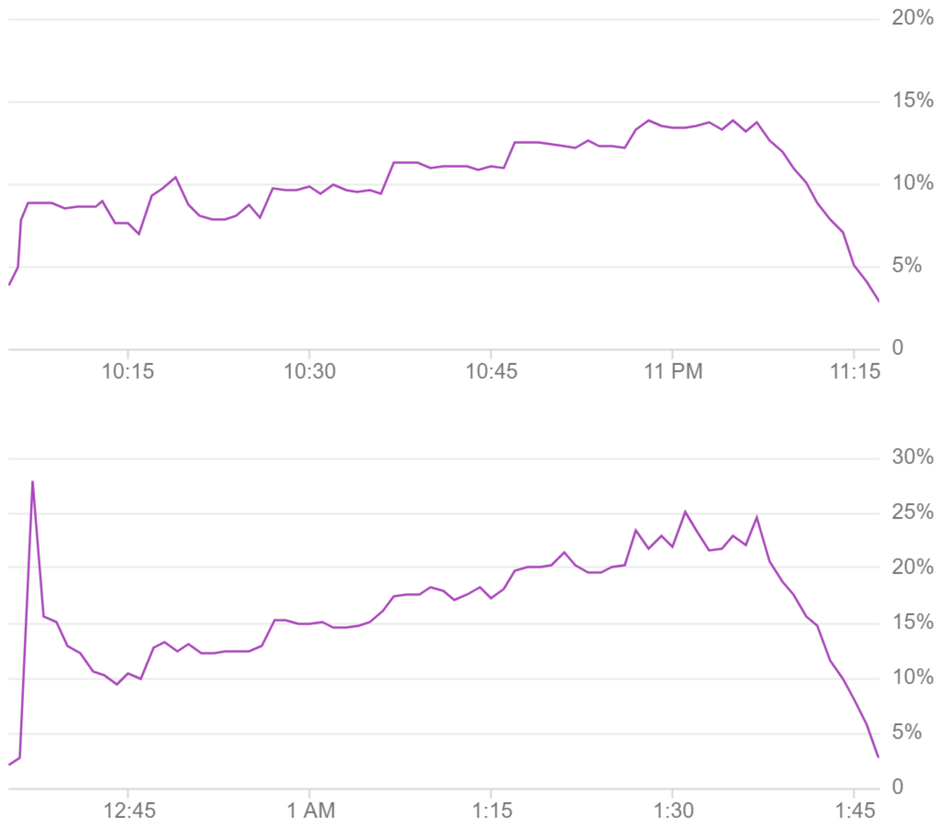
havior when compared to LoadSun, at the beginning of the tests, during the first ramp up, the CPU usage showed peaks of up to 17% more usage, stabilizing soon after. But after the last ramp up and stabilization with 900 VUs the CPU showed several usage peaks, these significantly lower, up to 4%. During each 150 VU ramp up, CPU usage using JMeter rose by about 3-5%, more than double the LoadSun's observed consumption figures.

Comparing both graphs, it was concluded that in CPU consumption the largest difference, excluding the initial peak, was 12% lower than JMeter, this was observed while the systems performed the highest load defined for the test at 900 VUs. While the smallest difference between the consumption of the two tools was during the execution of the smallest number of VUs of the scenario (150 consecutive VUs), with JMeter showing a 3% higher consumption.

7.5.2.2 Memory Consumption Comparison

Another important aspect of resource consumption is memory usage, it is of great value to know the estimated memory consumption for the load to be generated. Soon we will monitor and compare the memory consumption required to run both tools individually using the same traffic set already defined in this case study.

Figure 12 – CPU usage of the Testbed with LoadSun (above) and JMeter (bellow).



The memory usage graphs of LoadSun shown in the top of Figure 13 show an increase of 18% from the beginning of the tests until the end of testing, the highest peak consumption was 33.2% during the execution of the highest number of VUs.

Apache JMeter's memory consumption can be observed in the bottom graph of Figure 13. The total increase in memory usage was 28% from 150 to 900 consecutive VUs. This increase is around 10% higher than LoadSun's consumption. Excluding the number differences, the two graphs show a very similar curve, indicating that both behave similarly when it comes to memory usage.

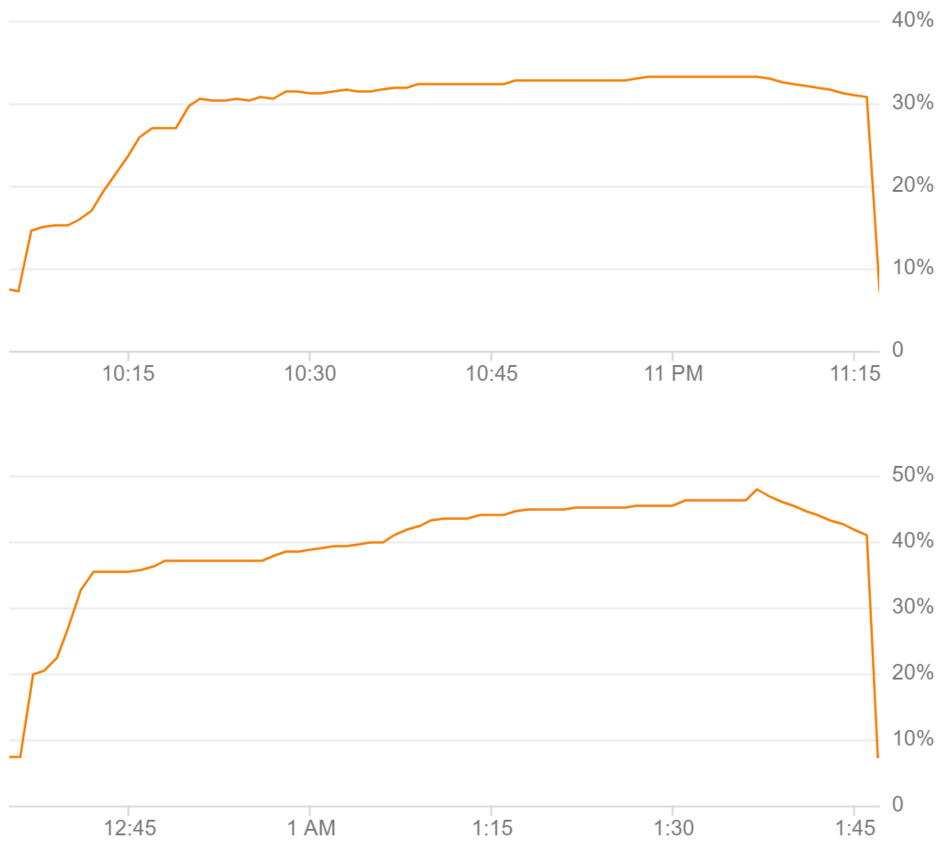
7.5.2.3 Throughput Comparison

It is interesting to compare throughput data as unnecessary high throughput can result in testing limitations due to possible limitations of the testbed network. The throghput graphs of the machines are distributed in Figure 14, LoadSun's graph is arranged higher than the graph while the graph below refers to Apache JMeter.

Both tools produced almost identical results, as they were producing the same amount of requests utilizing the same test scenario. Both reached a peak of 96KiB/s of sent bytes during the greatest number of simultaneous VUs in the tests, and during the times of the lowest load the number of sent bytes was 18KiB/s.

The number of received bytes was also the same, as it is to be expected as they were

Figure 13 – Memory usage of the Testbed with LoadSun (above) and JMeter (bellow).



making requests from the same application. The peak of received bytes was 87KiB/s and the lowest was 15KiB/s, resulting in a maximum Throughput of 183KiB/s and a minimun of 33KiB/s.

7.5.2.4 Disk I/O Comparison

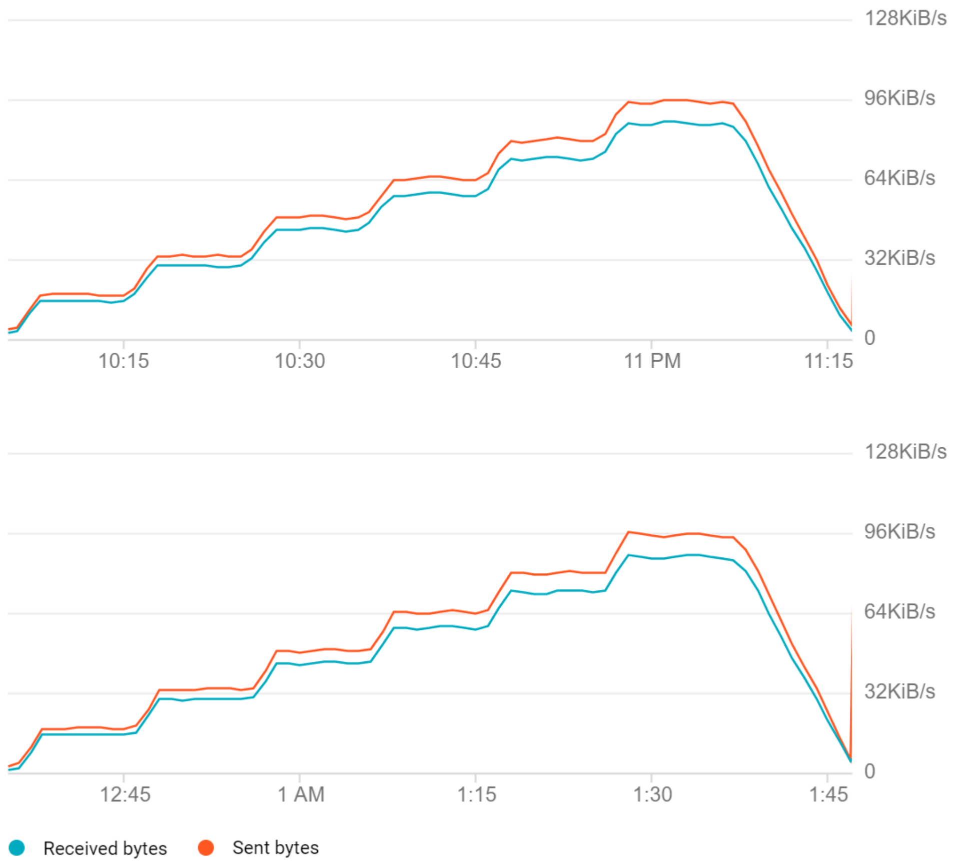
Since it was possible to replicate the test scenario identically in both tools, with parametrization through external files, random reading of these files, etc. Disk usage, in theory, should also be similar as long as both tools generate logs in a similar way, e.g. 1 log for each request.

Disk usage graphs are available from Figure 15, again with the graphs for this term paper solution at the top and the Apacha JMeter graphs below.

As expected the disk usage numbers are very similar, the only notable differences is that LoadSun peaked before Apache JMeter, but Apache JMeter showed larger and more unstable peaks during testing.

The maximum usage excluding peaks with LoadSun was 65KiB/s while the largest figures of Apache JMeter were 70KiB/s, showing slightly higher usage than LoadSun. As for the lowest numbers, they were 16KiB/s for both tools during the low load periods.

Figure 14 – Throughput of the Testbed with LoadSun (above) and JMeter (bellow).



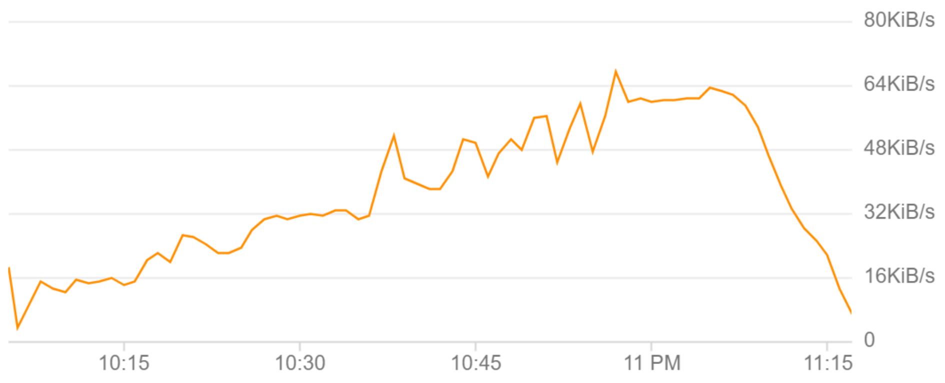
7.6 Experiment 2: VUsers number Comparison

An interesting factor to discover is the ability of both tools to generate load, to measure this factor we monitor the rate of Responses Per Second (RPS) that each tool can generate in a given time.

The objective is to find the limit of both tools with the available hardware and find out the maximum possible number of RPS with both, so keep in mind that we only selected the first use case for the tests, as this kept the most factor. HTTP protocol coverage was high, but the think time between each request was completely removed, with the intent that the tools would execute as many requests as their internal processing allowed. It also removed ramp up and ramp down, and the use of external file parameters so that reading files did not limit the amount of RPS.

The tools were ran for 120s with varying concurrency levels, as listed in Table 14. When doing performance testing, running a test for 120 seconds may not be enough to get stable and statistically significant results. However, when you have a simple, controlled environment where you know pretty well what is going on (and where there is, literally, nothing going on apart from your tests) you can get quite stable results despite running very short tests.

Figure 15 – Disk I/O of the Testbed with LoadSun (above) and JMeter (bellow).



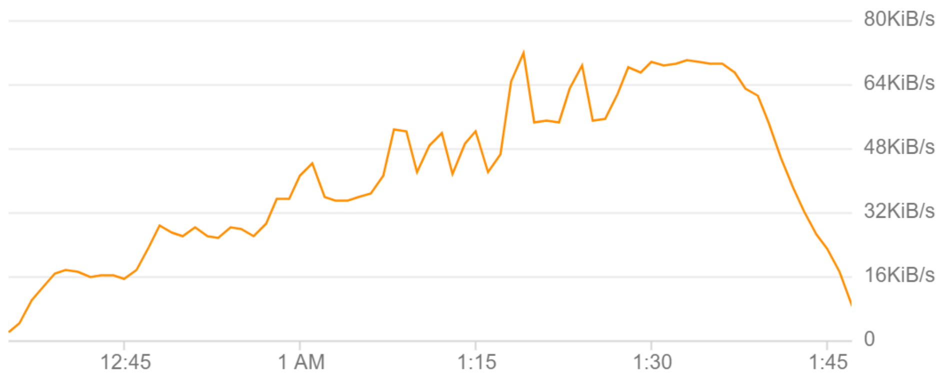


Table 14 – Experiment 3 Details.

Tool	VUs	Duration	Network RTT	Max Theoretical Request Rate
LoadSun	20	120s	10ms	2,000 RPS
	50			5,000 RPS
	100			10,000 RPS
Apache JMeter	20			2,000 RPS
	50			5,000 RPS
	100			10,000 RPS

The tests were repeated multiple times and seen only very, very small variations in the results, so we are confident that the results are valid for our particular server environment. You are of course welcome to run your own tests in your environment, and compare the results with those we got.

The most important parameter is how much concurrency the tool should simulate. Several factors mean that concurrent execution is vital to achieving the highest RPS numbers. Network (and server) delay means you can not get an infinite amount of requests per second out of a single connection because of Round Trip Time (RTT). So, it does not matter if the machines both have enough CPU and network bandwidth between them to do 1 million RPS, the max RPS you will ever see will be:

$$\frac{VUs}{RTT(seconds)} = Maxtheoreticalrate$$

A concurrency level of 20, however (20 VUs) means that our test should be able to perform 20 times as many RPS as it would when using only a single connection.

In our case, the actual network delay was about 2ms between source and target hosts in our server setup, plus the server response delay which amounted to around 10ms RTT, which means our theoretical max RPS rate per connection would be somewhere around $1/0.01 = 100$ RPS. If we use 20 concurrent connections in this case the server could support maybe 2,000 RPS in total.

Note that the parameters of these tests are likely not very realistic if you want to simulate end user traffic - there are few situations where end users will experience 2ms network delay to the back-end systems, and using only 20-100 concurrent connections/VUs is also usually too low to be a realistic high-traffic scenario. But the parameters may be appropriate for load testing something like a micro-services component.

As the most interesting metric is the max RPS number, we do not want the network bandwidth to be the limiting factor. Although the servers provided us with very fast connection speeds it could soon be a limiting factor when the tests were performing 8,000 RPS. Our SUT is very good for this as it had no user interface and the responses were very small, even smaller than the actual requests.

7.6.1 Results

The results of our experiment can be seen in Figure 16, they have shown that at low load levels the RPS rates have come very close to the max theoretical number. This is were we see the smallest difference between both tools.

The max theoretical rate at 20VUs was 2,000 RPS, LoadSun came very close to that, reaching 1879 RPS, followed by Apache JMeter that reached a total of 1,563 RPS.

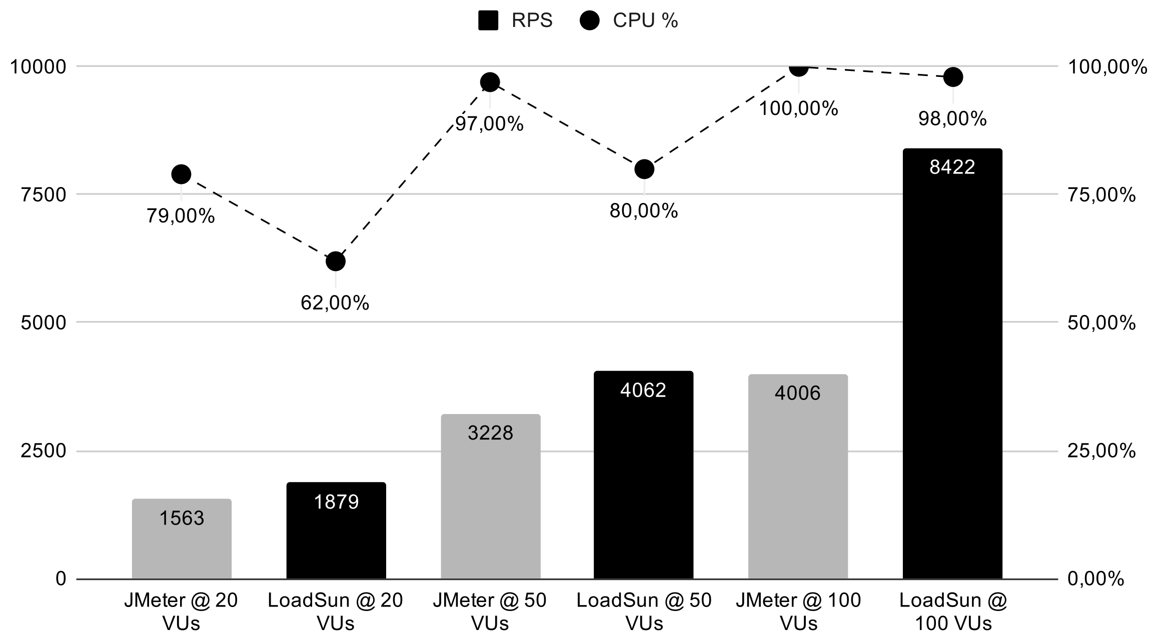
When the tests were run with 50 VUs the numbers started to diverge further from the max theoretical rate, which was 5,000 RPS. The figures we got were 3,228 RPS for Apache JMeter and 4,062 RPS in LoadSun, almost a thousand less than the max theoretical rate for LoadSun and 1,772 RPS less in Apache JMeter. This is due to the internal processing of the load generators to produce each request, which is affected by their own implementation, programming language and many other factors.

Here we start to see some bottleneck by the testbed CPU with Apache JMeter, in the 50 VU test it reached as high as 97% of utilization. While LoadSun maximum measurement was 80% CPU utilization.

More VUs generally means higher RPS rates, up to a point where factors like CPU bottlenecking start limiting the concurrent processing the generators can accomplish. We can also see the impact of CPU bottleneck in Apache JMeter in the following test of 100VUs, where it got 4,006 RPS. LoadSun did not suffered from this, as it got 8,422 RPS in the 100VUs run, reaching 98% CPU utilization.

Memory utilization was around 10% to 14% higher in Apache JMeter as well,

Figure 16 – Requests Per Second Rates.



however memory bottlenecks were not found with our hardware configuration.

7.7 Threats to Validity

In this section, different validity threats related to the benchmark are discussed. The author used Creswell e Creswell (2017) to explain different validity threats in the research.

7.7.1 Internal Validity

Internal validity focus on how sure we can be that the treatment actually caused the outcome. There can be other reasons that have caused the result on which we do not have control over or have not measure (FELDT; MAGAZINIUS, 2010).

The internal validity threats in this research are:

- The author had only used the selected tool (Apache JMeter) once before. To overcome this threat, the author learned how to conduct the performance testing by taking help from the online tutorials. After the author learned how to properly use the tool, the experiment was conducted.

- There was a threat that the monitored metrics in the experiment can really explain the outcome the author wants to research. To overcome this threat, some pilot tests were conducted before the execution of the real experiment benchmark. The researcher used both tools and tested the scenarios to validate the results of both.

7.7.2 External Validity

External validity is related with whether the results can be generalized outside the scope of the study (FELDT; MAGAZINIUS, 2010).

- There was the threat of network infrastructure with unstable Internet speed. This experiment was done in a rented server of Google Cloud Platform where the Internet speed was stable and high enough that it would not limit the testing capabilities of any of the tools.

7.7.3 Construct Validity

Construct validity motivation is on the relation between the theory behind the experiment and the interpretations. The interpreted result might not correspond to the effect what is being measured (FELDT; MAGAZINIUS, 2010).

- There was a threat that the selected tool can answer the selected parameters for the experiment. To overcome this threat, different literature, through the systematic mapping previously discussed in this term paper, and the official websites of the selected tools were studied and confirmed that the tool can satisfy the selected parameters.

7.7.4 Conclusion Validity

Conclusion validity concentrate on how sure the treatment used in an experiment really is related to the actual result obtained (FELDT; MAGAZINIUS, 2010).

- Conclusion validity is a threat that can lead the research to an incorrect conclusion. To overcome this threat, the author acquired a background in performance testing research and was assisted by his supervisor and co-supervisor, both researchers in the field of software engineering and performance testing.
- The author used a Web application for testing. There was a threat that if the Web application is down from the hosting side. To overcome this threat, the experiment was conducted in a proprietary solution, which was hosted in a server the researcher had total control of.

7.8 Chapter Sumary

This chapter served the purpose of evaluating LoadSun against the industry's leading open-source workload generation tool, Apache JMeter. For this purpose, Section 7.2 described the use-cases, scenarios, and scripts used in the benchmark. The tools used in the experiment were described in Section 7.3, and the other experiment settings in Section 7.4.

In Section 7.5 and Section 7.6 the experiments performed and the results obtained are described. The identified trial validity threats are available from Section 7.7.

In the next chapter the conclusions and possible future works from this term paper are presented.

8 CONCLUSIONS AND FUTURE WORK

To ensure the quality of large scale systems, performance testing is required in addition to conventional functional testing procedures. Furthermore, performance testing is becoming more important, as an increasing number of services are being offered in the cloud to millions of users.

It is well known that the feasibility of performance testing depends on automation tools, particularly tools that generate synthetic workloads in the SUTs in order to identify defects, boundaries and bottlenecks.

Through the SMS executed in this term paper, it was possible to identify that despite the demand, the available options do not include high quality tools, that are open-source, lightweight and can be easily used by students with little experience when the focus is the teaching-learning process.

To mitigate this problem, in this term paper project, a tool was developed that allows generating workloads in Web-based applications, with the purpose of being lightweight and open source, to be used by undergraduate students at UNIPAMPA, and possibly, performance testing engineers in the industry.

In the present state of our research, it can be stated that it's main objective (see Section 1.2) was reached. The proposed solution was implemented, tested, and evaluated against another workload generator. The two software testing tools compared in this research on the basis of different parameters are LoadSun and Apache JMeter. Both of them are open source software. LoadSun is a 100% Go application while Apache JMeter and is built in java. Apache JMeter is the open source industry standard for application performance testing. Although both tools are very good for performance testing and very simple to install and run, through the experiments conducted in this term paper it was possible to conclude that LoadSun has an edge over Apache JMeter when it comes to raw performance.

The results of the implemented evaluation indicate that LoadSun is capable of generating the same load level when the hardware is not a limiting factor in a test run, and in the cases where the hardware does limit the workload generator LoadSun outperformed Apache JMeter by a great margin.

For future work, LoadSun needs a parser to be able to accept inputs from the Canopus Domain Specific Language (DSL) and be fully integrated in the COSMOS Performance Testing Solution. Allowing the generation and execution of performance tests from a DSL (a solution that is not available today) with the option of generating tests both textually and graphically.

The description of all the tasks performed in this term paper is presented in Section 8.3 in the form of a schedule. The main lessons learned are described in the following section.

8.1 Lessons Learned

During the preparation of this term paper project the main lessons learned are listed below:

- The planning and formulation of well-known systematic mapping protocols should not be underestimated, as these require time and experience, as well as several iterative rounds of revisions and improvements. The best approach is to start as soon as possible, so that, at the end of the mapping, it presents valid and relevant results to contribute to the body of knowledge. Also as a good practice, it is recommended to use tools that support this process, which can prove to be too laborious. In the mapping performed by this study the Thoth tool of LESSE's research group was used, which assisted from the planning to the final phases of data extraction and reporting;

- Taxonomies can serve as frameworks for analyzing and classifying things, as well as assisting in the definition and use of common terms in related research areas. However, the popularization and availability of the taxonomy for the stakeholders is essential;

- Performance tests, while essential, must be run correctly so that the results match the reality. The use of performance testing tools on the same device where the SUT is hosted is a bad practice and should be avoided, running this type of tool generates load on the system and interferes with test results, making them practically invalid and unusable;

- The most important aspect in scientific research must be the emphasis, the concern in the application of the scientific method rather than the emphasis on the results obtained. Bad results may serve as example for future research and development.

8.2 Publications

It is worth mentioning successful publications, attempted publications, and planning for future publications derived from this term paper. The events described below were sorted in chronological order.

- **ESEM 2019** - The ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) is the premier conference for presenting research results related to empirical software engineering. (Attempted publication.)

- **SBES 2019** - The XXXIII Brazilian Symposium on Software Engineering (SBES), annually promoted by the Brazilian Computer Society (SBC), is the premier Software Engineering event in Latin America. SBES is held in conjunction with CBSoft - Brazilian Conference on Software: Theory and Practice. (Attempted publication.)

- **IPCCC 2019** - The International Performance, Computing, and Communications Conference is a premier IEEE conference presenting research in the performance of computer and communication systems. For over three-and-a-half decades, IPCCC has been a research forum for academic, industrial, and government researchers. (Paper was accepted, but unfortunately it was not possible to gather resources to travel to London.)

- **ERES 2019** - The Regional School of Software Engineering (ERES) is an event promoted annually by the Brazilian Computer Society (SBC). The third edition of the event, ERES 2019, has taken place in Rio do Sul (SC), the Alto Vale do Itajaí region, from October 7 to 9, 2019, and was jointly organized by the Federal Institute of Santa Catarina (IFC) and Santa Catarina State University (UDESC). (Paper accepted and presented.)

- **SIEPE 2019** - The 11th International Teaching, Research and Extension Salon (SIEPE) was held in Santana do Livramento (Brazil) and Rivera (Uruguay) on October 22, 23 and 24, 2019. (Paper accepted and presented in oral and poster modalities.)

- **SAC 2020** - The 35th ACM/SIGAPP Symposium On Applied Computing (SAC). In the Software Verification and Testing (SVT) Track. (Two papers Accepted.)

- **STVR** - Software Testing, Verification and Reliability (STVR) is an international journal, publishing 8 issues per year. It publishes papers on theoretical and practical issues of software testing, verification and reliability. (Planned for publication.)

8.3 Schedule

This term paper activities were planned and executed as it's described in the schedule in Table 15.

The planning and definition of the SMS protocol executed by this study began in September 2018, while the search process of running the search strings in the bases was carried out in late December of the same year, with the application of the inclusion and exclusion criteria, quality assessment, data extraction and data analysis extending until the end of April 2019.

The analysis and design of the proposed tool occurred during the period from May to June, concurrently with the writing of the term paper project.

LoadSun's development has started in late June followed by the planing and execution of the tools evaluation in November, winding up in the writing and presentation of the final term paper in the same month.

Although this report ends here, LoadSun's future evolution is already planned to be executed whitin LESSE's research group in the years to come.

Table 15 – Schedule.

	2018/2				2019/1							2019/2			
Task	**Set**	**Oct**	**Nov**	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**
Planning the Systematic Mapping	▨	▨	▨	▨	-	-	-	-	-	-	-	-	-	-	-
Executing the Systematic Mapping	-	-	-	▨	▨	▨	▨	▨	-	-	-	-	-	-	-
Writing Term Paper Project	-	-	-	-	-	-	-	▨	▨	▨	-	-	-	-	-
Analysis & Design	-	-	-	-	-	-	-	-	▨	▨	-	-	-	-	-
Presenting Term Paper Project	-	-	-	-	-	-	-	-	-	▨	-	-	-	-	-
LoadSun's Development	-	-	-	-	-	-	-	-	-	▨	▨	▨	-	-	-
Plan Tool Evaluation	-	-	-	-	-	-	-	-	-	-	-	▨	▨	▨	-
Writing Term Paper	-	-	-	-	-	-	-	-	-	-	-	-	-	▨	▨
Tool Evaluation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	▨
Presenting Term Paper	-	-	-	-	-	-	-	-	-	-	-	-	-	-	▨
Refining Term Paper for Homologation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	▨

BIBLIOGRAPHY

ABBORS, F. et al. Model-based performance testing in the cloud using the MBPeT tool. In: . [S.l.: s.n.], 2013. p. 423–424. Cited in page 39.

AGNIHOTRI, J.; PHALNIKAR, R. Development of Performance Testing Suite Using Apache JMeter. In: BHALLA, S. et al. (Ed.). **Proc. Intelligent Computing and Information and Communication**. Singapore: Springer Singapore, 2018. p. 317–326. ISBN 978-981-10-7245-1. Cited in page 39.

AMIRANTE, A. et al. Jattack: a WebRTC load testing tool. In: **Proc. Principles, Systems and Applications of IP Telecommunications**. [S.l.: s.n.], 2016. p. 1–6. Cited in page 39.

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2016. Cited in page 19.

ANDRAWOS, M.; HELMICH, M. **Cloud Native Programming with Golang: Develop microservice-based high performance web apps for the cloud with Go**. [S.l.]: Packt Publishing Ltd, 2017. Cited in page 59.

APTE, V. et al. AutoPerf: Automated Load Testing and Resource Usage Profiling of Multi-Tier Internet Applications. In: **Proc. ACM/SPEC International Conference on Performance Engineering**. New York, NY, USA: [s.n.], 2017. (ICPE '17), p. 115–126. ISBN 978-1-4503-4404-3. Disponível em: <http://doi.acm.org/10.1145/3030207.3030222>. Cited in page 39.

ARLITT, M.; KRISHNAMURTHY, D.; ROLIA, J. Characterizing the scalability of a large web-based shopping system. **ACM Transactions on Internet Technology**, v. 1, n. 1, p. 44–69, 2001. Cited in page 28.

AVRITZER, A. et al. Software performance testing based on workload characterization. In: ACM. **Proceedings of the 3rd international workshop on Software and performance**. [S.l.], 2002. p. 17–24. Cited in page 28.

BARBER, R. S. Beyond performance testing by. Citeseer, 2003. Cited in page 28.

BARBER, S. Creating effective load models for performance testing with incomplete empirical data. In: IEEE. **Proceedings. Sixth IEEE International Workshop on Web Site Evolution**. [S.l.], 2004. p. 51–59. Cited in page 28.

BARN, B.; BARAT, S.; CLARK, T. Conducting systematic literature reviews and systematic mapping studies. In: **Proceedings of the 10th Innovations in Software Engineering Conference**. New York, NY, USA: ACM, 2017. (ISEC '17), p. 212–213. ISBN 978-1-4503-4856-0. Disponível em: <http://doi.acm.org/10.1145/3021460.3021489>. Cited in page 33.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. p. 85–103. Cited in page 27.

BRUNE, P. Simulating User Interactions: A Model and Tool for Semi-realistic Load Testing of Social App Backend Web Services. In: **Proc. WEBIST**. [S.l.: s.n.], 2017. p. 235–242. Cited in page 39.

CHEUNG, C. M. K.; LEE, M. K. O. The asymmetric effect of website attribute performance on satisfaction: An empirical study. **Proceedings of the 38th Annual Hawaii International Conference on System Sciences**, p. 175c–175c, 2005. Cited in page 20.

CHUNYE, D.; WEI, S.; JIANHUA, W. Based on the analysis of mobile terminal application software performance test. In: **Proc. IEEE/ACIS 18th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing**. [S.l.: s.n.], 2017. p. 391–395. Cited in page 39.

CLARKE, P.; MALLOY, B. A. A Unified Approach to Implementation-Based Testing of Classes. In: **Proc. 1st Annual International Conference on Computer and Information Science**. [S.l.: s.n.], 2001. Cited in page 43.

COOK, T.; CAMPBELL, D. **Quasi-Experimentation: Design and Analysis Issues for Field Settings**. [S.l.]: Houghton Mifflin, 1979. Cited in page 49.

CRESWELL, J. W.; CRESWELL, J. D. **Research design: Qualitative, quantitative, and mixed methods approaches**. [S.l.]: Sage publications, 2017. Cited in page 78.

CUCOS, L.; DONCKER, E. de. “gRpas”, a Tool for Performance Testing and Analysis. In: **Proc. Springer-Verlag 5th International Conference on Computational Science - Volume Part I**. Berlin, Heidelberg: [s.n.], 2005. (ICCS’05), p. 322–329. ISBN 3-540-26032-3, 978-3-540-26032-5. Disponível em: <https://doi.org/10.1007/11428831_40>. Cited in page 39.

DALAL, S. R. et al. Model-based testing in practice. In: **Proc. IEEE International Conference on Software Engineering**. [S.l.: s.n.], 1999. p. 285–294. ISSN 0270-5257. Cited 2 times in pages 40 and 45.

DENARO, G.; POLINI, A.; EMMERICH, W. Early performance testing of distributed software applications. In: **Proceedings of the 4th International Workshop on Software and Performance**. New York, NY, USA: ACM, 2004. (WOSP ’04), p. 94–103. ISBN 1-58113-673-0. Disponível em: <http://doi.acm.org/10.1145/974044.974059>. Cited in page 28.

DEVASENA, M. S. G.; KUMAR, V. K.; GRACE, R. K. LTTC: A Load Testing Tool for Cloud. In: MODI, N.; VERMA, P.; TRIVEDI, B. (Ed.). **Proc. Springer Singapore International Conference on Communication and Networks**. [S.l.: s.n.], 2017. p. 689–698. ISBN 978-981-10-2750-5. Cited in page 39.

DILLENSEGER, B. CLIF, a framework based on Fractal for flexible, distributed load testing. **annals of telecommunications - annales des télécommunications**, v. 64, n. 1, p. 101–120, 2009. ISSN 1958-9395. Disponível em: <https://doi.org/10.1007/s12243-008-0067-9>. Cited in page 39.

DIN, G.; PETRE, R.; SCHIEFERDECKER, I. A workload model for benchmarking ims core networks. In: IEEE. **IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference**. [S.l.], 2007. p. 2623–2627. Cited in page 63.

ENGSTRöM, E.; PETERSEN, K. Mapping software testing practice with software testing research — SERP-test taxonomy. In: **Proc. IEEE 8th International**

Conference on Software Testing, Verification and Validation Workshops. [S.l.: s.n.], 2015. p. 1–4. Cited in page 50.

FAN, H.; MU, Y. A performance testing and optimization tool for system developed by Python language. Institution of Engineering and Technology, p. 24–27(3), 2013. Cited in page 39.

FELDT, R.; MAGAZINIUS, A. Validity threats in empirical software engineering research-an initial survey. In: **Seke**. [S.l.: s.n.], 2010. p. 374–379. Cited 2 times in pages 78 and 79.

FERRARI, D. On the foundations of artificial workload design. In: **Proceedings of the 1984 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems**. New York, NY, USA: ACM, 1984. (SIGMETRICS '84), p. 8–14. ISBN 0-89791-141-5. Disponível em: <http://doi.acm.org/10.1145/800264.809309>. Cited 2 times in pages 19 and 30.

GAO, J.; TSAO, H.-S.; WU, Y. **Testing and quality assurance for component-based software**. [S.l.]: Artech House, 2003. Cited in page 28.

GIRARDON, G. PerfMoon: Proposal of a Tool for Monitoring the Performance of Web Applications. **forthcoming**, 2019. Cited in page 67.

HABUL, A.; KURTOVIC, E. Load testing an AJAX application. In: **Proc. IEEE 30th International Conference on Information Technology Interfaces**. [S.l.: s.n.], 2008. p. 729–732. Cited in page 39.

HAMED, O.; KAFRI, N. Performance testing for web based application architectures (.NET vs. Java EE). In: **Proc. First International Conference on Networked Digital Technologies**. [S.l.: s.n.], 2009. p. 218–224. ISSN 2155-8728. Cited in page 39.

JOHN, L. K.; EECKHOUT, L. **Performance evaluation and benchmarking**. [S.l.]: CRC Press, 2018. Cited in page 63.

JOVIC, M. et al. Automating Performance Testing of Interactive Java Applications. In: **Proc. ACM 5th Workshop on Automation of Software Test**. New York, NY, USA: [s.n.], 2010. (AST '10), p. 8–15. ISBN 978-1-60558-970-1. Cited in page 39.

KALITA, M.; BEZBORUAH, T. Investigation on performance testing and evaluation of PReWebD: a .NET technique for implementing web application. **IET Software**, v. 5, n. 4, p. 357–365, 2011. ISSN 1751-8806. Cited in page 39.

KAMRA, M.; MANNA, R. Performance of Cloud-Based Scalability and Load with an Automation Testing Tool in Virtual World. In: **Proc. IEEE 8th World Congress on Services**. [S.l.: s.n.], 2012. p. 57–64. ISSN 2378-3818. Cited in page 39.

KHAN, R.; AMJAD, M. Web application's performance testing using HP LoadRunner and CA Wily introscope tools. In: **Proc. International Conference on Computing, Communication and Automation**. [S.l.: s.n.], 2016. p. 802–806. Cited 2 times in pages 29 and 39.

KIM, G.-H.; KIM, Y.-G.; CHUNG, K.-Y. Towards virtualized and automated software performance test architecture. **Multimedia Tools and Applications**, v. 74, n. 20, p. 8745–8759, 2015. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-013-1536-3>. Cited in page 39.

KIM, H.; CHOI, B.; WONG, W. E. Performance Testing of Mobile Applications at the Unit Test Level. In: **Proc. IEEE 3rd International Conference on Secure Software Integration and Reliability Improvement**. [S.l.: s.n.], 2009. p. 171–180. Cited in page 39.

KIRAN, S.; MOHAPATRA, A.; SWAMY, R. Experiences in performance testing of web applications with Unified Authentication platform using Jmeter. In: **Proc. International Symposium on Technology Management and Emerging Technologies**. [S.l.: s.n.], 2015. p. 74–78. Cited in page 39.

KITCHENHAM, B. A. **Guidelines for performing Systematic Literature Reviews in software engineering. EBSE Technical Report EBSE-2007-01**. [S.l.: s.n.], 2007. Cited 2 times in pages 33 and 49.

KOZIOLEK, H. Goal, question, metric. In: **Dependability metrics**. [S.l.]: Springer, 2008. p. 39–42. Cited in page 33.

KRISHNAMURTHY, D.; ROLIA, J. A.; MAJUMDAR, S. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. **IEEE Transactions on Software Engineering**, v. 32, n. 11, p. 868–882, 2006. ISSN 0098-5589. Cited in page 39.

KRIŽANIĆ, J. et al. Load testing and performance monitoring tools in use with AJAX based web applications. In: **Proc. IEEE 33rd International Convention MIPRO**. [S.l.: s.n.], 2010. p. 428–434. Cited 2 times in pages 39 and 40.

LEE, J.; BEN-NATAN, R. **Integrating Service Level Agreements: Optimizing Your OSS for SLA Delivery**. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471428663. Cited 2 times in pages 30 and 49.

LI, P.; SHI, D.; LI, J. Performance test and bottle analysis based on scientific research management platform. In: **Proc. 10th International Computer Conference on Wavelet Active Media Technology and Information Processing**. [S.l.: s.n.], 2013. p. 218–221. Cited in page 39.

MAâLEJ, A. J.; HAMZA, M.; KRICHEN, M. WSCLT: A Tool for WS-BPEL Compositions Load Testing. In: **Proc. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises**. [S.l.: s.n.], 2013. p. 272–277. ISSN 1524-4547. Cited in page 39.

MEMON, A. M.; SOFFA, M. L. Regression testing of GUIs. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 28, n. 5, p. 118–127, 2003. Cited 2 times in pages 40 and 45.

MENASCÉ, D. A. Tpc-w: A benchmark for e-commerce. **IEEE Internet Computing**, IEEE, v. 6, n. 3, p. 83–87, 2002. Cited in page 63.

MICHAEL, N. et al. CloudPerf: A Performance Test Framework for Distributed and Dynamic Multi-Tenant Environments. In: **Proc. ACM/SPEC 8th International Conference on Performance Engineering**. New York, NY, USA: ACM, 2017. (ICPE '17), p. 189–200. ISBN 978-1-4503-4404-3. Disponível em: <http://doi.acm.org/10.1145/3030207.3044530>. Cited in page 39.

MINAYO, M. C. d. S.; DESLANDES, S. F.; GOMES, R. C. Pesquisa social: teoria, método e criatividade. In: **Pesquisa social: teoria, método e criatividade**. [S.l.: s.n.], 2015. Cited in page 23.

MYERS, G. J.; SANDLER, C. **The Art of Software Testing**. USA: John Wiley & Sons, 2004. ISBN 0471469122. Cited in page 19.

NETTO, M. A. S. et al. Evaluating Load Generation in Virtualized Environments for Software Performance Testing. In: **Proc. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum**. [S.l.: s.n.], 2011. p. 993–1000. ISSN 1530-2075. Cited in page 39.

PATTON, R. **Software Testing (2Nd Edition)**. Indianapolis, IN, USA: Sams, 2005. ISBN 0672327988. Cited in page 27.

PERRY, W. E. **Effective methods for software testing: Includes complete guidelines, Checklists, and Templates**. [S.l.]: John Wiley & Sons, 2007. Cited in page 19.

PODELKO, A. Reinventing performance testing. In: CMG. [S.l.], 2016. Cited in page 39.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. [S.l.]: Editora Feevale, 2013. Cited 2 times in pages 23 and 24.

PU, Y.; XU, M. Load testing for web applications. In: **Proc. IEEE First International Conference on Information Science and Engineering**. [S.l.: s.n.], 2009. p. 2954–2957. Cited in page 39.

PUTRI, M. A.; HADI, H. N.; RAMDANI, F. Performance testing analysis on web application: Study case student admission web system. In: **Proc. International Conference on Sustainable Information Engineering and Technology**. [S.l.: s.n.], 2017. p. 1–5. Cited 2 times in pages 29 and 39.

RAMANATHAN, N. **Part 21: Goroutines**. 2017. Disponível em: <https://golangbot.com/goroutines/>. Cited in page 60.

RODRIGUES, E. M. et al. PLeTsPerf - A Model-Based Performance Testing Tool. In: **Proc. IEEE 8th International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2015. p. 1–8. ISSN 2159-4848. Cited in page 39.

RODRIGUES, E. M. et al. Evaluating capture and replay and model-based performance testing tools: an empirical comparison. In: **Proc. ACM 8th International Symposium on Empirical Software Engineering and Measurement**. [S.l.: s.n.], 2014. p. 9. Cited in page 39.

SHAMS, M.; KRISHNAMURTHY, D.; FAR, B. A model-based approach for testing the performance of web applications. In: **Proc. ACM 3rd international workshop on Software quality assurance**. [S.l.: s.n.], 2006. p. 54–61. Cited in page 19.

SINGH, M.; SINGH, R. Load Testing of web frameworks. In: . [S.l.: s.n.], 2012. p. 592–596. ISBN 978-1-4673-2922-4. Cited in page 39.

SMITH, C. U. **Performance Engineering of Software Systems**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN 0201537699. Cited in page 28.

STUPIEC, E.; WALKOWIAK, T. Automatic Load Testing of Web Application in SaaS Model. In: ZAMOJSKI, W. et al. (Ed.). **Proc. Springer International New Results in Dependability and Computer Systems**. Heidelberg: [s.n.], 2013. p. 421–430. ISBN 978-3-319-00945-2. Cited in page 39.

TOGASHI, N.; KLYUEV, V. Concurrency in go and java: performance analysis. In: IEEE. **2014 4th IEEE International Conference on Information Science and Technology**. [S.l.], 2014. p. 213–216. Cited in page 59.

VOKOLOS, F. I.; WEYUKER, E. J. Performance testing of software systems. In: ACM. **Proceedings of the 1st International Workshop on Software and Performance**. [S.l.], 1998. p. 80–87. Cited in page 63.

WOODSIDE, M.; FRANKS, G.; PETRIU, D. C. The future of software performance engineering. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. p. 171–187. Cited 2 times in pages 19 and 28.

WU, Q.; WANG, Y. Performance testing and optimization of J2EE-based web applications. In: **Proc. IEEE Second International Workshop on Education Technology and Computer Science**. [S.l.: s.n.], 2010. v. 2, p. 681–683. Cited in page 39.

YAN, M. et al. Delivering Web service load testing as a service with a global cloud. v. 27, n. 3, 2014. Cited in page 39.

YAN, M. et al. Building a TaaS Platform for Web Service Load Testing. In: **Proc. IEEE International Conference on Cluster Computing**. [S.l.: s.n.], 2012. p. 576–579. ISSN 1552-5244. Cited in page 39.

YAN, M. et al. WS-TaaS: A Testing as a Service Platform for Web Service Load Testing. In: **Proc. IEEE 18th International Conference on Parallel and Distributed Systems**. [S.l.: s.n.], 2012. p. 456–463. ISSN 1521-9097. Cited in page 39.

YAN, X. et al. Performance Testing of Open Laboratory Management System Based on LoadRunner. In: **Proc. IEEE First International Conference on Instrumentation, Measurement, Computer, Communication and Control**. [S.l.: s.n.], 2011. p. 164–167. Cited in page 39.

ZHANG, L. et al. Design and implementation of cloud-based performance testing system for web services. In: **Proc. IEEE 6th International Conference on Communications and Networking in China**. [S.l.: s.n.], 2011. p. 875–880. Cited in page 39.

ZHOU, J.; ZHOU, B.; LI, S. Automated Model-Based Performance Testing for PaaS Cloud Services. In: **Proc. IEEE 38th International Computer Software and Applications Conference Workshops**. [S.l.: s.n.], 2014. p. 644–649. Cited in page 39.

INDEX