Scalability performance measurement and testing of cloud-based software services

By

Amro Mohammad Hani Kh. Al-Said Ahmad

Submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Keele University

June 2019

In loving memory of my late Father

(20 April 1943 – 10 Dec 2018)

To my loving Mother

Abstract

Cloud-based software services have become more popular and dependable and are ideal for businesses with growing or changing workload demands. These services are increasing rapidly due to the reduced hosting costs and the increased availability and efficiency of computing resources. The delivery of cloud-based software services is based on the underlying cloud infrastructure supported by cloud providers, which delivers the potential for scalability that follows the pay-as-you-go model. Performance and scalability testing and measurements of those services are necessary for future optimisations and growth of cloud computing to support the Service Level Agreement (SLA) compliant quality of cloud services, especially in the context of rapidly expanding quantity of service delivery.

This thesis addresses an important issue, understanding the scalability of cloud-based software services from a technical perspective, which is very important as more software solutions are migrated to the cloud. A novel testing and quantifying approach for the scalability performance of cloud-based software services is described. Two technical scalability metrics for software services that have been deployed and distributed in cloud environments, have been formulated: volume and quality scalability metrics based on the number of software instances and the average response time.

The experimental analysis comprises three stages. The first stage involves demonstrating the approach and the metrics using real-world cloud-based software service running on Amazon EC2 cloud using three demand scenarios. The second stage aims to extend the practicality of the metrics with experiments on two public cloud environments (Amazon

EC2 and Microsoft Azure) with two cloud-based software services to demonstrate the use of these metrics. The experimental analysis considers three sets of comparisons to provide the platform to construct the metrics as a basis that can be used effectively to compare the scalability of software on cloud environments, consequently supporting deployment decisions with technical arguments. Moreover, the work integrates the technical scalability metrics with an earlier utility-oriented scalability metric. The third stage is a case study of application-level fault injection using real-world cloud-based software services running on Amazon EC2 cloud to demonstrate the effect of fault scenarios on the scalability behaviour.

The results show that the technical metrics quantify explicitly the technical scalability performance of the cloud-based software services, and that they allow clear assessment of the impact of demand scenarios, cloud platform and fault injection on the software services' scalability behaviour. The studies undertaken in this thesis have provided a valuable insight into the scalability of cloud-based software services delivery.

Acknowledgements

First, I would fully express my deep appreciation and sincere gratitude to my supervisor Prof. Peter Andras, for his constant support, motivation, encouragement, and invaluable advice. I am grateful to have work with him. I also like to thank Prof. Pearl Brereton for her supervision and support during the conduct of the literature review in this work. I would also like to thank Prof. Fiona Polack for her support during my time here at Keele, especially for her useful comments and suggestions for this work. I would also like to thank my thesis examiners Prof. Karim Djemame (Leeds University) and Dr. Ed de Quincey (Keele University) for their discussion and comments.

I am particularly grateful to the financial support provided by Philadelphia University – Jordan, with a full three years PhD scholarship.

I like to thank my friends and colleagues at the School of Computing and Mathematics, Keele University. I would like to thank all the PhD students (past and present) in the school who our path has crossed. Thank you all for your friendship and support.

Last but not least, my endless and deepest appreciations go to my Mother, Bothers, Sister, and Friends for their continuing love and support. Finally, without the commitment and motivation of my late Father, I cannot think what state my life would be in now.

Author's Declaration

During this Ph.D. a significant portion of work has already been published or been sent for publication. Details of papers along with Journal, conference, poster and seminar activity, are presented in this section:

Journal Papers

- A. Al-Said Ahmad and P. Andras, "**Scalability Analysis Comparisons of Cloud-based Software Services**," In revision cycle, Journal of Cloud Computing, Springer.
- A. Al-Said Ahmad and P. Andras, "**Cloud-based Software Services Delivery from the Perspective of Scalability**," International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, 2019. https://doi.org/10.1080/17445760.2019.1617864

Refereed Conference Papers

- A. Al-Said Ahmad and P. Andras, "**Measuring and Testing the Scalability of Cloud-based Software Services**," The Fifth IEEE International Symposium on Innovation in Information and Communication Technology (ISIICT), Amman, 2018, pp. 1-8. https://doi.org/10.1109/ISIICT.2018.8613297
- A. Al-Said Ahmad and P. Andras, "**Measuring the Scalability of Cloud-Based Software Services**," 2018 IEEE World Congress on Services (SERVICES), San Francisco, CA, 2018, pp. 5-6. https://doi.org/10.1109/SERVICES.2018.00016
- A. Al-Said Ahmad, P. Brereton and P. Andras, "**A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods**," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, 2017, pp. 555-562. https://doi.org/10.1109/QRS-C.2017.94

External Talks

- **Measuring and Testing the Scalability of Cloud-based Software Services.** 2018 Fifth IEEE International Symposium on Innovation in Information and Communication Technology (ISIICT 2018), Amman, Jordan, November 2018.
- **Measuring the Scalability of Cloud-based Software Services.** The 2018 IEEE World Congress on Services (SERVICES 2018), San Francisco, USA, July 2018.

- **A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods**. The IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C 2017), Prague, Czech Republic, July 2017.

Internal Talks

- **Scalability Analysis of Cloud-based Software Services,** Computer Science Journal Club, Keele University, February 2019.
- **Cloud-based Software Services Delivery from the Perspective of Scalability**, 8th Computing and Mathematics Postgraduate Research Day, Keele University, April 2018.
- **A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods**, The Faculty of Natural Sciences' Postgraduate Symposium, Keele University, May 2017.
- **Software Cloud Testing: A Review**, The Faculty of Natural Sciences' Postgraduate Symposium, Keele University, June 2016.
- **Software Testing Using Cloud Computing Resources**, 6th Computing Postgraduate Research Day, Keele University, April 2016.

Posters

- **Measuring the Scalability of Cloud-based Software Services**, The Faculty of Natural Sciences' Postgraduate Symposium, Keele University, May 2018.
- **Software Large-Scale Testing over the Cloud,** The Institute of Liberal Arts and Sciences' Postgraduate Conference, Keele University, April 2016.

Table of Contents

List of Tables

List of Figures

Abbreviations

SLA	Service level agreement
QoS	Quality of Service
SUT	Software under test
SaaS	Software-as-a-Service
IaaS	Infrastructure-as-a-Service
PaaS	Platform-as-a-Service
TaaS	Testing-as-a-Service
Amazon EC2	Amazon elastic compute cloud
VMs	Virtual machines
η^{I}	Volume scalability metric.
η^{t}	Quality scalability metric.
D and D′	Service demand volumes.
I and I′	The corresponding number of software instances.
t_r and t'_r	The corresponding average response times.
NFS	Network file system
RIA	Rich internet application
GUI	Graphical user interface
YCSB	Yahoo! Cloud serving benchmark
CAGR	Compound annual growth rate
CVM	Commercial virtual machine monitor
KVM	Kernel-based virtual machine
AWS	Amazon web services

REST	Representational state transfer
ALFI	Application-level fault injection
SR	Systematic review
SDLC	The software development life cycle

Chapter 1 Introduction

This chapter introduces the overall focus of this thesis and places the motivation for the research into context. An introduction to scalability performance measurements and testing of cloud-based software services is provided. The research objectives and questions are explained. The novelty of the thesis and how it contributes to knowledge is also stated. Finally, the structure of the thesis is presented.

1.1 Introduction and Motivation

The delivery of Cloud-based software services is based on the underlying cloud infrastructure including networking, operating systems, servers, and storage capability [1]. Such software services are expected to scale up and down depending on the usage demand, supported by the virtual scaling infrastructure provided by the cloud providers [2]. Such software services are provided as Software-as-a-Service (SaaS) which are on-demand applications that follow the pay-as-you-go model [3]. In general, the delivery of cloud-based software services

is supported by the provision of Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) cloud computing services [4].

The number of cloud-based applications is increasing rapidly due to the reduced hosting costs and the increased availability and efficiency of computing resources. To maximise the scalability and performance of any software system, it is essential to incorporate performance and scalability testing and assessment into the development life cycle [5]. This provides an important foundation for future optimisation and supports the Service Level Agreement (SLA) compliant quality of cloud services, especially in the context of rapidly expanding the quantity of service delivery [5], [6].

Scalability testing of cloud-based software services is key for both performance measurements and the technology settings. Furthermore, scalability testing is necessary for the delivery of business objectives, i.e. gaining more users interacting with the system [5], [7].

As important as measuring and testing scalability is, so is to collect the right measurements, and to interpret those measurements using the right metrics. This thesis will develop a consistent interpretation of the fine-grained performance measurement data through the lenses of relevant scalability performance metrics. This interpretation enables a better understanding of the factors that influence performance metrics of the scalability of cloud-based systems and will help software engineers to fine-tune such systems to achieve better performance.

Cloud computing, Auto Scaling and Load Balancing features provide the support for cloud-based applications to be more scalable, which enables such applications to deal with sudden workload changes by adding or dropping instance(s) at runtime. Furthermore, as cloud-based applications are being offered as SaaS and multi-tenancy architectures are being used [8], there is an increased need for scalability that supports the availability and productivity of the services and on-demand resources.

A relevant review [9] on provisioning of cloud resources and related research challenges identifies, among others, predictable performance and scalable resource management as key challenges. Gao et al. [10] reviewed testing in relation to cloud-based software services. They highlight scalability and performance testing as major research directions.

There are three typical requirements that are associated with the performance of cloud-based applications: scalability, elasticity, and efficiency [11], [12]. In this thesis, the technical definitions of these performance features provided by Lehrig et al. [13] have been adopted. Scalability is the ability of the cloud layer to increase the capacity of the software service delivery by expanding the quantity of software service that is provided. Elasticity is the level of autonomous adaptation provided by the cloud layer in response to variable demand for the software service. Efficiency is the measure of matching the quantity of software service available for delivery with the quantity of demand for the software service.

However, it should be noted that alternative utility-oriented (i.e. economic cost/benefit-focused) approaches are also used in the literature for the conceptualisation and measurement of these performance aspects of cloud-based services [14], [15]. Technical scalability measurements and testing are essential when assessing and measuring the performance of cloud-based software services [5], [7]. Both elasticity and efficiency aspects depend on the level of scalability performance.

According to a systematic review of the relevant reports in the literature, there are only a few research studies (e.g. project reports, MSc dissertations) which attempt to address the assessment of the technical scalability of cloud-based software services [13]. However, recently, a number of publications addressed the technical measurement of the elasticity of cloud-based provision of software services (i.e. [16], [17]). On the other hand, other recent publications addressed the scalability of cloud-based software services from a utility perspective [14]–[16], [18].

To attempt to improve the scalability of any software system, there is a need to understand the system's components that affect and contribute to the scalability performance of the service. This could help to design suitable test scenarios and provide a basis for future studies aiming to maximise the scalability performance. Assessing the scalability from a utility perspective is insufficient for the above purpose, as it works from an abstract perspective that is not necessarily closely related to the technical components and features of the system.

Technical scalability metrics provide the baseline for more detailed investigations of cloud-based software services' scalability performance. Fault injection at the application level would help to evaluate the application's response to those artificial faults [19] over the quality aspects of cloud-based software services, such as performance, scalability, and security. Therefore, comparing the scalability performance of a cloud-based software service after a fault-injection attack with the performance analyses with normal workload will provide an indication about the resiliency of that software service and how the scalability behaviour of such application will be impacted in such fault scenarios.

Such analysis and metrics of scalability behaviour can help practitioners; such as software developers, testers, cloud providers, and cloud consumers, to compare cloud software systems rapidly and can be a useful tool in evaluating the usage and quality of software services. Performance and scalability testing of cloud-based software services is important, in order to validate the reliability of the software system for changing or increasing workload demands, this may also help to determine the cloud infrastructure support such services to be able to scale when demand change. Testing of such applications is important as more SaaS solutions are migrated to the cloud, in order to offer a compatible solution that is suitable for business with growing or changing workload demands.

It is clear that the technical analysis of scalability measurements and testing of cloud-based software services is critical for the delivery of such services and the development of cloud computing. Therefore, an in-depth investigation to analyse

and compare different delivery platforms for such services would help practitioners to gain a better understanding of assessing and testing the scalability of cloud-based delivery of software services in technical terms. Furthermore, integrating technical and utility-oriented metrics will enhance the analysis of software services' scalability from both technical and production-driven perspectives.

1.2 Thesis Objectives

This thesis is primarily concerned with measuring and testing the scalability performance of cloud-based software services from a technical perspective. The objective of this thesis is not only to contribute to our understanding of the scalability performance of cloud-based software services, but also to provide a better understanding of how to test and measure the scalability of such services from a technical perspective. Developing and using technical scalability metrics can help to identify differences in the behaviour of the assessed system in the context of different usage scenarios and cloud platforms. It also enables an understanding of how components of the cloud-based software service system that contribute to the scalability performance of the system help in designing appropriate test scenarios and identifying options for changes and upgrades that can improve the scalability performance of the system. The main objectives of this thesis are:

- To identify the current empirical practice in the area of cloud-based software testing, especially in the area of measuring and testing the scalability of cloud-based software services;
- To identify and collect the right measurements after testing the scalability of cloud-based software services from a technical perspective to interpret those measurements into the right technical scalability metrics;
- To develop efficient metrics that can support effective measurements and testing for the scalability performance of software services from a technical perspective to highlight differences in the system's behaviour based on different scaling scenarios, cloud platforms, and software services;
- To integrate the technical metrics with previous proposed utility-oriented approaches to measuring scalability to enable the scalability analysis from both technical and production-driven perspectives;
- To use metrics to compare the scalability of software on cloud environments and consequently to support deployment decisions with technical arguments; and
- To determine how faults, affect the scalability behaviour of cloud-based software services when using application-level fault injection.

1.3 Research Questions

The overall aim of this thesis is to investigate the scalability performance measurements and testing of cloud-based software services. The above described objectives are synthesised into four research questions.

RQ1: How can we test the scalability of cloud-based software services?

RQ2: What do we measure in relation to the technical scalability of cloud-based software services?

RQ3: How do we interpret the technical scalability performance measurements?

RQ4: How can faults affect the scalability of cloud-based software services?

1.4 Original Contributions

This thesis reports a novel investigation into the scalability performance measurements and testing of cloud-based software services. The methodology of testing and quantifying the scalability measurements of cloud-based services presented in this thesis is an original work on providing scalability metrics for such services from a technical perspective for both volume and quality scaling. Furthermore, an earlier metric of scalability from a utility-oriented perspective is integrated with the presented technical metrics to analyse the scalability

performance of cloud-based software services from both technical and production-driven perspectives. The demonstration of this methodology involves using two public cloud environments (Amazon EC2 and Microsoft Azure), multiple cloud-based software services (both open-source applications and those that can be rented through the SaaS marketplaces), different usage demand scenarios, and different hardware and software settings. A new case study of application-level fault-injection testing for measuring the scalability of cloud-based software systems is described. The remainder of this section provides more detail on how the work has contributed to knowledge in this area. A significant portion of the work in this thesis has already been published or been sent for publication. All publications have been through a peer-review process to accommodate for crossover between published works, and therefore, some of the chapters can include one or more publications. Details of published work and more details on how these works contributed to the knowledge are as follows:

- Al-Said Ahmad et al. [20]: "A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods". Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, IEEE, 555-562. The work, which was selected for an oral presentation, appears here in Chapter 2. The systematic mapping study investigates the empirical studies in the software cloud testing area, performed in the early stages of the research, which provided the related empirical works in the area of scalability performance measurements and testing of cloud-based software services. This study allows us to obtain a clear view of the current empirical work and

practice in the whole area of cloud testing, and more precisely in the area of scalability performance of cloud-based software services.

- Al-Said Ahmad and Andras [21]: "Measuring the Scalability of Cloud-based Software Services". Proceedings of 2018 IEEE World Congress on Services (SERVICES), IEEE, 5-6. The work was selected for an oral presentation. The work introduces a novel approach to measure and quantify scalability of cloud-based software services and explains the metrics based on the measurement approach. The approach of quantifying scalability presented in this thesis continues to evolve through to the most recent publication.

- Al-Said Ahmad and Andras [22]: "Measuring and Testing the Scalability of Cloud-based Software Services". Proceedings of 2018 IEEE Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT), IEEE, 1-8. The work was selected for an oral presentation. The work provides more explanations of the approach to measure and quantify scalability, and explains the volume and quality scaling metrics for evaluating cloud-based software services' scalability performance based on the measurement approach. This work introduces the demand scenarios and demonstrates a practical example of the metrics. This work established the need to determine how the technical scalability metrics can be integrated into an earlier utility-oriented metric of scalability. The work has been invited for an extended version for journal publication.

- Al-Said Ahmad and Andras [23]: "Cloud-based Software Services Delivery from the Perspective of Scalability". The work presents an extension publication from the previous work [22] by including an additional evaluation scenario, a description of the related experiments and results, more details in the explanation of the results, and discussion of the new experimental results in relation to the proposed metrics. The work shows how to integrate the technical scalability metrics into an earlier utility-oriented metric of scalability and calculate the values for each demand scenario to enable the scalability analysis from technical and production-driven perspectives. This work appears here in Chapter 4. The work has been published in the International Journal of Parallel, Emergent and Distributed Systems, published by Taylor and Francis.

- Al-Said Ahmad and Andras [24]: "Scalability Analysis Comparisons of Cloud-based Software Services". This work uses two cloud-based systems to demonstrate the usefulness of the technical metrics and compare their scalability performance in two cloud platforms: Amazon EC2 and Microsoft Azure. The experimental analysis considers three sets of comparisons: first, comparing the same cloud-based software service hosted on two different public cloud platforms; second, comparing two different cloud-based software services hosted on the same cloud platform; and finally, comparing the same cloud-based software service hosted on the same cloud platform with two different auto-scaling policies. The work not only provides an extension of the applicability of the metrics, but also provides the platform to construct the technical scalability metrics as a basis to effectively comparing the scalability of

software on cloud environments, and supporting deployment decisions with technical arguments. This work is presented here in Chapter 5. The manuscript is under review, following second revision, in the Journal of Cloud Computing: Advances, Systems and Applications, published by Springer.

- A case study of application level fault injection (ALFI) testing for measuring the scalability of cloud-based software system, using Amazon EC2. An experimental approach has been explained, combining four components; workload generator, software fault, scalability measures, and the system under test and its environment. Here we simulate delay latency injection with two different times; 800 and 1600 ms, and compared the results with the baseline data. The results show that the proposed approach allows clear assessment of the fault scenario impact on the cloud-based software service' scalability performance. The work is being prepared for submission for publication.

1.5 Thesis Outline

The remainder of this thesis is organised as outlined below.

Chapter 2 provides a novel investigation of the empirical studies of cloud software testing. The mapping study identifies and classifies cloud testing methods, the application of these methods, and the purpose of testing using these methods. The systematic review has been used together with an additional review of the

literature to update the background related to the scalability performance measurement and testing of cloud-based software services.

Chapter 3 describes the road map of the methodology that the researcher followed during the study, including the scalability testing methodology and planning following the IEEE 829 standards. In this chapter, the cloud platforms, services, software applications and load generators used in this study are also described. Moreover, the cloud elasticity concept is described as well.

Chapter 4 describes the implementation of an application example using three different usage scenarios to demonstrate the measurement approach and metrics using a concrete cloud-based software service (OrangeHRM) run through the Amazon EC2 Cloud. The calculation of both technical metrics and integrated metrics values is reported here to ascertain the impact of using demand scenarios on the scalability behaviour and delivery.

Chapter 5 describes experiments on two public cloud environments (AWS, Azure) with two cloud-based applications (MediaWiki, OrangeHRM) to demonstrate the use of the quality and volume scalability metrics. The experimental analysis considers three sets of comparisons: first, comparing the same cloud-based software service hosted on two different public cloud platforms; second, comparing two different cloud-based software services hosted on the same cloud platform; and finally, comparing the same cloud-based software service hosted on the same cloud platform with two different auto-scaling policies. The results show that the metrics can be used effectively to compare the scalability of software on

cloud environments and consequently to support deployment decisions with technical arguments.

Chapter 6 describes a preliminary experimental analysis of ALFI to investigate the scalability performance of cloud-based software services has been presented. The experimental approach has been explained, combining four components. A case study was demonstrated using a cloud-based software service run on the EC2 cloud platform, considering one demand scenario and one type of fault. Our results show that the proposed approach allows clear assessment of the impact of fault scenario on the cloud-based software service' scalability performance.

In Chapter 7, the findings from the different studies reported in this thesis are brought together and discussed in relation to the thesis research questions and objectives. Place the work into the related work and compare it with closest research, also some metrics deployment challenges are presented. The research limitations are also outlined.

Chapter 8 concludes this work by addressing the contributions made in this thesis and proposes a number of possible future research directions.

Chapter 2 Literature Review

This chapter details and reviews the empirical practice of cloud software testing in general; moreover, there will be a review of relevant studies and techniques used to test and measure the scalability performance of cloud- based software services.

First, an in-depth investigation into the empirical studies has been conducted in the area of cloud software testing from 2010-2015, in order to identify and classify the state-of-the-art of the area of software cloud-based testing. Manual and automatic search strategies, and snowballing technique were used in order to identify the primary studies. A set of procedures have been adopted to validate the result of the mapping study; including checking all of the primary studies that are reported in the previous related reviews, and a team of two reviewers performed extraction of data from a random sample of studies. After applying, the methodology 75 research papers were identified as the final set of primary studies.

The mapping study highlights that studies present primarily preliminary results, often describing an example of the software cloud-based testing methods or a simple application experiment to evaluate the proposed approach. This mapping study is presented in Section 2.1. The study reported in this section has been

published in the 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (Al-Said Ahmad et.al.[20]).

During the work on the reported research, a further review of the related literature was performed. This covered the area of scalability measurements and testing on cloud-based software services, and works related to this area of research. This was done in order to ensure all relevant works published following the systematic mapping study, have been identified. This additional review of the literature is reported in Section 2.2. The discussion of the literature review implications is outlined in Section 2.3. Finally, Section 2.4 summarises this chapter.

2.1 A Systematic Mapping Study of Software Cloud Testing Methods

2.1.1 Introduction

Systematic review (SR) is a methodology that aims to be reliable, exhaustive, and auditable to allow researchers to collect evidence on a particular research question, topic area, or subject of interest [25],[26]. The SR plays a major role in supporting academic research as well as enriching practices in software engineering [27]. The SR process starts with the development and validation of a review protocol [25]. The review protocol provides a plan for the process of conducting a review, including study selection and data extraction, with the aim to answer the research

questions [28]. The protocol preparation is followed by locating potentially relevant studies in an automatic or manual way, selecting primary studies based on inclusion and exclusion criteria, extracting data, and reporting the SR, including its limitations [25]. A mapping study is a form of SR which provides a classification of the relevant research for a particular subject without necessarily assessing the quality of each study [25].

Software testing is one of the main technical activities in the software development cycle, which consumes more than 30% of a project's budget, effort, and time [29]. When the budget and time are not sufficient to cover all test cases, suites, and scenarios, an efficient strategy that involves tools and technical solutions will be key to enhancing and speeding up the testing process.

Cloud computing provides integrated services that help to create an environment for speeding up the development process by allowing organizations to transfer some of the development processes -such as testing, deployment, installations, and tracking failures- into the cloud. In the context of testing, cloud computing has been described as a resource that offers virtualization, storage, and software services that can reduce the time and cost of managing and applying large test suites [30]. Virtualization can be used in large-scale testing [31], and the cloud can support on-demand test laboratories [32]. Furthermore, it can be used for auto-run and management of test suites [33]. On the other hand, the cloud has changed the way services are delivered. As cloud-based services have grown in popularity, so has the need for testing those services.

This section presents a mapping study that addresses the functional and non-functional testing methods on/using cloud-based services. The study provides an overview of primary studies, published in the period of 2010-2015, that evaluate cloud testing methods. The methodology is based on a well-defined protocol to build a structure and classification scheme to analyse the research area of cloud-based testing (see Appendix A, for change records). This mapping study collected 247 research papers from which a total of 69 primary studies reported in 75 research papers were selected. The study look at how methods are applied, and what is being tested using those methods. Several papers that report the same study are included as a group. Each study has been identified using the notation [S+ID] where ID is the numeric identifier of the study – the study ID is included and highlighted at the end of the bibliographic data of each appropriate paper in the reference list.

2.1.2 Related Work

There have been a number of literature surveys and reviews and one mapping study within the software cloud testing area. A systematic mapping study is reported in two research papers, [2][34], using the 5W+1H (who, what, where, when, why, how) model for reporting systematic reviews. Studies are categorized based on research questions, authors and countries, research objectives, research ideas, patterns of papers on different types of cloud service and publication type, immediacy of article citation, and article inter-relevance. The mapping study does

not include clear inclusion/exclusion criteria, however. Further, the study covers published papers dated during the period 2010-2012. In contrast, this study focuses on various subjects, such as research aim and objectives, functional and non-functional testing methods, and test coverage. Moreover, this study has a well-defined protocol and clear constraints regarding the studies' selection and categories, as it includes only studies which provide an evaluation of the testing method used.

Some literature surveys have been published in conferences and journals. In particular, one study focuses on publications dated during the period 2009-2012 and classifies relevant literature according to the type of testing activities for cloud services and the type of application domains [35]. An overview of research related to cloud testing tools, types, and challenges, and a comparison of testing tools are presented in [36]. A survey that identifies the need for cloud testing tools and presents the current testing methods and tools has also been published [37]. Studies [38][39] provide an overview of software testing as a service (TaaS), while literature survey [40] highlighting the current situation of security measurement and testing on the cloud. Study [10] discusses SaaS testing on the cloud, including tools, issues, challenges, and needs.

In order to support the credibility of this study, after applying the mapping study method, we checked that all of the primary studies that are reported in the previous reviews mentioned above were located by the search process and either

complied with the inclusion criteria or were excluded based on the research inclusion/exclusion criteria.

2.1.3 Methodology

This section describes the systematic mapping methodology adopted in this study, by following the guidelines provided by [25], to provide an overview of empirical studies about cloud software testing methods, to answer the research questions, and reveal the current situation regarding the research topic. The steps in the mapping study method are documented below.

2.1.3.1 Mapping Study Questions

The major focus of this study, is to determine and classify the available information regarding functional and non-functional cloud testing methods, and the subject and attribute of the testing methods. The research questions addressed by this mapping study are:

Question 1: What types of functional and non-functional testing methods have been evaluated on/using cloud-based services?

Question 2: How were these testing methods applied, and what was being tested?

2.1.3.2 Search and Selection Process

The search and selection process summarized in Figure 2.1 is shown below.

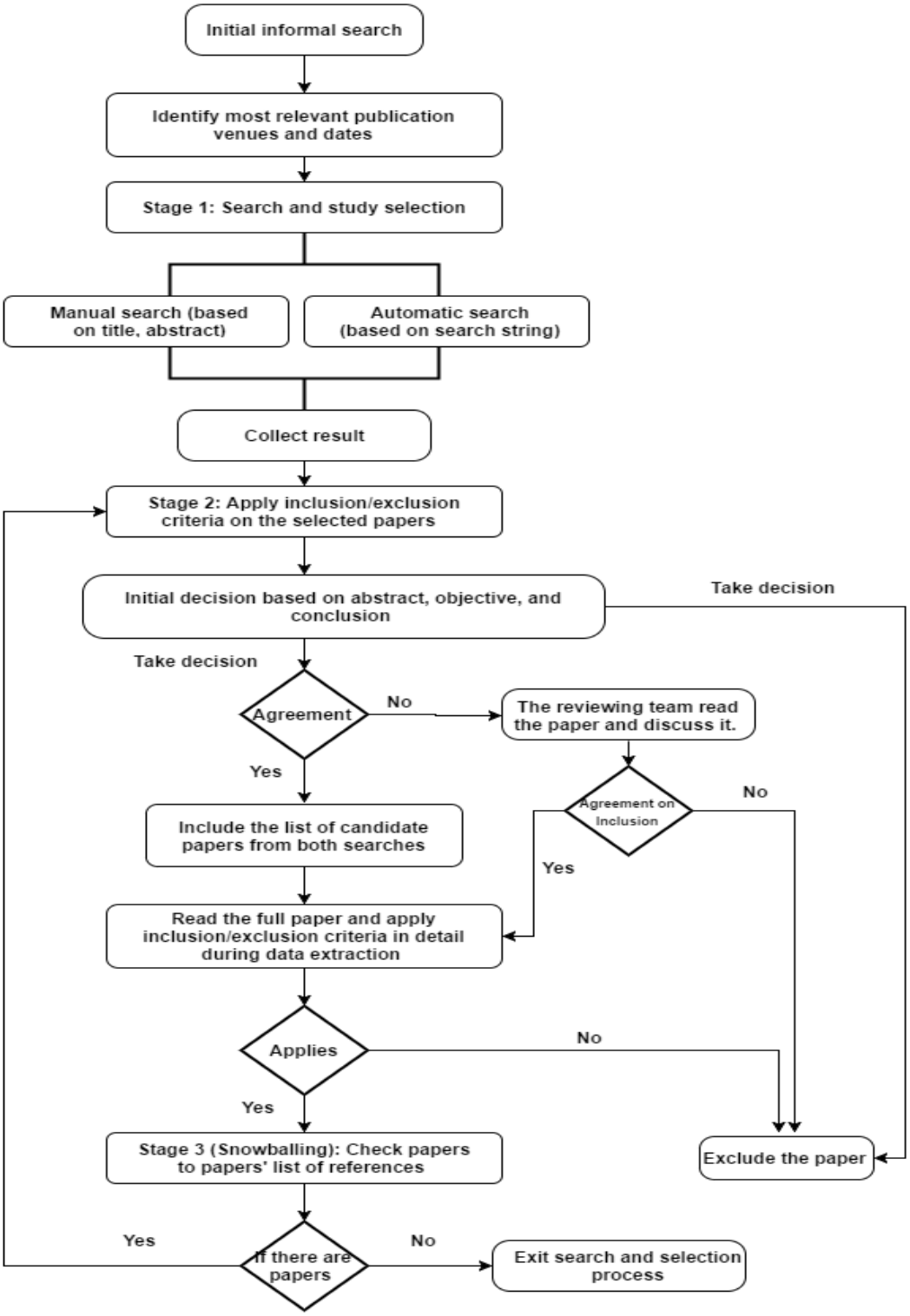

Figure 2.1: Search and selection process stages

An initial informal search was performed using ScienceDirect, ACM Digital Library, IEEE Xplore Digital Library, Springer, and Wiley, which identified publications' sources and dates for the topic of the study. This enabled us to select the relevant journals and conference proceedings, as well as the targeted publication period (2010-2015).

The search strategy included manual and automatic searches which were considered suitable after performing initial searches when devising the protocol. Relevant high-ranking* journals and conference proceedings were selected in the domains of software testing or cloud computing. Some high-ranking magazines such as IEEE Software were excluded, because no empirical studies related to cloud software testing methods were found during the initial search, and the search and selection process stages.

A manual search is more time-consuming than an automated search. It can give better completeness in terms of the number of relevant studies found, however [25]. Therefore, two manual searches were conducted: one in the peer-reviewed journals, and the other in the conference proceedings. An automated search of the International Conference on Software Engineering proceedings and the IEEE Cloud computing community conferences proceedings was conducted due to the huge number of papers that had been accepted and workshops that had been held in the conferences in each year. The search strings included the following: "(Cloud OR Cloud services) AND (Testing)" and (Testing Cloud services).

* The selection was based on the ISI web of knowledge/impact factor (Thomson Reuters), as well as well-respected scientific journals, conferences, and publishers.

Moreover, the snowballing method was used after the end of the second stage of the selection process in order to find more primary studies. The selected journals, proceedings, and additional sources are shown in Table 2.1.

Table 2.1: Selected journals, proceedings, additional sources

Source: Peer-reviewed Academic Journal	Publisher
Automated Software Engineering: An International Journal	Springer
Journal of Systems and Software	Elsevier
Information and Software Technology	Elsevier
ACM Transactions on Software Engineering and Methodology	ACM
Software Testing, Verification, and Reliability	Wiley
Software Quality Journal	Springer
Empirical Software Engineering	Springer
Software: Practice and Experience	Wiley
Journal of Software: Evolution and Process	Wiley
The Journal of Cloud Computing	Springer
IEEE Transactions on Software Engineering	IEEE
IEEE Transactions on Services Computing	IEEE
Source: Conference Proceedings	**Publisher**
International Symposium on Software Testing and Analysis	ACM
International Conference on Automated Software Engineering	IEEE/ACM
International Conference on Software Testing, Verification, and Validation	IEEE
International Symposium on Big Data and Cloud Computing Challenges	IEEE/ACM
International Conference on Software Engineering	IEEE/ACM
International Conference on Software Security and Reliability-Companion	IEEE
International Symposium on Cloud Computing	ACM
International Symposium on the Foundations of Software Engineering	ACM
International Symposium on Service Oriented System Engineering	IEEE
IEEE Cloud Computing Community Conference list Proceedings*	IEEE
Additional Sources: Edited Books	
Software Testing in the Cloud: Perspectives on an Emerging Discipline (Tilley S. and Parveen T., eds.)	

* http://cloudcomputing.ieee.org/conferences

2.1.3.3 Study Selection

Selection criteria were applied to ensure that only relevant literature was accepted in the mapping study, the criteria are listed in Table 2.2. The selection involved a three-stage process: (1) performing a screening activity – based on the paper title, abstract, and keywords; (2) reading the whole of paper/s by the lead researcher due to the possibility that the paper might be excluded in the data extraction stage; and (3) applying the snowballing technique on the accepted primary studies' list of references for the period 2010-2015, and repeating stages two and three on the targeted studies.

Table 2.2: Inclusion/exclusion criteria

Inclusion Criteria
Papers will be included if they are based on empirical research; experimental reports, case studies, or feasibility studies, with evidence that answer one or more research questions.
Several papers that report the same study will be included as a group.
Papers will be included if the publication date is 2010-2015, and writen in English.
Papers will be included if they describe testing methods used for cloud-based testing and provide an evaluation of the method used.
Exclusion Criteria
Letters, white papers, short papers with fewer than six pages, literature surveys, opinion papers, and reactions and responses to publications will be excluded.
Papers published in non-reviewed journals, conference proceedings, or magazines will be excluded.

2.1.3.4 Data Extraction

The aim of this stage was to produce proper systematic mapping by clustering the primary studies into mapping categories. All data had been extracted by Al-Said Ahmad, while the team of two reviewers performed extraction of the data from a

random sample of studies (see Appendix A for the reviewing team details). The review team held a meeting to reconcile the data with different points of view; options were evaluated and discussed, and a consensus on the right option was taken for each case. During the extraction stage, the full text of each paper was read, and the extracted data were stored in an independent spreadsheet (using Microsoft Excel). Further information (for some situations) that was considered useful was added as a new column in the data spreadsheet. The standard information extracted from each study was:

- Study identification (Study ID)
- Author/s
- Year of publication
- Paper title
- Publication title
- Keywords
- Publication type (journal, conference, book chapter).

Specific data extracted from each study included, possible values are noted below:

- Type of study (experiment, case study, feasibility study)
- Study aims and objectives (focus of study).
- Security testing options – vulnerability scan and assessment (e.g. fuzz test), security review, security audit, penetration test, or INP (If not provided).
- Scalability testing options – scalability testing, scaling-up (vertical), scaling-down (horizontal), or INP.
- Performance testing options – load testing, stress testing, endurance testing, or INP.
- Reliability testing options – regression testing, load testing, or INP.

- Model-based testing options – model-based security testing, model-based assessment, model-based performance/load testing, or INP.
- Mutation testing and injection-based testing options – mutation testing, fault injection, or INP.
- Functional testing options – functional testing or INP
- Test coverage options – percentage of coverage by (%) or INP.
- A number of experiments (examples) and case studies, with a brief description.
- Validation method options – simulation and modelling, cross-validation, qualitative data analysis, quantitative data analysis, or by a single example.
- Contribution facets – testing method or approach, testing framework, tool, or test case generation.
- Prototype study or not.

2.1.4 Result and Anlysis

The extracted data were analysed and structured to answer the research questions. An analysis of the primary studies and the data extracted relating to the research questions is provided in this section.

2.1.4.1 Overview of Result

In 75 research papers relating to evaluated testing methods using cloud-based services and resources, 69 primary studies were identified. The search was conducted using the method described in Section 2.1.3. As a result of this step, a total of 247 papers been obtained: 123 papers from conference proceedings, 36 from journals, 18 from additional resources, and 70 from applying the snowballing

technique. Table 2.3 shows the search results and the number of (included) papers remaining after each search and selection phase.

Table 2.3: Remaining studies after each search and selection step

Source		Initial search result	Phase 1	Phase 2
Automatic Search (621 results)		61 selected based on title, abstract, and keywords	24	20
Manual Search	**Academic Journals**	36	18	13
	Conference Proceedings	62	24	19
	Additional Resources	18	5	5
Snowballing		70	21	18
Total		247	92	**75**

Thus, 92 papers entered phase 1 of the search and selection process and 17 research papers failed to meet the inclusion/execution criteria during the data extraction process. Moreover, 70 research papers were found via snowballing and 48 papers were eliminated due to the exclusion criteria, as they were outside the date. Of the 75 research papers, 14 papers (19%) came from academic journals, 56 papers (75%) came from conference proceedings, and five papers (6%) were book chapters from the additional resources. Figure 2.2 shows the distribution of primary studies by year of publication.

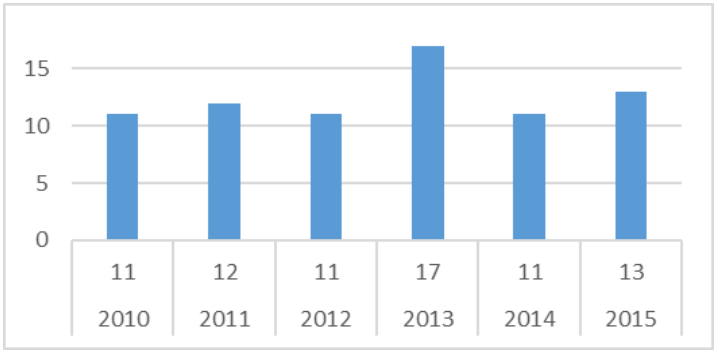

Figure 2.2: Distribution of primary studies by year of publication

Thirty (43%) primary studies used quantitative data analysis; however, 20 of these did not report specific statistical tests. Six (9%) studies used simulation and modelling techniques, three studies used cross-validation, one study used qualitative data analysis, and another study used both qualitative and quantitative data analysis. About 41% (28) of the primary studies evaluated their method using an example, and six of those studies provided some numerical data. Forty (58%) of the studies are feasibility studies, providing results about a limited scope and often partial implementation of the proposed approach or methodology, without considering a real-world scenario or complex software under test (SUT). There are 32 (47%) studies which describe a complete prototype implementation and there are 11 (16%) studies which present a single complete case study. There are only 18 (26%) studies which describe more extensive experiments (e.g. multiple case studies).

2.1.4.2 Results for Research Question 1

The primary studies were classified according to the testing methods, i.e., functional and non-functional. Table 2.4 shows the classification scheme that have been developed after applying the methodology described in Section 2.1.3, which was based on the used testing methods. The studies have been classified into seven main categories.

Table 2.4: Studies under testing methods

Category		Studies	#
Functional testing		S1, S2, S3, S4, S5, S7, S8, S11, S12, S14, S15, S17, S19, S20, S24, S27, S28, S30, S32, S34, S35, S36, S38, S42, S44, S48, S50, S51, S52, S54, S55, S56, S58, S64, S67, S68	36
Security testing	Vulnerability scan and assessment	S5, S7, S12, S21, S23, S25, S26, S33, S49, S46, S47, S53, S57	13
	Security review	S5, S7, S24, S26, S41, S47	6
	Security audit	S5, S7, S46	3
	Penetration test	S12, S16, S33	3
Scalability testing	Scalability testing	S3, S28, S37, S39, S48, S65, S66, S69	8
	Scaling-up	S13, S31, S42, S45, S53, S60, S62, S67	8
	Scaling-down	S9, S31, S42, S45, S13, S60, S62, S67	8
Performance testing	Load testing	S1, S3, S4, S6, S8, S9, S10, S11, S13, S14, S17, S18, S19, S20, S24, S28, S29, S31, S36, S37, S39, S43, S45, S40, S48, S50, S51, S59, S60, S61, S63, S65, S66, S67	34
	Stress testing	S9, S15, S18, S31	4
	Endurance testing	S9, S18, S31, S37, S45	5
Reliability testing	Regression testing	S4, S9, S30, S34, S42, S50	6
	Load testing	S1, S3, S11, S42, S48, S56, S59	7
Model-based testing	Assessment	S1, S8, S15, S27	4
	performance	S29	1
	security testing	S5, S25	2
Injection-based testing	Mutation testing	S25, S52	2
	Injection-based testing	S1, S3, S5, S6, S11, S19, S35, S36, S37, S40, S47, S53, S54, S56	14

Of the 69 studies, 36 (52%) studies involved functional testing methods, 55 (80%) studies involved non-functional testing methods, 14 studies focused only on functional testing, and 33 studies focused only on non-functional testing. Table 2.4 presents the studies included for this classification.

In the context of non-functional testing, 16 (23%) studies covered security testing, while 35 (51%) used one or more types of performance testing, 17 (25%) studies applied scalability testing methods, 12 (17%) studies used reliability testing, 13 studies applied mutation testing and injection-based testing to test non-functional features, and three studies applied a model-based technique to test a non-functional feature.

2.1.4.3 Results for Research Question 2

Based on the main purpose and key-wording of the primary studies the authors classified the studies into eight groups based on our view assessment: web services/app testing, mobile testing, Vulnerability and security configuration testing, benchmarking, testing SaaS, testing cloud services, large-scale testing, and other ways of application of testing.

During the classification, it was noted that some papers could be included in more than one group, so the decision was taken based on the consideration of the main purpose of the study. Studies that had a purpose that was not related to any of the other seven groups were labelled under other ways of application of testing. Table 2.5 shows the studies included for each group.

Table 2.5: Studies under main purpose

Category	Studies (S)	Total
Web services/app testing	S18, S21, S31, S43, S58, S59, S63, S65, S66, S67	10
Mobile testing	S23, S24, S44, S48, S53, S61, S68	7
Vulnerability and security configuration testing	S7, S12, S16, S25, S26, S33, S41, S46, S47, S49, S57	11
Benchmarking	S6, S9, S10, S13, S20, S36, S37, S40	8
Testing SaaS	S5, S11, S19, S28, S30, S38, S39, S42, S45, S64	10
Testing cloud services	S3, S8, S14, S15, S22, S27, S29, S32, S60, S62	10
Large-scale testing	S1, S2, S17, S35, S51, S52, S54	7
Other ways of application of testing	S4, S34, S50, S55, S56	6

2.1.4.3.1 Web services and web application testing

The feasibility study [41] examined the performance of web applications running on the three types of Amazon EC2 instances. Based on httpref (performance testing tool) PHP script workload and in-cloud load generator, the system stability was checked by generating load requests on the web server for a whole week. The study [42] presents a framework integrated with benchmarking and monitoring tools. A number of smaller-scale experiments are carried out to test performance and scalability of a web application using different instance types to measure the response time, compute units, and throughput. A framework for web security in the cloud [43], which examines vulnerability scanning for web applications, proposes a prototype TaaS framework for security testing, and is evaluated through experiments using 456 web applications, with 21,141 critical vulnerabilities detected. A prototype hybrid cloud testing platform called AGARIC is presented in [44] that uses both cloud resources and human resources to test web applications in a scalable way. Two experiments were conducted: one

with 10 computers and a local server to test a simulated application and another one using resources deployed in LAN and dokuwiki.org as the SUT.

ASTORIA [45], a prototype for automatic testing of performance and scalability on rich Internet applications, was tested with 1,000 virtual users in Amazon EC2. The study [46] presents an experiment for static testing the performance of web applications to measure their reliability. They use two VMs using VMware, and generated and executed test cases automatically by JMeter tool. Four studies present testing for SOA applications and web services using cloud-based resources. The first [47], is a feasibility study presenting a prototype to capture web service change at runtime by using functional regression testing to verify the selected services. The second [48] provides a cloud-based scalable PaaS for a dynamically chosen node in the IaaS layer. They use load testing for scaling-up and down in a case study of their previous work [49], WS-TaaS, testing the load capacity of three real web services. They simulate the service environment, applying 959 slices for deploying WS-TaaS on PlanetLab and using 50 nodes as the test node. The study [50] concerns cloud-based performance testing for web services. It reports prototype experiments in Amazon EC2 with 100 test tasks for three performance test methods, with each task assembled with 2 web services.

2.1.4.3.2 Mobile testing

Five studies present a TaaS framework for mobile testing, and two studies report testing of mobile applications. One study presents a TaaS framework for mobile development [51], evaluating the framework with one example and implementing

a web user interface using a VAADIN framework, Google App Engine application, and Jersey RESTful web services. A simulation-based mobile testing environment [52], emulating mobile devices using VMs and IaaS is evaluated using analytical techniques. A prototype mobile TaaS framework [53] is tested using a functional approach, comparing the result with two other test script generations. An automated TaaS is presented in [54], with a feasibility case study to evaluate it using private cloud services, with 9 mobile devices, 5 mobile applications and 84% test case coverage. The study [55] uses a prototype framework for load balancing implemented with OpenStack with 63 hosts and 400 requests, comparing the proposed method with other algorithms. A white-box automated security testing approach [56] for cloud-based Android apps is evaluated by an example run over 1,000 test cases using 100 parallel instances. The study [57] presents a testing approach with experiments evaluated on a combination of 1,000+ emulated instances and 10 actual devices.

2.1.4.3.3 Vulnerability and security configuration testing

A real-life case study [58] with six design stages is evaluated using a sequence of interviews. Study [59] presents a penetration TaaS, with two case studies that let POTASSIUM capture the exact SUT into a mirror and save it as a live snapshot. They ran a penetration test against the snapshot using a cluster of three different memory size Ubuntu VMs. An automated risk assessment framework (Nemesis) is presented in [60], involving vulnerability assessment by using their previous work [61]. To evaluate their approach, a cloud environment and its services are designed

using OpenStack, applying the framework on 10 IT products. A security testing approach is presented in [62] targeting two situations. First, they aim to determine the vulnerabilities of Ubuntu Server with the OpenStack node; second they aim to determine the vulnerabilities of cloud instances with different operating systems. A prototype framework for vulnerability assessment in cloud systems is presented in [61] and [63], with one example about developing an automated process for their proposed approach. Security validation as a service is presented in [64], with two hosts providing the proposed service to two midsize business processes, repeating the requests every 15 minutes for security validation. A vulnerability scan and assessment approach is presented in [65] with four test cases: two cases for security assessment from inside the cloud, and two from outside the cloud. The study [66] presents a prototype model-based security testing approach. The authors employed risk analysis to test the cloud environment, which is evaluated by one example using VMware's vCloud.

The study [67] presents an approach for detecting security vulnerabilities by checking for software updates and scanning virtual machines, with one experiment using Debian penetration suite, repeated 20 times. Another vulnerability assessment approach [68], applies three different scenarios to explain how the cloud affects the security vulnerability. A model-driven approach is shown in [69] to facilitate the creation of security configurations. The approach is assessed by applying it to a model developed using the Oryx tool.

2.1.4.3.4 Benchmarking

The study [70] presents a benchmarking-as-a-service framework that automatically scales the injection load platform. Three experiment scenarios were performed, with two SUTs selected to test in these scenarios. The study presented in two research papers [71], [72] introduces performance and scalability testing of SaaS using IaaS. The experiments measure the performance of two SaaS applications using three public clouds, and three private clouds, evaluating both the scaling up and out in Amazon EC2, and scaling out in Emulab and Open Cirrus. The study [73] presents a modelling framework (ROAR) for automated cloud resource allocation, optimisation, and benchmarking. In two experiments using Amazon and Google clouds, they use the ROAR to deploy multi-tier applications to cloud providers and an auto-scaling engine. The study [74] presents C-MART, a benchmark application emulating a web application running in the cloud. C-MART is run against data-centre benchmarks comparing the results. The study [75] proposes a cloud-based load testing model for cloud infrastructure. The validation involved two experiments for benchmarking as a service using two e-commerce systems (TPC-W), one with MySQL and the other with NoSQL. The study [76] presents a toolset called DS-Bench, which operates through benchmarks and fault injectors that simulate the overload in system resources, aiming to measure dependability.

A framework is presented in [77] to facilitate performance comparisons of cloud data serving systems, using 6 server machines to verify the scalability of YCSB.

They run one experiment with PNUTS on a 47 server clusters with a database that contains 120 million records. A benchmark for virtualized and cloud environments is presented in the study [78], they run several experiments using Libvirt, oVirt, Sar, Faban, KVM, and Collectd.

2.1.4.3.5 Testing SaaS

In the context of testing SaaS, [79] introduces a novel model-driven security engineering approach for multi-tenant SaaS applications. To evaluate the proposed approach, they applied it to seven open-source web-based applications developed using ASP.Net. The study [80] presents Trio, an open-source Java prototype topology robustness indicator that simulates failure sequences. By using a domain-specific language (CloudML), Trio is used to evaluate the robustness of various topologies through a number of experiments. The study [81] presents an approach to automate performance testing of cloud applications and a prototype based on load-testing tools and using IBM's WAIT expert system. Two experiments were conducted: one to evaluate the overhead using JPetStore and IBM WebSphere Portal applications, the other to evaluate the productivity of the approach by injecting three common performance issues in JPetStore.

TaaS with tools are presented in [82], which describes a single case study with 100% test coverage. Using the OrangeHRM (SaaS) application with two functional features, and two black-box test methods, system-level test cases have been designed for each feature. The prototype study [83] aims to improve the test effectiveness and efficiency of SaaS using a regression testing approach with 61%-

72% test coverage. The study reports one case study using two versions of an industrial application. They generate test cases from the requirements scenarios and execute each test case manually. A prototype testing approach to detect scalability bottlenecks in NoSQL schemas is presented in [84]. Concurrent writes are generated by running a servlet on Google App Engine. A case study uses an article-oriented scenario, creating one single article, and a series of 20, 100, 500, and 1,000 write requests runs against the single article. The study [85] presents a code generation tool for automated performance testing of distributed applications in IaaS called Expertus. Experiments were performed using three SaaS solutions deployed on five IaaS solutions.

A prototype approach to support SaaS continuous testing and policy enforcement is presented in [86]. The study describes one case study using test cases generated from Metadata. The test cases are ranked based on their importance, WebStra's framework ranking, and their history. They establish a test oracle by voting and automatically analyse the oracle using statistical techniques. The study [87] presents a testing model that evaluates SaaS performance and analyses scalability in the cloud. A case study is reported using Amazon EC2 with four load configuration scenarios. An automated integration testing approach of SaaS is introduced in [88]. A prototype of unit testing framework is described using Windows Azure and Visual Studio 2010.

2.1.4.3.6 Testing Cloud Services

The study [89] presents a tool for automated quality of service and scalability analysis for system reliability testing using load variation and fault injection. Experiments were performed to evaluate the proposed tool using seven user loads to measure the scalability and the quality of the SUT. A study presented in two research papers [90], [91] uses integration testing of data-centric and event-based dynamic service compositions. Four distributed performance test experiments were run on a single virtual machine using Ubuntu Linux. A testing framework for test scripts and test case generation that measures service performance, called CLTF, is presented in [92]. The authors applied the framework to over 1,300 realistic cloud services from 50 projects collected from the enterprise private PaaS cloud.

A prototype model-based assessment approach is presented in [93]. They evaluated the proposed approach with a case study simulating system prototypes in the face of hostile environment conditions. Another study [94] presents a cloud service selection model through a set of experiments. They used 59 real cloud services to do real-time performance evaluation. The study [95] presents a prototype testing framework for cloud platforms and infrastructures. To evaluate their framework a case study was conducted with 18 Google App Engine test cases. A prototype platform is presented in [96] for testing services and users. The platform enables the setting up of unit testing by selecting the most suitable unit testing method and cloud service, test case generation, execution, and reporting

testing result in an automatic way. Another prototype framework for cloud services test cases generation is introduced in [97], with one experiment. The system is separated into a web service semantics side that generates test cases from source code and transmits these to the UDDI side that allows the users to discover cloud services. Research paper [98] presents a simulated cloud service based testing approach. The study proposes a solution for testing and quality estimation for both bottleneck detection and fault diagnosis using an offline testing technique. The study [15] presents a scalability testing approach to model the performance for cloud-based services at different abstraction levels. The paper constructs preliminary models for IaaS, and the benchmark program (SaaS) on the cloud using Amazon resources and services.

2.1.4.3.7 Large-scale Testing

The study [99] presents a model-based testing approach using a local cloud to test the global properties of a large-scale system. An experiment was conducted on two clusters of 32 nodes to validate the functionality of two popular clouds' open-source distributed hash tables, data insertion, and retrieval. An analysis of crowdsourcing testing methods for a large-scale system by using INP is presented in [100]. Three experiments are presented: to determine the min-time test case combinations, to compare the proposed approach with the performance of CPLEX ILP formulation, and to evaluate the performance of the proposed testing approach.

The study [101] presents peer-to-peer load testing approaches to isolate bottleneck problems in a large-scale system. The experiments used load testing validate performance having point-to-point connection between the test driver and the SUT, or using tools that provide a test driver to allow submission of operations based on load type, with one machine to simulate the SUT and five others to simulate the clients. The study [102] presents an investigation of cloud computing to facilitate the testing of large-scale software. They evaluate the proposed mutation functional testing using a case study on Google Chrome and Amazon EC2 with 820 implemented mutations.

The study [103] presents a case study of resource management infrastructure to enable integration testing of distributed real-time and embedded system applications. They used a modelling tool (CUTS) to evaluate an infrastructure-level system (RACE) scenario in the Emulab test cloud. A study is reported in three research papers about D-Cloud [31], [104], [105], a simulated Eucalyptus-based testing environment for large-scale distributed systems. The authors apply D-Cloud to two real systems: a highly available server system and RI2N. The study [106] is a feasibility study that introduces a framework for testing the IaaS-based delivery model, which is evaluated by using FaultVM and D-Cloud.

2.1.4.3.8 Other ways of testing

The study [107] presents an automated verification approach for virtual machine patches with three stages of experiments. An approach to manage, compose and test services on the cloud is presented in [108]. The study provides limited data on

the results. Test case generation using JUnit is presented in [109], with three series of experiments. They determined the performance of the JUnit test execution using one machine, then they used HadoopUnit to coordinate testing on four nodes in a cluster, finally they tested the reduction of map tasks by increasing the workload of each map task. A simulation test case generation using parallel symbolic execution is presented in [110] based on MC/DC test cases and suite generation with six case examples.

Cloud9 [111], [112] is a prototype platform for automated testing of real-world applications that run on Amazon EC2, private clusters, and multi-core machines. 5 case studies are reported, using different operating systems and simulated services. Scalability Explorer, an automated framework for scalability testing is presented in [113], introducing scalability testing as TaaS through one experiment to evaluate a web service-based distributed matrix multiplication system hosted on Amazon EC2.

2.2 Additional Literature Review Update

The mapping study [20] from 2010-2015 provides the related work for the thesis focus area. In of the covered papers 17 studies have been found, which relate to scalability testing in/on the cloud/cloud services. The majority of these papers performed scalability testing for Platform as a service (PaaS), Infrastructure as a service (IaaS), mobile applications, or web applications. Only five studies focus on

scalability testing on cloud-based software services. Considering this and due to the importance of scalability analysis of cloud-based software services from technical and business perspectives, the decision has been made to focus the research of this PhD project on the area of testing and measuring the scalability performance of cloud-based software services.

To ensure all relevant work (published between January 2016 and March 2019) is considered in this thesis, an additional review of existing scalability testing and measurements of cloud-based software services was undertaken.

As mentioned in Chapter 1, the performance of cloud-based software services depends on three interrelated aspects; scalability, elasticity and efficiency [11], [12]. Both elasticity and efficiency aspects depend on a sufficient level of scalability performance. Most studies related to performance measurement and testing focus on quantifying and measuring the elasticity of cloud services. A related systematic literature review [13] covers cloud performance assessments and metrics in terms of scalability, elasticity, and efficiency. They highlight of the key findings are that most of the reviewed papers focus on elasticity, and regarding scalability, they report that the papers were either early and preliminary results or initial ideas of research students. The review [13] provides the definitions of the key performance aspects, such as scalability, elasticity and efficiency, which have been adopted in this thesis (see Section 1.1).

The majority of the studies focus on measuring the elasticity of cloud services from a technical perspective [12], [17], [87], [114]–[118]. For example, Herbst et al. [12]

sets a number of key concepts that allows measuring cloud service elasticity in technical term, such as the quantity and time extents for periods of time when the service provision is either below or above what is required by the service demand. Elasticity measures are defined by [12], [114] as: the timeshares and average time lengths in under-provisioned and over-provisioned states. Further elaboration [115] that extended the above metrics introduced other factors and ways such as reconfiguration time, functions of resource inaccuracy, and scalability. This concept will be discussed with more details in Chapter 3.

From the utility-oriented perspective of measuring and quantifying scalability, note the work of Hwang et al. [15], [18]. Their production-driven scalability metric includes the measurement of a quality-of-service (QoS) and the cost of that service, in addition to the performance metric from a technical perspective [15], [18]. This approach is useful from a utility perspective, as it depends on multiple facets of the system (including cost measures). However, this approach cannot easily provide useful and specific information in terms of contribution of system components to scalability in a technical perspective.

Technically-oriented measurements or metrics for cloud-based software scalability research are limited. Such as [12] provides a technical scalability metric, however, this is a rather elasticity driven metric which measures the sum of over- and under-provisioned resources over the total length of time of service provision. While, Jayasinghe et al. [71], [72] provides a technical scalability measure in terms of throughput and CPU utilization of the virtual machines, but the work does not

provide a metric or measure. The work focuses on presenting an experimental analysis of performance variations on three public cloud platforms (EC2, Open Cirus, and Emulab) using two cloud applications (RUBBoS and/or Cloudstone), and three private clouds that have been built using the three mainstream hypervisors (XEN, KVM and CVM). Jamal et al. [119] describe practical measurements of systems throughput with and without multiple virtual machines (VMs), without clearly formulating specific measurements or metric of scalability. Gao et al. [87] evaluate software as services (SaaS) performance and scalability from the capacity of the system perspective, by using the system load and capacity as measurements for scalability, a case study using a sample of Java-based program been reported using Amazon EC2. Another recent work [120] focuses on building a model that helps to measure and compare different deployment configurations in terms of costs, capacity, and elasticity by evaluating the proposed metrics using CloudStore on EC2, they identified the scalability in terms of the number of simultaneously simulated users as a current limitation. Brataas et al. [121] offered two scalability metrics, one based on the relationship between the capacity of cloud software services and its use of cloud resources; the second is the cost scalability metric function that replaces cloud resources with cost, in order to demonstrate the metrics, they used CloudStore application hosted in Amazon EC2 with different configurations. In an earlier work, [122] provides a theoretical framework of scalability for mobile multi-agent systems, however, which remains limited to theory and modelling results.

In terms of comparisons, [71], [72] compared the performance and scalability of two applications on three public clouds (Amazon, Open Cirrus, and Emulab), and three private clouds. As mentioned above the comparison were based on CPU utilization and throughput without providing any metric or measure. Similarly, Hwang et al. [15], [18] introduces a set of experiments involving five benchmarks, three clouds, and set of different workload generators. Only three benchmarks were considered for scalability measurements, the comparison was based on the scaling scenarios, and what the effect on performance and scalability. Gao et al. [87] run the same experiments in two different AWS EC2 instance types, one with load balancing and one without. While Vasar et al. [42] introduces a framework for testing web application scalability on the cloud, run the same experiments settings to measure response time on three different EC2 instance types.

In terms of fault injection, related survey studies [123], [124] show that most of the work is focused on using fault injection to measure the fault tolerance in cloud computing. The majority of the studies use the technique of injecting the fault on IaaS and PaaS levels [125]–[128], or by introducing a test environment system that injects faults into hardware devices or VMs levels [104]. However, there have been some studies that address the fault injection technique on cloud applications level. These studies describe either prototypes or the use of this technique to build fault detection and diagnosis models. Herscheid et al. [129] proposed a draft architecture for "fault injection as a service" within the OpenStack, the implementation of the service itself is a work in progress. Ye et al. [130] proposed a fault injection framework for artificial intelligence applications in container-based

clouds, in order to observe the fault behaviour and interference phenomenon, however, the work focuses on presenting fault detection models that can detect the injected faults.

2.3 Discussion

Before the primary area of research could be identified, a full review of the empirical studies of cloud software testing had to be conducted. Following this the next task was to identify the empirical practice in this area, to help to identify research gaps and further research opportunities.

The mapping study reported in Section 2.1, has discussed 69 unique primary studies on software cloud testing reported in 75 research papers. The mapping study reported here presents a state-of-the-art analysis of existing cloud-based testing methods that were experimentally evaluated during the period 2010-2015. This was done methodically by following a well-defined mapping study protocol.

It is possible that not all relevant studies were identified in the mapping study, however the considered review papers were used to validate the sufficient coverage of relevant primary studies. Multiple reviewers have been used to check the quality of the extracted data.

The majority of the reviewed studies present only preliminary results, often describing an example of the software cloud-based testing methods or a simple application experiment to evaluate the proposed approach. Many of the

considered studies rely on limited scope or relatively simple implementations and case studies. Only a minority of the studies used quantitative analysis combined with rigorous statistical tests. The considered studies spread relatively evenly across the testing topic categories that have been used in this study. Many of the studies present early work and results that their authors expect to lead to further more extensive studies. Often the assessment of the proposed solutions is based on a single experiment. These indicate growing interest across the field of cloud-related testing and the potential for much more research to follow the early results.

Following the mapping study, the decision was made to focus on the research area of testing and measuring the scalability of cloud-based software services. Based on the result of the above discussed outcome from the mapping study, and the importance of scalability analysis of such services from technical and business perspectives. Although, the number of studies that are working on scalability testing (subsection) 17 as shown in Table 2.4, most of these studies were related to scalability testing in/on the cloud/cloud services. The majority of these papers performed scalability testing for Platform as a service (PaaS), Infrastructure as a service (IaaS), mobile applications, or web applications. Also, as shown in Table 2.5 only five studies focus on testing on cloud-based software services or SaaS, however, only two studies were focusing on scalability/performance testing, those studies were discussed further in Section 2.2. The first, Jayasinghe et al. [71], [72] provides a technical scalability measure in terms of throughput and CPU utilization of the virtual machines, but the work does not provide a metric or measure. The second study, Gao et al. [87] evaluate SaaS performance and

scalability from the capacity of the system perspective, by using the system load and capacity as measurements for scalability. Both of the studies did not provide full technical analysis or specific metric of scalability.

According to systematic review [13] reports that most of the studies are based in literature, there are only a few research works (e.g. project reports, MSc dissertations) which attempt to address the assessment of the technical scalability of cloud-based software services. An additional search of related studies to technical scalability analysis of cloud-based services shows that most work presents early results and their authors expect to carry their work forward on to more extensive studies, or measuring the elasticity of cloud software services from a technical perspective. On the other hand, an alternative utility-oriented approaches used in the literature for the measurement of the scalability of cloud software services.

The results of this mapping study and the additional literature review serve as a basis for the programme of research reported in this thesis. In this section, the implications of the literature review have been discussed.

2.4 Chapter Summary

The systematic mapping study and additional literature review reported in this chapter explored the current empirical practice in the area of cloud software testing, focusing on the area of testing the scalability of cloud-based software

services. The findings of this chapter aim to address thesis objective number 1 (see Section 1.2). The evidence gathered indicates that most studies in cloud software testing methods present an early stage practice or evaluate their methodologies using simple examples. Considering the state of the current research on scalability evaluation on cloud-based software services and the importance of this it was concluded to focus this PhD research project on this area of research.

Chapter 3 Methodology

In this chapter, introducing software development life cycle in cloud computing, the stakeholders, and the importance of scalability testing for cloud-based software services. Furthermore, details of the scalability test plan, testing environments and resources are reported. The scalability test plan is based on the IEEE 829 standards. The test plans were developed to test the scalability delivery of cloud-based software services as described in Section 3.2. In addition to the explanation of the testing environment, the resources relied on to test the scalability of cloud-based systems are outlined in Section 3.3. The cloud elasticity concept is described in Section 3.4. Finally, Section 3.5 summarises this chapter.

3.1 Introduction

Software testing and measurement is part of the software development life cycle (SDLS), this life cycle is a systematic process in order to produce a software system, this process involves systematic steps to ensure the quality of the software build. SDLS should consist of a well detailed and clear plan(s) which explains how to analyse, design, implement, test and maintain of the software system build.

Cloud computing software services are based on functional and non-functional requirements as well as the traditional applications. However, cloud application services developments have more additional non-functional requirements (i.e. multi-tenancy, on-demand self-service, elasticity, and scalability) that are key for cloud computing software service delivery [131]. Therefore, the SDLS for cloud-based applications should incorporate the satisfaction cloud consumers but also cloud services providers. These requirements are not only driven from customers' perspective for functionality purposes but other cloud-specific non-functional requirements, such as elastic scaling, and multi-tenancy. Therefore, software engineers should be aware of these cloud non-functional requirements and incorporate it into the SDLS.

According to some studies that focus on SDLS in cloud computing [131]–[133] following the Iterative and Incremental development model, such as Agile, Incremental, Spiral or V-model, which incorporates the normal SDLS processes; requirements, analysis, design, implementation, testing, and maintenance. Furthermore, the studies agree to integrate the cloud-specific non-functional throughout the whole process and considers cloud computing's specific nature.

Software developers, testers, and cloud service providers should work together in order to meet cloud-specific requirements and functionality of the software system build. This includes planning; analysis of the requirements to include both functional and non-functional; designing the architecture of the software to fit with REST (Representational State Transfer) and SOA (Service-oriented architecture);

testing for both traditional and cloud-specific attributes (elasticity, scalability, efficiency … etc.); and maintaining the software to meet any new cloud requirements. Figure 3.1 illustrates the stages of iterative-process model proposed by [131], which include traditional SDLS process in addition to cloud-specific process.

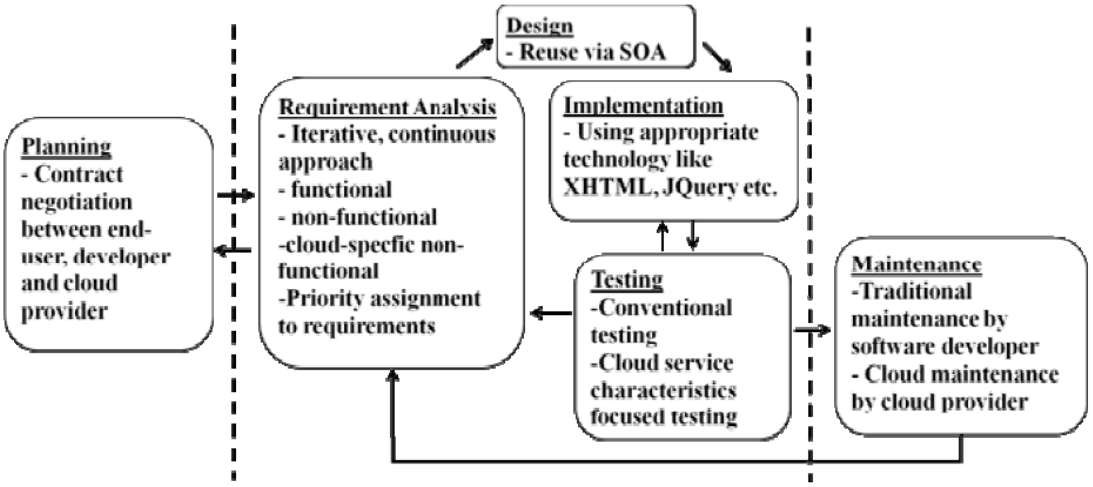

Figure 3.1: Stages in the Cloud SDLS model [131]

This thesis focuses on testing one of the cloud services requirements; scalability. As mentioned in Chapter 1, this thesis adopts the following definition of scalability. It is the ability of the cloud layer to increase the capacity of the software service delivery by expanding the quantity of software service that is provided [13].

Scalability testing is used to measure an application's ability to scale and increase the system performance by adding more resources. Testing for scalability is an extension of performance testing. The purpose of scalability testing is to identify major workload dynamics that may reduce capability, which can hinder the

scalability of the application. Scalability testing is important for developing scalable applications, measuring performance, and ensuring that the application can handle future growth needs [134], [7].

Scalability is one of the major benefits offered by the cloud, especially the dynamic scaling service for cloud-based applications [134], which makes the cloud software services more elastic. Furthermore, cloud computing offers features to provide the support for cloud-based software services to be more scalable, such as Auto-Scaling and Load-Balancing, which enables such services to deal with sudden workload changes by adding or dropping instance(s) at runtime. There is an increased need for scalability testing and measurement to support the delivery, availability and productivity of the services and on-demand resources. This will provide an important foundation for future optimisation and support the service level agreement (SLA) compliant quality of delivery of these services, especially in the context of rapidly expanding the quantity of service delivery [6].

To measure the scalability of any application, a scalability test plan must be in place to collect and monitor the indicators from the scaling behaviour of the system under test (SUT). To achieve this, further investigation on the scalability of cloud-based software services needs to be conducted to select the right measurements and testing of such services that allows those measurements to be interpreted by the right metrics. To support the right practice of collecting the scalability measurements, the right test plan should specify the right testing environment: workload generation, targeted software service(s), and cloud

platforms and services. Therefore, the scalability test plan is described in detail. Moreover, all the types of resources, software systems and load generators used to deliver the practical part of this thesis are described. Figure 3.2 illustrates the research methodology.

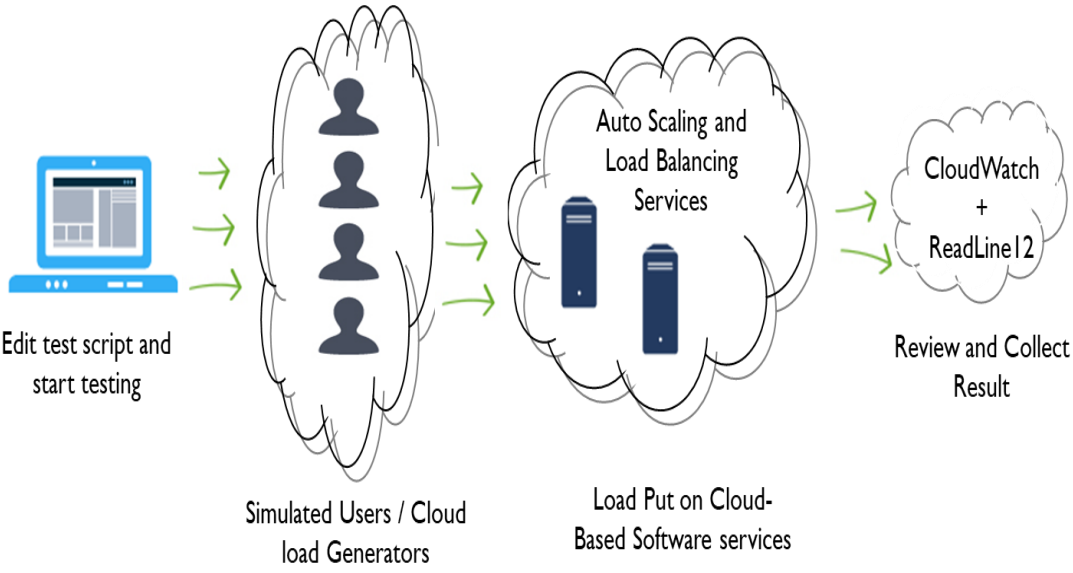

Figure 3.2: Research methodology for testing the scalability of cloud-based services

3.2 Test Plan

One of the main focuses of this test plan is to ensure that the test is performed within the guidelines for process, design, approach and specifications as defined by IEEE 829 standards [135] for a software test plan (see Appendix B). However, in this thesis, some points in the standards (i.e. features not to be tested, schedule, approvals, and staffing and training needs) are not applicable due to the nature of this project, i.e. there are no team members, and the work focuses on testing the scalability feature only. The test plan is described below.

3.2.1 Test Plan Identifier

Test plan for scalability performance TP_1.0 (this refers to the ID of each test plan, and the plan version).

3.2.2 Introduction

The test plan outlines the operational aspects of executing the scalability testing strategy to collect the right measurements of performance. The plan investigates the scalability performance of real cloud-based software services hosted in the public cloud environment (see Section 3.3). This plan was developed to define the tools to be used throughout the scalability testing process and to define environmental needs and how the tests will be conducted.

3.2.3 Test Items

The software services (OrangeHRM and Mediawiki) to be tested include a graphical user interface (GUI) website to measure the scalability performance of those services. The software services should be hosted in a public cloud platform (Amazon AWS and Microsoft Azure), and both Auto Scaling and Load Balancing services should be connected to the service.

3.2.4 Approach (Test Script and Demand Scenarios)

Tests will be conducted using a demand generator to generate workload demands (demand scenarios) on the targeted systems. Demand scenarios may follow certain patterns expected to test the scalability of the system in specific ways. There are three kinds of demand patterns that appear as natural and typical choices. The first scenario is a steady increase followed by a steady decrease in the demand with a set level of the peak. The second scenario is a stepped increase and decrease, again with a set peak level of demand. The final, third scenario is a varied stepped increase and decrease scenario with a set peak level of demand. These demand scenarios will be discussed in details later on Chapter 4.

There are three kinds of demand patterns that appear as natural and typical choices, following the patterns recommended by Fehling et al. [136], which can follow static, periodic, once-in-a-lifetime, unpredictable, or continuously changing workload patterns. Any demand scenario or workload must represent real customer workload. So in this thesis, we have adopted and followed those patterns, and developed our own versions of these recommended patterns. Here the first scenario is a steady increase followed by a steady decrease in the demand with a set level of the peak, this scenario follows the static workload pattern, which is suitable for private cloud-based applications of small and medium-sized companies, these systems are usually used internally by employees or a small user group [136]. The second scenario follows the pattern of periodic workload, this workload is represented by stepped increase and decrease of workload/demand,

these kind scenarios are suitable for cloud-based software services that follow growing and changing demand with peaks. This is important to show how the scalability of cloud-based software services is adjusted automatically to the rate at which growing or changing happened [136]. The Third scenario follows the pattern of unpredictable workload, this workload is represented by varied stepped increase and decrease of workload/demand, these suitable for cloud-based software services that follow growing and changing demand with random peaks. This is important to show how the scalability of cloud-based software services is handling the unpredictable random peaks [136].

In this thesis, we report the behaviour of the service software in response to the most basic service request, i.e. a generic HTTP request. The JMeter script allows us to send an HTTP/HTTPS request to the targeted application, and parses HTML files for images and other embedded resources (i.e. applets, stylesheets (CSS), external scripts, frames, iframes, background images...etc.), and sends HTTP retrieval requests [137]. For our purposes it was sufficient to issue the simplest HTTP Request, i.e. logging in to the software service and getting in response an acceptance of the login request.

To generate the workload demand scenarios using JMeter, *Thread Group* is used to generate the demand volumes for the first scenario. Then *jp@gc – Stepping Thread Group* is used to generate the demand volumes for the second scenario, and finally, *jp@gc - Ultimate Thread Group* is used to generate the demand volumes for the third scenario. Each scenario varies the volume of demand and we used experiments

with four maximal demand sizes: 100, 200, 400 and 800 service requests in total. An example of using *jp@gc – Stepping Thread Group* to generate 800 service requests is shown in Figure 3.3.

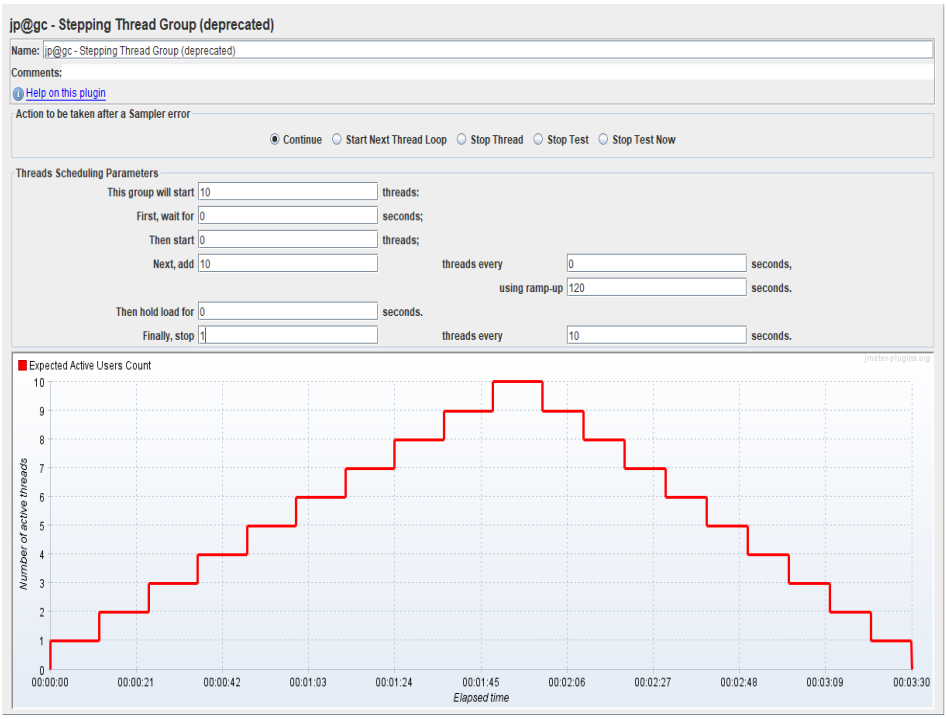

Figure 3.3: jp@gc – Stepping Thread Group (800 service requests) example.

To ensure the repeatability of the test, RedLine13 services are used, which allows JMeter test scripts to be deployed easily and the tests to be repeated without the need to reset the test parameters. Each test is repeated 20 times. The service requests consist of an HTTP request to all pages and links of the software system(s) by gaining login access using the following steps via the Apache JMeter:

- Path = /.
- Method = GET.
- Parameters = username, password, and login button.

Figure 3.4 illustrates the test approach, including the developed tests using JMeter (test script), prepare to test (i.e. preparing the environment to run the tests), run tests, and review the results.

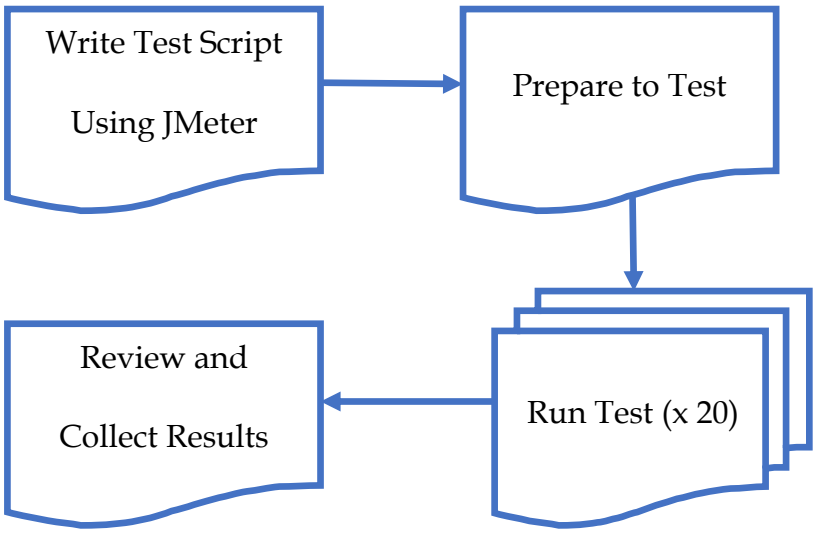

Figure 3.4: Test approach

3.2.5 Item Pass/Fail Criteria

Each test must have completed without any errors with a success rate must be 100% (the success rate determined for the overall test plan). That means all virtual users in one test must complete the test successfully in the allotted time for each test, i.e. if we assigned 800 virtual users/demand to hit the system in an x minute(s), we expect that all 800 will successfully finish.

Any failed test should be repeated and excluded from the results, however, during the experiments presented in this thesis, a very small number (4 out of around 900 tests) of failed test were detected, which were repeated.

3.2.6 Suspension Criteria

The test plan should be paused in the case of test failure or a success rate value less than 100%.

3.2.7 Test Deliverables

Upon completion, the experimental data will be collected through both the Redline13 service and Amazon's CloudWatch or Azure Monitor services. All the test results will be saved and reviewed in the test summary reports, and the average and standard deviations for all test runs must be included. The researcher's responsibility is to monitor each test at runtime, and collect and evaluate the results after each test has finished.

3.2.8 Testing Tasks

The following activities must be completed:

- Test plan in place.
- Testing environment should be ready (including test data, test logins).
- Run all tests, and deliver test results including average and standard deviations for each test x 20 times, for all tests, and prepare the test summary reports. To ensure the results are statistically significant (i.e. tests should be repeated 20 times to say that the result can be considered as

benchmark data; if the tests rely on collecting a performance indicator, the value of one test should be obtained and compared with previous tests).

3.2.9 Environmental Needs

The following represent the essential environmental needs:

- Software application(s) must be uploaded/hosted and functioning well on public cloud platform(s): Amazon AWS and/or Microsoft Azure.
- Both Auto Scaling and Load Balancing services must be attached to the software(s) instances.
- Cloud-based monitoring services must be connected and customised to report any failure from within the cloud environments.
- The JMeter script(s) for each scenario must be ready and uploaded in the RedLine13 service.
- Each test must be saved under a unique ID.

3.3 Cloud Platforms, Services, Software, and Load Generators

This section will discuss the resources that were used to complete the experiments explained in this thesis, i.e. cloud platforms, services, software, and load generators.

3.3.1 Amazon Elastic Compute Cloud (EC2)

A web-based interface service that provides resizable compute capacity in the Amazon Web Services (AWS) cloud, EC2 is designed to accommodate web-scale cloud computing for users [138], available from https://console.aws.amazon.com. This allows the researchers to run the selected software services in the cloud and provide the capacity for those services, and allows complete control of the computing resources. It provides the tools to take control over you cloud account and instances. Amazon EC2 cloud pricing system is based on pay-on-use model, as in this thesis, EU (London) "eu-west-2b" is the region/pricing that been used to deliver all the experiments.

3.3.2 Microsoft Azure

Microsoft Azure is a web-based portal service that allows building, testing, deploying and managing application services over the Microsoft cloud available from https://portal.azure.com/. It provides measured services to run software services in the cloud, and supports such services with all the infrastructure and management tools that are required while also ensuring end-to-end services to support that [139]. Azure cloud pricing system is based on pay-on-use model, as in this thesis, UK South is the region/pricing that been used to deliver all the Azure experiments.

3.3.3 Auto Scaling Services

Auto Scaling services are services that help to ensure that an application has the proper number of instances dynamically, can handle the workload during runtime [140], and provide an automatic way to manage the application's capacity [141]. These services enable the user to monitor the application's performance and required resources based on user policies and conditions [142]. The experiments discussed in this thesis rely on the Auto Scaling services from Amazon EC2 and Microsoft Azure to complete the experiments outlined in this study.

This service can be linked through the cloud console management portal for both EC2 and Azure, while you set the service up; there are some parameter we need to consider:

- Max capacity for the auto-scaling group: The maximum number of instances that the Auto Scaling Group should have at any time.
- Launch Configuration: is an instance configuration template that an Auto Scaling group uses to launch EC2 instances (Include the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping).
- Scaling Policies: A scaling policy is a set of instructions for making such adjustments in response to an Amazon CloudWatch/Azure Monitor alarm that have been assigned to it. In this thesis, we relied on CPU Utilization.

- Attach the Load-balancers to the Auto Scaling group.

3.3.4 Elastic Load Balancing

Elastic Load Balancing "*automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances*" [143]. This service helps to achieve fault tolerance for software services by ensuring scalability and performance. A Load Balancing application is designed for load balancing HTTP, TCP and other traffic where better performance is required. In this thesis we relied on an Application Load Balancer attached to the Auto Scaling group; which deals with HTTP and HTTPS traffic.

3.3.5 AWS CloudWatch and Azure Monitor

Both Azure Monitor and AWS CloudWatch monitoring services are used to monitor and collect the experimental data at runtime. This services can be attached with application's instance(s), to check on the instance health parameters from the console management portal.

3.3.6 Cloud-based Software Services and Taxonomy

To validate the proposed metrics (see Chapters 4 and 5), OrangeHRM and Mediawiki are used as cloud-based software services. Both are available as open-

source application. These cloud-based services are also available as SaaS that can be rented via the Amazon and Azure web marketplaces.

OrangeHRM is an open-source human resource (HR) management software system available from https://www.orangehrm.com, is implemented using PHP and MySQL. This service has been optimized to fit cloud hosting, and the architecture is based on offering a scalable HR solution [122]. It considers the most popular open source human resource management (HRM) software in the world, used by more than three and half million users around the world [144]. The application architecture supports REST (Representational State Transfer) caching [145] in order to improve performance; by caching the data and the code, which will reduce the amount of time required to execute each HTTP request and therefor reducing the CPU usage [146].

Mediawiki is an open-source wiki software system available from https://www.mediawiki.org. Is an "*extremely powerful, scalable software and a feature-rich wiki implementation that uses PHP to process and display data stored in a database*"[147]. MediaWiki application is written in PHP, uses MySQL database to store data, uses REST and RESTBase [148] for cache the data and the code to improve performance.

In addition to the REST nature of the chosen software services, which are highly adopted by cloud and application providers, and the most frequently used software services. The taxonomy of any chosen services represents widely used application categories, the categories are sorted into groups based on the

functional area (deployment method) or vertical industry (business) [149]. The categories of application's taxonomy were based on 100 different fields of data including: on premise and cloud customers; cloud platform infrastructure providers; cloud subscriptions; hybrid cloud; and 100,000+ customer adoptions of applications. To achieve the aims of this thesis we focus on the functional taxonomy of the chosen software services.

The first software service (OrangeHRM) from the functional area is a HR management service, so this is placed under the category of Enterprise applications, and more specifically under the Human Capital Management sub-category. This category includes: customer relationship; content; enterprise performance; ERP (Enterprise resource planning) services and operations; and other management tools. So any use case within the category, OrangeHRM in our case, would represent a wide set of cloud-based applications under the Enterprise applications taxonomy.

The second software service (MediaWiki) from the functional area is a knowledge management service, so it goes under the category of Collaboration applications, these category includes: cloud/application tools for web conferencing; team collaboration; knowledge management; and other online community tools. So any use case within the category, MediaWiki in our case, would represent a wide set of cloud-based applications under the Collaboration applications taxonomy.

3.3.7 Apache JMeter and RedLine13

Apache JMeter is Java-based open-source load testing tool for measuring and analysing the performance of a variety of services, with a focus on web-based/cloud applications, available from http://jmeter.apache.org/. JMeter considered as a multi-threading framework allows concurrent demand sampling [150]. The Apache JMeter v.3.3 has been used in this thesis.

RedLine13, a testing service with a focus on bringing load testing to the cloud available from https://www.redline13.com, allows JMeter test scripts to be deployed easily inside a personal cloud domain once connected to the cloud account. This allows tests to be repeated without the need to reset the test parameters. This service required linking the AWS account, using Setup Instructions provided by RedLine13 using Access Management (IAM).

3.4 Cloud Elasticity Concept

As mentioned in Section 2.2, most studies related to performance measuring and testing focuses on quantifying and measuring the elasticity of cloud services. The concept of elasticity is described below, in accordance with earlier technical metrics of elasticity. This thesis follows on from these elasticity concepts to propose the scalability metrics of cloud-based software services that will be discussed fully in Chapter four and five.

Herbst et al. [12] sets a number of key concepts that allows measuring cloud service elasticity in technical term as shown below in Figure 3.5, such as the quantity and time extents for periods of time when the service provision is either below or above what is required by the service demand. Elasticity measures defined by [12], [114] are: the timeshares and average time lengths in under-provisioned and over-provisioned states; the amounts of the over-provisioned and under-provisioned resources per time unit; the averages of the excess and lacking resources; and the jitter, which is the number of resource adaptations during a specific time of provisioning the service.

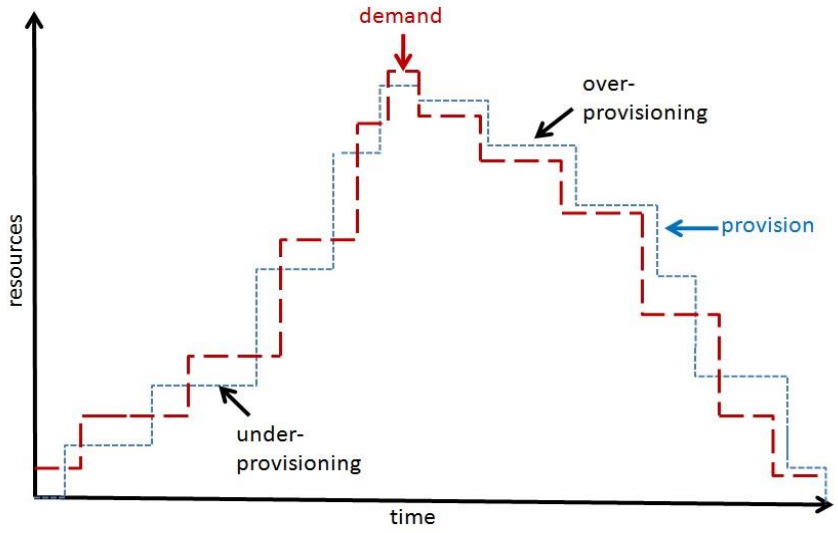

Figure 3.5 Key concepts for measuring elasticity

The up-elasticity and the down-elasticity metrics are defined as the reciprocal value of the product of the average under-provisioned/over-provisioned time length and average lack of resources. Further elaboration [115] that extended the above metrics introduced other factors and ways such as reconfiguration time, functions of resource inaccuracy, and scalability.

The set of the metrics that been proposed by [12], [114], designed to capture the accuracy and timing aspects of elastic platforms. The under-provisioning accuracy metric is calculated as the sum of areas where the resource demand exceeds the supply (provision) in a period, as illustrated in Figure 3.4. While the over-provisioning accuracy metric is calculated based on the sum of areas where the resource supply exceeds the demand [114]. The timing metric of elasticity characterize from the two viewpoints, one from the pure provisioning timeshare, and the other from the viewpoint of the induced jitter accounting for superfluous or missed adaptations [114].

The concept of the service provision elasticity is the short-term flexible provision of the resources [114]. In contrast, this thesis focuses on whether the system can expand/shrink the quantity of the service when this expansion/shrinking is required by demand over a sustained period of service provision.

3.5 Chapter Summary

This chapter describes the tools and established techniques adopted for testing the scalability of cloud-based software services. The chapter starts by explaining the test plan, including the test approach and tasks, as discussed in Section 3.2. It then explains that the plan was developed following IEEE 829 standards. It also describes the test plan in detail, including test items, approach, deliverables, and the environment required. In Section 3.4, provides an explanation of the cloud

elasticity, which have been followed to propose the scalability metrics discussed in the next chapter. Finally, the testing environment, tools and resources used to assess the delivery of this project are discussed in Section 3.3.

In Chapters Four and Five, work undertaken to investigate the measurements and test the scalability performance of cloud-based software services from technical perspective is developed following the methodology described in this chapter. Chapter Six, work undertaken to investigate the fault injection and its impact on the scalability performance of cloud-based software services, uses the methodology described in this chapter as part of the fault injection approach.

Chapter 4 Cloud-based Software Services Delivery from the Perspective of Scalability

In this chapter, a novel investigation to the scalability delivery of cloud-based software services is presented. The study introduces volume and quality scalability metrics based on the number of software instances, and the average response time. An experimental analysis on AWS cloud environment with one cloud-based application (OrangeHRM) is used to demonstrate these metrics, considering three demand scenarios. Practitioners will benefit from the metrics discussed here by better understanding the assessment and testing the scalability of cloud-based delivery of software services in terms of volume and quality. The findings of this chapter have been reported as an extended journal article [23] from previous conference publications; the first paper was published at the fifth IEEE International Symposium on Innovation in Information and Communication Technology [22] and a short paper at the 2018 IEEE World Congress on Services [21].

4.1 Introduction

Measuring and testing the scalability and performance of cloud-based software services is critical for the delivery of such services, and the development of cloud computing [5], [6]. There are three interconnected Cloud-based software services' performance aspects: elasticity, efficiency and scalability [11], [12]. Both elasticity and efficiency are depending on the delivery of a sufficient level of scalability performance. This chapter is focused on testing and measuring the cloud-based software services scalability from a technical perspective.

The work here follow ideas proposed in the context of measurements and metrics for cloud elasticity [71], [72], [87] to propose technical measurement and metrics for scalability of cloud-based software services. The proposed scalability metrics address both volume and quality scaling of cloud-based software services. The metrics can be useful in order to support effective measurement and testing of scalability performance of those services from technical perspective.

This work demonstrates the application of the proposed metrics to a concrete cloud-based software service (OrangeHRM) run through the Amazon EC2 Cloud using three demand scenarios. A discussion on how the metrics can be used to identify differences in the behaviour of the assessed system in the context of different usage scenarios. This work integrates the technical scalability metrics into an earlier utility- oriented scalability metric [18] and calculates the values for each

demand scenarios, in order to enable the scalability analysis from a technical and production- driven perspective.

The rest of the chapter is structured as follows. Section 4.2 provides the approach to measure and quantify scalability of cloud-based software services and explain the metrics based on the measurement approach. Section 4.3 presents the result of an application example using three different usage scenarios to demonstrate the measurement approach and metrics. This is followed by a discussion of the study in Section 4.4, including the implications and importance of the approach and metrics. Finally, the chapter is closed by the summery and conclusion in Section 4.5.

4.2 Scalability Performance Measurement

Scalability is the ability of the cloud-based system to increase the capacity of the software service delivery by expanding the quantity of the software service that is provided when such increase is required by increased demand for the service [13]. This work focus is whether the system can expand the quantity of the service when this expansion is required by demand over a sustained period of service provision. In this work we are not concerned with the short-term flexible provision of the resources, which basically term elasticity of the service provision [114]. The purpose of elasticity is to match the service provision with actual amount of the needed resources at any point in time [114]. Scalability is the ability

of handling the changing needs of an application within the confines of the infrastructure by adding resources to meet application demands as required, in a given time interval [16], [151]. Therefore, the elasticity is scaling up or down at a specific time, and scalability is scaling up by adding resources in the context of a given time frame. The scalability is an integral measurement of the behaviour of the service over a period of time, while elasticity is the measurement of the instantaneous behaviour of the service in response to changes in service demand.

The increase of cloud capacity usually happens by expanding the volume of service demands served by one instance of the software or by providing a lower volume of service through multiple instances of the same software, or a combination of these two approaches. Generally, we expect that if a service scales up the increase in demand for service should be matched by the proportional increase in the service's provision without degradation in terms of quality. In this work, the quality of the service may be seen for example in terms of response time.

The ideal scaling behaviour of the service system should be substantial over a sufficiently long timescale, in contrast with cloud elasticity that looks at short-term mismatches between provision and demand. If the system does not show ideal scaling behaviour, it will increase the volume of the service without changing the quality of that service. Ordinarily, real systems are expected to behave below the level of the ideal scaling and the aim of scalability testing and measurements is to quantify the extent to which the real system behaviour differs from the ideal behaviour.

To match the ideal scaling behaviour, we expect that the system will increase the quantity of the software instances proportionately with the rise in demand for the software services, i.e. if the demand is doubled, we would ideally expect the base number of software instances to also double. We also expect that the system maintains the quality of service in terms of maintaining the same average response time irrespective of the volume of service requests, i.e. if demand was increased by 50%, we would ideally expect no increase in average response time. Formally, let us assume that D and D′ are two service demand volumes, D′ > D. Let I and I′ be the corresponding number of software instances that are deployed to deliver the service, and let t_r and t'_r be the corresponding average response times. If the system scales ideally we expect that for any levels of service demand D and D′ we have that

$$D'/D = I'/I \qquad (1)$$

$$t_r = t'_r \qquad (2)$$

Equation (1) means that the volume of software instances providing the service scale up linearly with the service demand. Equation (2) means that the quality of service, in terms of average response time, remains the same for any level of service demand.

In order to measure the values of I and t_r the system must perform the delivery of the service over a period of time, such that short-term variations corresponding to system elasticity do not influence the measurements. This means that the

measurements should be based on an average number of software instances and average response time measured regularly (e.g. every second) during the execution of a demand scenario following a particular pattern of demand variation. The same demand pattern should be executed multiple times to get reliable averages.

Demand scenarios may follow certain patterns expected to test the scalability of the system in specific ways (more details and justification regarding these demand scenarios in subsection 3.2.4). There are three kinds of demand patterns that appear as natural and typical choices, these three demand scenarios are shown in Figure 4.1. The first scenario is a steady increase followed by a steady decrease in the demand with a set level of the peak. The second scenario is a stepped increase and decrease, again with a set peak level of demand; with this scenario, we schedule to start with 10% of the demand size, then increase 10% stepwise over time, followed by a 10% stepped decrease over time. The final scenario is a varied stepped increase and decrease scenario with a set peak level of demand. This scenario starts with 40% of the demand size, followed by a stepped increase of 20% over time until the assigned demand size is reached, and then a stepped decrease of 10% over time. The purpose of having multiple scenarios is to see how the Auto Scaling service (services that automatically help to ensure that an application has the proper number of instances dynamically, can handle the workload during runtime [140], [141]) handles cloud-based software services with different patterns of growth of workloads and to verify that the cloud resources cover the target system's needs without experiencing a drop in performance.

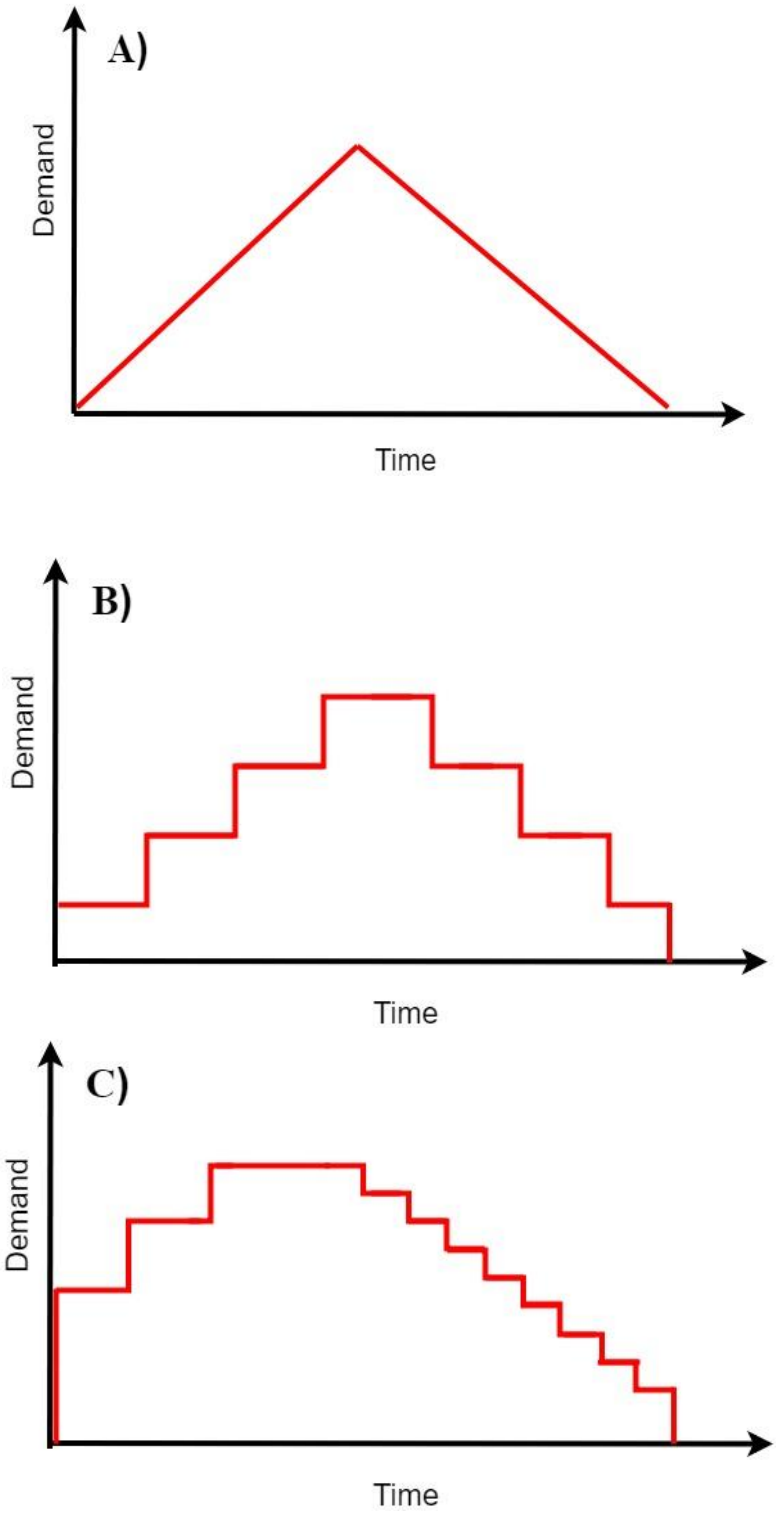

Figure 4.1 Demand scenarios: A) steady rise and fall of demand; B) stepped rise and fall of demand; C) varied stepped rise and fall of demand

Any demand scenario is characterized by a summary measure of the demand level, which may be the peak level or the average or total demand level. This characteristic of a demand scenario is denoted as D.

In general, real-world cloud-based systems are unlikely to deliver the ideal scaling behaviour. Given the difference between the ideal and the actual system scaling behavior, it makes sense to measure technical scalability metrics for cloud-based software services using as reference the ideal scalability behaviour defined in equations (1) and (2).

In terms of provision of software instances for the delivery of the services, the scaling is deficient if the number of actual instances is lower than the ideally expected number of scaling instances. To quantify the level of deficiency we pick a demand scenario and start with a low level of characteristic demand D_0 and measure the corresponding volume of software instances I_0. Then we measure the number of software instances I_k corresponding to a number (n) of increasing demand levels D_k following the same demand scenario, we can then calculate how close are the I_k values to the ideal I^*_k values (in general we expect $I_k < I^*_k$). Following the ideal scalability assumption of equation (1) we get for the ideal I^*_k values:

$$I^*_k = (D_k / D_0) \cdot I_0 \qquad (3)$$

Considering the ratio between the area defined by the (D_k, I_k) values, k = 0,…,n, and the area defined by the (D_k, I^*_k) values we get the metric of service volume scalability of the system η_I:

$$A^* = \sum_{k=1,\ldots,n} (D_k - D_k\text{-}1) \cdot (I^*_k + I^*_{k-1}) / 2 \quad (4)$$

$$A = \sum_{k=1,\ldots,n} (D_k - D_k\text{-}1) \cdot (I_k + I_{k-1}) / 2 \quad (5)$$

$$\eta_I = A / A^* \quad (6)$$

where A and A^* are the areas under the curves evaluated piecewise as shown in Figure 3.2A calculated for actual and ideal I values and η_I is the volume scalability performance metric of the system. The system is close to the ideal volume scalability if η_I is close to 1. If the opposite is the case and η_I is close to 0, then the volume scalability of the system is much less than ideal.

Equation number (5) is based on the default assumption, which assumes that the number of corresponding software instances is below the ideal number of instance, however, any over-provision of cloud service instances that exceed the ideal scaling behaviour is as much of an issue as under-provision, this has been taken into account in Chapter 5 (see subsection 5.3.2.2). In this case, the volume performance metric should be modified to cover over-provision scale, by considering the systematic nature of the deviation from the ideal (downward or upward) in terms of its impact on the performance and on the geometric calculation in equation (5).

Here the definition of the system quality scalability in a similar manner by measuring the service average response times t_k corresponding to the demand levels D_k. Here, the system average response time measures as the average time that the system takes to process a request once it was received. The ideal average response time represented as t_0, following the ideal assumption of equation (2). The system quality scalability is less than ideal if the average response times for increasing demand levels increase, i.e. $t_k > t_0$. By considering the ratio between the areas defined by the (D_k, t_k) values, k = 0,…,n, and the area defined by the (D_k, t_0) values we get a ratio that defines a metric of service quality scalability for the system η_t:

$$B^* = \sum_{k=1,\ldots,n} (D_k - D_{k-1}) \cdot t_0 = (D_n - D_0) \cdot t_0 \qquad (7)$$

$$B = \sum_{k=1,\ldots,n} (D_k - D_{k-1}) \cdot (t_k + t_{k-1}) / 2 \qquad (8)$$

$$\eta_t = B^* / B \qquad (9)$$

where B and B^* are the areas under the curves evaluated piecewise as shown in Figure 3.2B calculated for actual and ideal t values and η_t is the quality scalability performance metric of the system. If η_t is close to 1 the system is close to ideal quality scalability. On the other hand, if η_t is close to 0 the quality scalability of the system is far from the ideal.

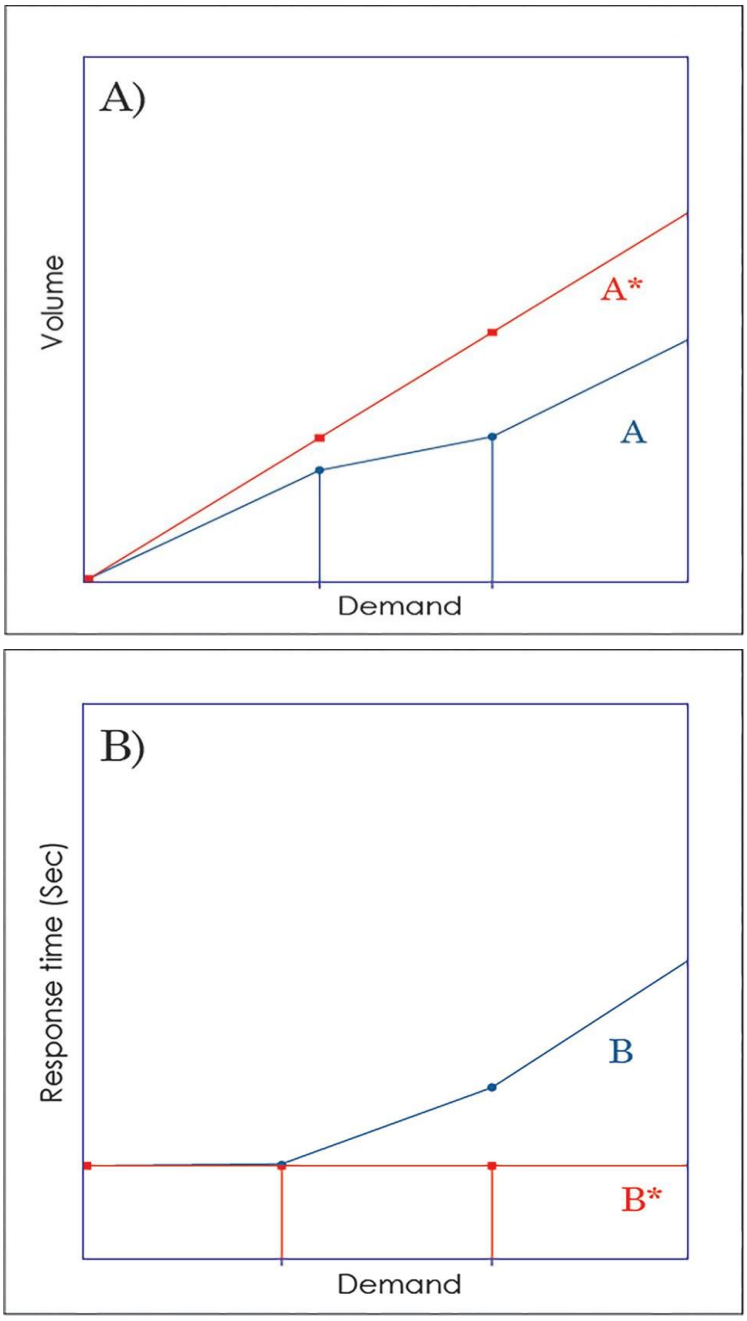

Figure 4.2: The calculation of the scalability performance metrics. A) the volume scalability metric is η_l, which is the ratio between the areas A and A^* – see equation (6); B) the quality scalability metric is η_t, which is the ratio between the areas B^* and B – see equation (9). The red lines indicate the ideal scaling behavior and the blue curves show the actual scaling behaviour

Figure 4.2 illustrates the calculation of the two scalability performance metrics. In Figure 4.2 A, A^* is the area under the red line showing the ideal expectation about the scaling behaviour (see equation (1)) and A is the shaded area under the blue curve, which corresponds to the actual volume scaling behaviour of the system. The blue curve is expected in general to be under the ideal red line, indicating that the volume scaling is less efficient than the ideal scaling. In Figure 4.2 B, B^* is the shaded area under the red line indicating the expected ideal behaviour (see equation (2)) and B is the area under the blue curve, showing the actual quality scaling behaviour of the system. Again, in general, we expect that the blue curve is above the ideal red line, indicating that the quality scaling is below the ideal. We chose nonlinear curves for the examples of actual scaling behaviour (blue curves in Figure 4.1) to indicate that the practical scaling of the system is likely to respond in a nonlinear manner to changing demand.

The above-defined scalability metrics allow the effective measurement of technical scalability of cloud-based software services. These metrics do not depend on other utility factors such as cost and non-technical quality aspects. This allows us to utilize these metrics in technically focused scalability tests that aim to spot components of the system that have a vital impact on the technical measurability, and additionally the testing of the impact of any change in the system on the technical system scalability.

Applying these metrics to different demand scenarios allows the testing and tuning of the system for particular usage scenarios and the understanding of how

system performance can be expected to change as the pattern of demand varies. Such application of these metrics may highlight trade-offs between volume scaling and quality scaling of the system that characterize certain kinds of demand pattern variation (e.g. the impact of the transition from low-frequency peak demands to high-frequency peak demands or to seasonal change of the demand). Understanding such trade-offs can help in tailoring the system to its expected or actual usage.

4.3 Application Example and Results

To demonstrate the applicability of the scalability metrics, the Amazon AWS cloud environment, and the OrangeHRM as the cloud-based software service, have been used. Here the work follows the testing methodology that has been presented in Chapter 3.

To measure the scalability, we simulate the user demand scenarios using the Apache JMeter script, and run through Redline13 services after connecting our Amazon account to the service. To provide the scaling of the service, we relied on the Auto-Scaling and Load-Balancer services provided by the Amazon AWS cloud. An EC2 instance has been set-up and configured to host the targeted application through the Amazon EC2 management console. Both Auto-Scaling and Load-Balancer services have been connected to the application instance, and the CloudWatch service to monitor the scaling performance and parameters been

attached to the software service. The experimental data has been collected through both Redline13 and Amazon's CloudWatch services. In this study, the system average response time was measured as the average amount of time that the application takes to process a HTTP request after it has received one. The parameters of the Amazon EC2 virtual machine, and Auto-scaling polices that have been used for the experiments are given in Table 4.1. The service requests consisted of HTTP requests to the main page of the application by gaining login access using the Apache JMeter script.

Table 4.1: EC2 virtual machine parameters and Auto-Scaling policies

Virtual Machine Parameters			
Instance type: t2.micro			
vCPUs	RAM (GiB)	CPU Credits/hr	Storage (GB)
1	1.0	6	10
Auto Scaling Policies			
Add Instance	When 80% >= CPUUtilization < +infinity		
Remove Instance	When 30% <= CPUUtilization > -infinity		

Redline13 services have been used by uploading the test script into our account; which allows us to easily deploy JMeter test plans inside our Amazon AWS domain and repeat the tests without the need to reset the test parameters again, this allows efficient extraction of the data.

Three demand scenarios have been used in this study. The first scenario follows the steady rise and fall of demand pattern shown in Figure 4.1A. The second scenario consists of a series of stepwise increases and falls in demand, conceptually similar to the demand pattern shown in Figure 4.1B. The third

scenario consists of a varied series of stepwise increases and decreases in demand shown in Figure 4.1C. Examples of the three kinds of experimental demand patterns (users running at runtime) are shown in Figure 4.3. The volume of demand and experimented were varied with four demand sizes, with 100, 200, 400 and 800 service requests in total.

All the experimental settings (i.e. demand pattern and demand volume combinations) have been run 20 times, in total 240 experimental runs. The average number of simultaneously active software instances and the average response time for all service requests for each experimental run, have been calculated. The averages and standard deviations of simultaneously active software instances and average response times over the 20 experimental runs, also have been calculated. Note that the standard deviations are small relative to the averages over the 20 runs. The average number of software instances for the three scenarios and for the four demand levels are shown in Figure 4.4. The average response times for the three scenarios and four demand levels are shown in Figure 4.5.

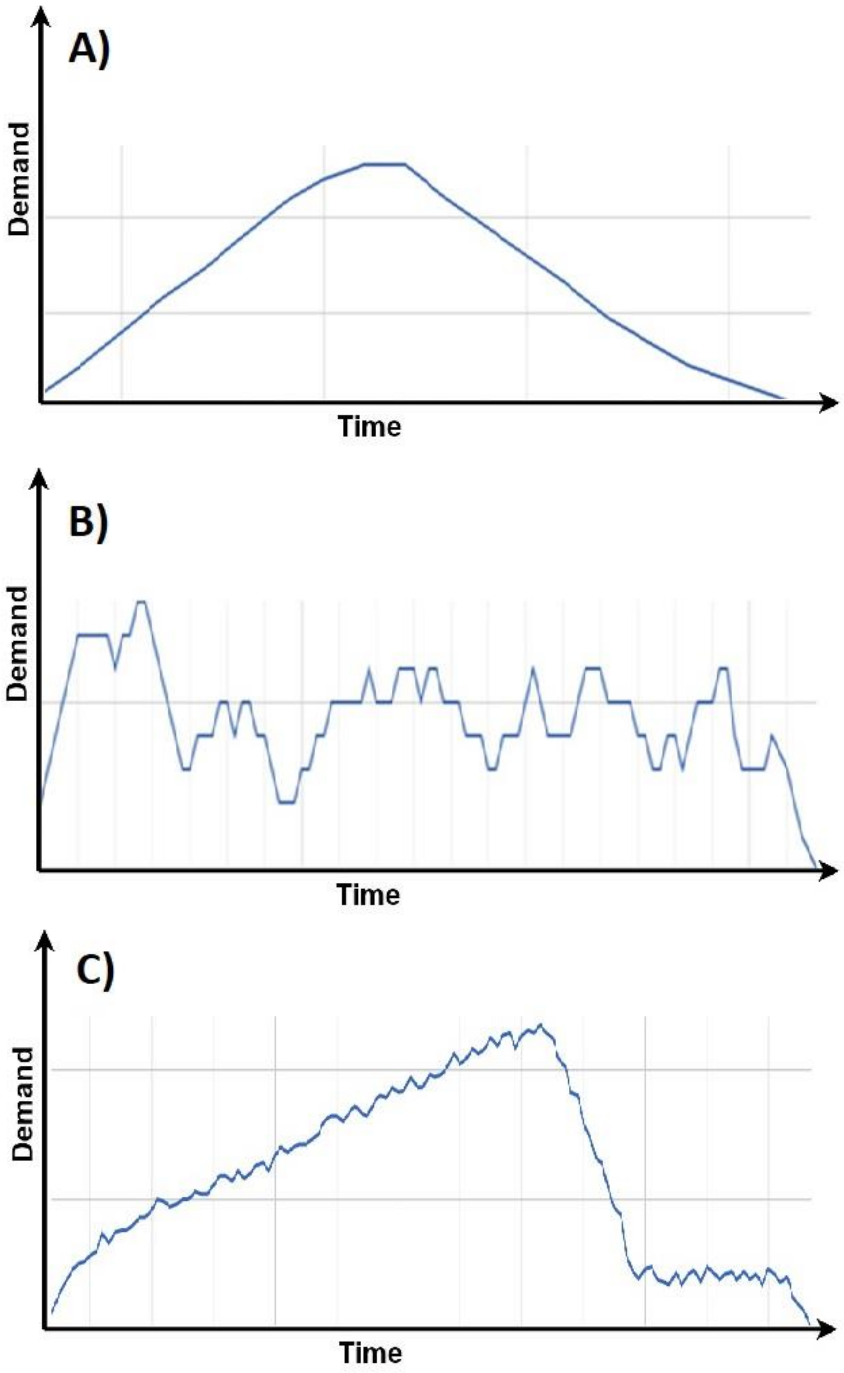

Figure 4.3: Typical experimental demand patterns: A) steady rise and fall of demand; B) series of step-wise increases and decreases of demand; C) varied stepped rise and fall of demand

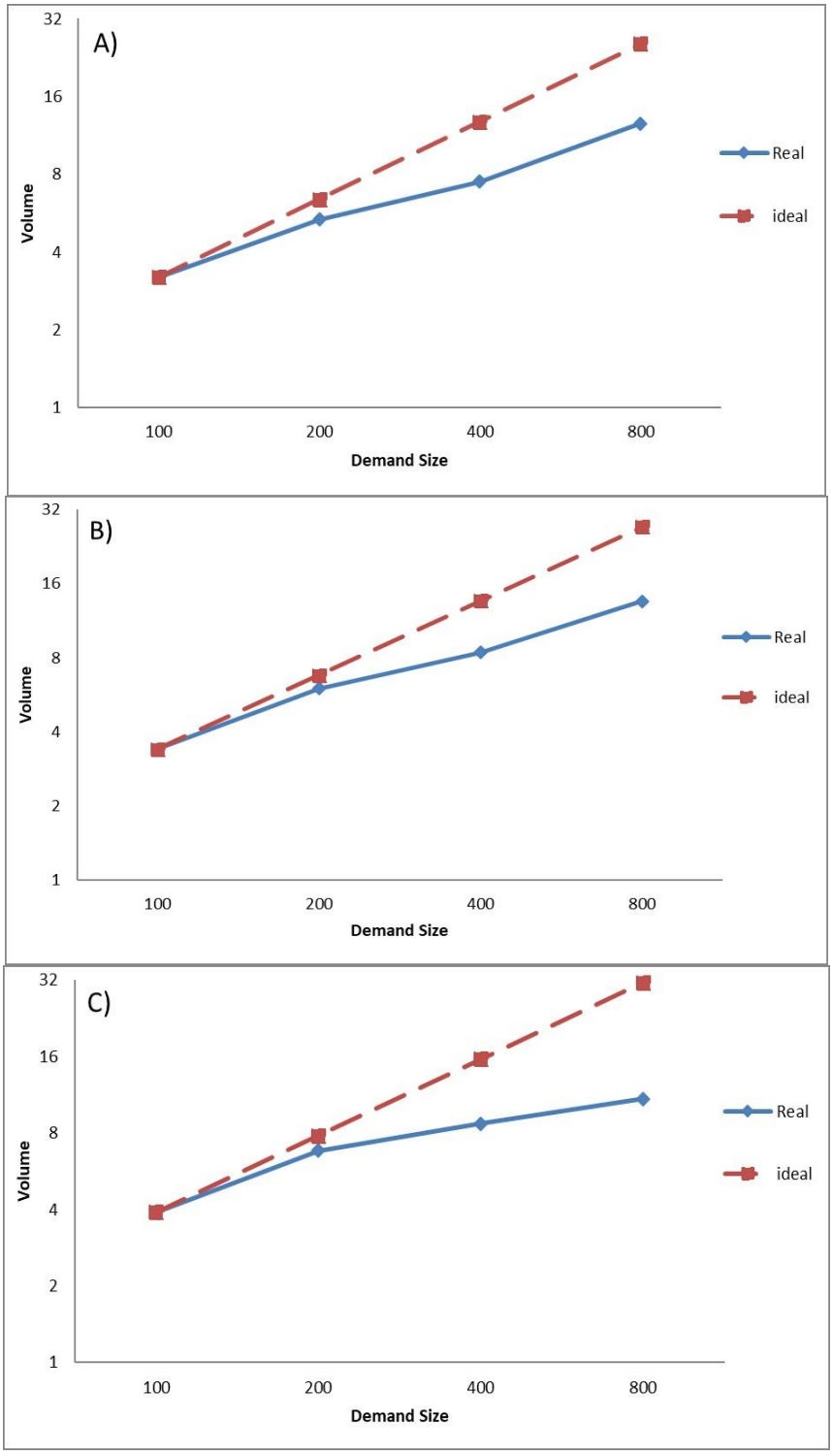

Figure 4.4: The average number of software instances: A) steady rise and fall of demand; B) series of step-wise increases and decreases of demand; C) varied stepped rise and fall of demand

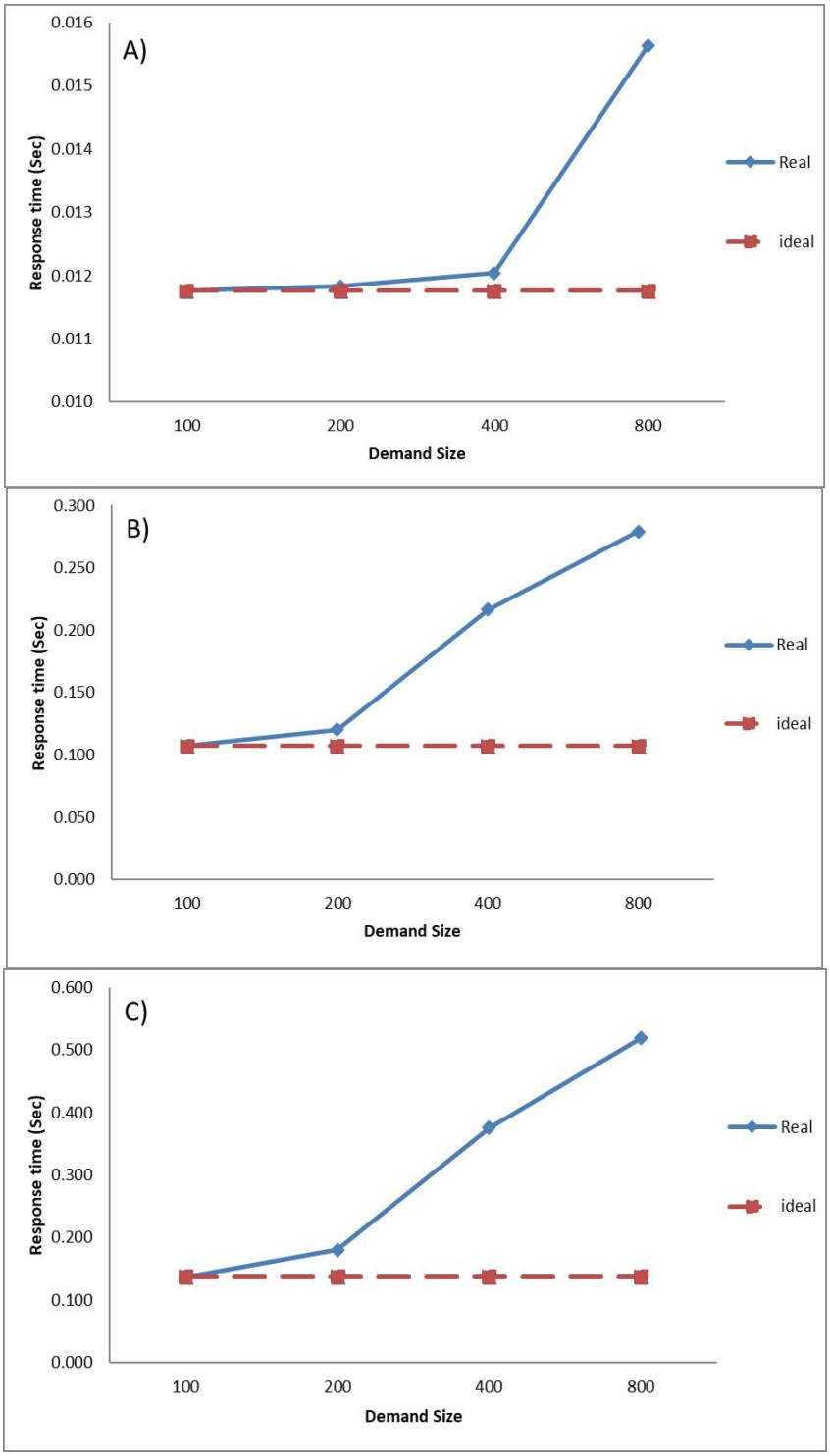

Figure 4.5: The average response times: A) steady rise and fall of demand; B) series of step-wise increases and decreases of demand; C) varied stepped rise and fall of demand

As Shown in Figure 4.4 and 4.5, the application performs similarly in term of volume (instances) scaling for the first two scenarios (steady rise and fall of demand, and series of step-wise increases and decreases of demand), while in the varied stepped rise and fall of demand as shown in Figure 4.4C, the scaling acted slightly differently when demand hit 400 the scaling volume dropped.

The observed average response time values for the stepped rise and fall of demand scenario are shown in Figure 4.5B and for varied stepped rise and fall of demand in Figure 4.5C, starting from demand size of 200 the average response time increases significantly. In contrast, average response time values for the first scenario which shown in Figure 4.5A, have increased gradually from demand size of 400 with less variation between values of average response times.

Values for the scalability metrics η_I and η_t for the three demand scenarios that we considered, are shown in Table 4.2. The calculated metrics show that in terms of volume scalability the first two scenarios are similar, the scaling being slightly better in the context of the scenario with step-wise increase and decrease of demand. The results show that the scaling volume for the third scenario dropped by 8-10 per cent in comparison with the first two scenarios

Table 4.2: Scalability metrics values

Scenario	**Metric**	
	η_I	η_t
Steady rise and fall	0.5687	0.9041
Step-wise increase and decrease	0.5882	0.5201
Varied Step-wise increase and decrease	0.4888	0.3834

In terms of quality scalability, the system scales much better in the context of the first scenario, steady rise and fall of demand, than in the case of the second scenario with step-wise increase and decrease of demand, and the varied Step-wise increase and decrease scenario.

The values of the metrics indicate that in the context of variable demand scenarios (the second and third scenarios) - which is likely to be more realistic demand scenarios for many cloud-based software services - the quality scaling performance drops considerably in comparison with the simpler demand scenario.

4.4 Discussion

The proposed scalability metrics address both volume and quality scaling of cloud-based software services, and provide a practical measure of these features of such systems. The works show how to integrate aspects of non-technical features [18] and also are distinct from elasticity oriented metrics [12]. This is important in order to support effective measurement and testing of scalability performance of the system from technical perspective.

Having an effective measure of the volume and quality scalability of the system allows exploring the contribution of various system components to the scalability performance of the system. For example, using mutation testing [152] we can test the impact of small changes to particular components on the scalability performance. Alternatively, by instrumenting the whole code of the system [153]

and then measuring its scalability through a range of demand scenarios we can identify the components of the system at various resolutions (e.g. units, classes, functions, methods) that contribute critically to variations in scalability performance. Such identification of scalability-critical components can drive the design of scalability tests, system revision and upgrade focused on improvement of scalability, or development of fine-grained monitoring of system scalability performance.

In this work the quality scaling is considered through measurement of the average response time of the system. Other aspects of quality scaling could be also used to define further similar but functionally distinct quality scaling metrics. For example, system throughput (i.e. the rate of successful delivery of service provision in response to service demand), or slowdown, or recovery rate [18] can be used for alternative quality scaling metrics. Expanding the range of quality scaling metrics provides a multi-factor view of quality scaling supporting the identification and definition of trade-off options in the context of quality-of-service offerings in terms of service scaling. The equations of the quality metric can be amended based on the nature of the quality factor that could replace or combine with the current quality scaling feature.

Due to the importance and need of measuring the scalability from an economic perspective, therefore, the proposed metrics can be integrated into the utility-oriented scalability metric proposed by Hwang et al. [18], by combining our metrics as the performance and/or quality components of their utility-oriented

scalability metric. This will allow the analysis of the scalability of cloud-based software services from both technical and production-driven perspectives. The utility oriented productivity metric (P(Λ)) is given as [18]:

$$P(\Lambda) = p(\Lambda) \cdot \omega(\Lambda) / c(\Lambda) \qquad (10)$$

where Λ is the system configuration, p(Λ) is the performance component of the metric – in our case this is the volume scalability metric, ω(Λ) is the quality component of the metric – in our case this is the quality scaling metric, and c(Λ) is the cost component of the metric. This leads to a re-definition of the utility-oriented metric as:

$$P(\Lambda) = \eta_l (\Lambda) \cdot \eta_t (\Lambda) / c(\Lambda) \qquad (11)$$

by adopting $p(\Lambda) = \eta_l(\Lambda)$ and $\omega(\Lambda) = \eta_t(\Lambda)$.

Table 4.3 show the calculated values of the integrated productivity metric based on values of technical scalability metrics (see Table 4.2 for the metrics values) and cost (AWS t2.micro instance (0.0132$/hour)). It should be noted that the stepped scenarios (step-wise increase and decrease, and varied step-wise increase and decrease) which more realistic and powerful scenarios has scored lower than the simpler scenario. The utility-oriented integrated scalability calculations show that in the case of the systems that we compared, the best choice is to use of simpler demand scenario on EC2.

Table 4.3: Integrated scalability metric values

Scenario	Values
Steady rise and fall	38.95
Step-wise increase and decrease	23.18
Varied step-wise increase and decrease	14.198

Here three demand scenarios have been used to demonstrate the effect of demands patterns on the scaling metrics. In principle, various demand scenarios may be used to fine-tune the cloud-based software service to fit particular demand scenario expectations. Similarly, considering a set of demand scenarios can also be used to identify changes in such scenarios that trigger interventions in terms of software upgrade or maintenance or direct investment of software engineering resources in development of focused upgrades for the system. Demand scenarios combined with multiple versions of quality scaling metrics can also be used to determine reasonable quality-of-service expectations and likely variations of such expectations depending on changes in demand scenarios. The review [154] which concerns to study of the current practice of cloud service performance evaluation from system modelling perspective. It can be useful to adopt another demand scenario that already been used in the field, in order to track the impact of such scenarios.

4.5 Summary and Conclusions

In this chapter, two technical scalability metrics for cloud-based software services have been introduced. One of these addresses the volume scalability of the service, while the other the quality scalability of the service. The metrics are based on simple principles of proportional scaling of the service volume and constant provision of the service quality, and are defined using the differences between the real and ideal scaling curves for both the volume and quality scalability.

The proposed metrics can be used alone or integrate into utility oriented metrics of cloud-based service scalability [11]. In order to facilitate scalability analysis of cloud-based software services from technical and utility-oriented perspectives. The metrics are demonstrated using a cloud-based software service (OragnceHRM) run on the Amazon AWS cloud platform and considering three demand scenarios. The results show that the proposed metrics quantify explicitly the technical scalability performance of the system and that they allow the clear assessment of the impact of demand scenarios on the cloud-based software service.

The proposed technical scalability metrics can be used to perform and design scalability testing of cloud-based software systems with the aim to identify system components that critically contribute to the technical scalability performance. Furthermore, the proposed metrics can be extended, by considering alternative service quality features, and combined with a range of demand scenarios to support the fine-tuning of the system, the identification of quality-of-service trade-

offs, and estimation of realistic scalability performance expectations about the system depending on demand scenarios. The findings of this chapter aim to address the thesis objectives 2-4 (see Section 1.2).

In this chapter only one cloud platform (Amazon AWS) and only one cloud-based software service (OrangeHRM) have been used to demonstrate the application and usefulness of the scalability metrics [21]. Naturally, expanding the experiments to cover multiple cloud platforms and multiple cloud-based software services would provide a fuller picture of the application of the proposed metrics. Finally, here one particular setting of the cloud service (i.e. virtual machine specification) and one load generator have been used to implement the demand scenarios and the scaling of the investigates cloud-based software service. Alternative load generators might have an impact on the values of the calculated metrics due to their implementation details, although in principle we would not expect major impact of these on the reported results. These constraints are discussed further in the following chapter.

Chapter 5 Scalability Analysis Comparisons of Cloud-based Software Services

The issues identified in Chapter 4 will be addressed here. Particularly, we expand the experiments to cover multiple cloud platforms and multiple cloud-based software services in order to provide a comprehensive picture of the application of the proposed scalability metrics. A novel investigation of the comparison of the scalability analysis of cloud-based software services delivery is described in this chapter. The Chapter addresses the important issue of the understanding of the scalability of cloud-based software services, which is increasingly important as more software are migrated to the cloud. The results show that the metrics can be used effectively to compare the scalability of software on cloud environments and consequently to support deployment decisions with technical arguments. The findings from this study have been reported as a journal article [24].

5.1 Introduction

In Chapter 1, the importance to investigate the scalability of cloud-based software services has been discussed, in particular in the context of supporting the future optimisation and growth of cloud computing based services. Measuring the scalability performance from a technical perspective is key for understanding the performance of cloud-based software services, especially with the exciting of auto-scaling and load-balancing services that can help such services to handle the events of sudden workload hits during runtime [140].

In this work, the technical measurement of the scalability of cloud-based software services have been used, that were introduced in Chapter 4. Two real-word cloud-based systems have been used to demonstrate the usefulness of the metrics and compare their scalability performance in two cloud platforms: Amazon EC2 and Microsoft Azure. The experimental analysis considers three sets of comparisons: first, comparing the same cloud-based software service hosted on two different public cloud platforms; second comparing two different cloud-based software services hosted on the same cloud platform; finally, comparing between the same cloud-based software service hosted on the same cloud platform with two different auto-scaling policies.

In this Chapter, the experiments have been expanded to demonstrate the metrics application by using two cloud-based software services (OrangeHRM and/or MediaWiki) run through the Amazon EC2 and Microsoft Azure clouds. The

metrics can be used to show differences in the system behaviour based on different scaling scenarios, configuration settings, or cloud platforms. A discussion on how to use these metrics for measuring and testing the scalability of cloud-based software services, is provided.

In the previous chapter the technical scalability measurements and metrics were proposed and demonstrated, thus, extending the applicability of the metrics is necessary as limited experimental settings were used. However, this chapter not only provides an extension of the practicality of the metrics, but it also provides the platform to construct the metrics as a basis that can be used effectively to compare the scalability of software on cloud environments, and consequently supporting deployment decisions with technical arguments.

The rest of the Chapter is structured as follows. Section 5.2 discuss the scalability performance metrics and demand scenarios that been used in this Chapter. Section 5.3 presents the result of an application example using three different usage scenarios to demonstrate the measurement approach and metrics. This is followed by a discussion of the study in Section 5.4, including the implications and importance of the work. Finally, the chapter is closed by the summary and conclusion in Section 5.5.

5.2 Scalability Performance Metrics and Demand Scenarios

Following the novel approach to measure and quantify scalability of cloud-based software services and the explanation of the metrics based on the measurement

approach, as presented in Chapter 4, section 4.2. In this Chapter, we describe an extension of the experimental analysis to include more cloud environments, cloud-based software services, and hardware configurations to demonstrate the use of the scalability metrics.

The measurement approach presented in Section 4.2, explains both scalability metrics; volume and quality scaling scalability of cloud-based software services (see Section 4.2). The volume metrics (η_I) is defined as follows:

$$A^* = \sum_{k=1,\ldots,n} (D_k - D_k\text{-}1) \cdot (I^*_k + I^*_{k-1}) / 2 \quad (1)$$

$$A = \sum_{k=1,\ldots,n} (D_k - D_k\text{-}1) \cdot (I_k + I_{k-1}) / 2 \quad (2)$$

$$\eta_I = A / A^* \quad (3)$$

The quality metric is defined (η_t) as follows:

$$B^* = \sum_{k=1,\ldots,n} (D_k - D_{k-1}) \cdot t_0 = (D_n - D_0) \cdot t_0 \quad (4)$$

$$B = \sum_{k=1,\ldots,n} (D_k - D_{k-1}) \cdot (t_k + t_{k-1}) / 2 \quad (5)$$

$$\eta_t = B^* / B \quad (6)$$

Where:

- D and D′ is the service demand volumes.
- I and I′is the corresponding number of software instances.
- t_r and t'_r is the corresponding average response times.

The performance measures the number of scaling instances, and average response times for cloud-based software services scalability, to provide a practical measure of these features of such systems. This is important in order to support effective measurement and testing the scalability of cloud-based software systems.

Two kinds of demand scenarios have been used in this chapter. The first scenario is a steady increase followed by a steady decrease in the demand with a set level of the peak. The second scenario is a stepped increase and decrease, again with a set peak level of demand; with this scenario, we schedule to start with 10% of the demand size, then increase 10% stepwise over time, followed by a 10% stepped decrease over time. These two demand scenarios are shown in Figure 5.1.

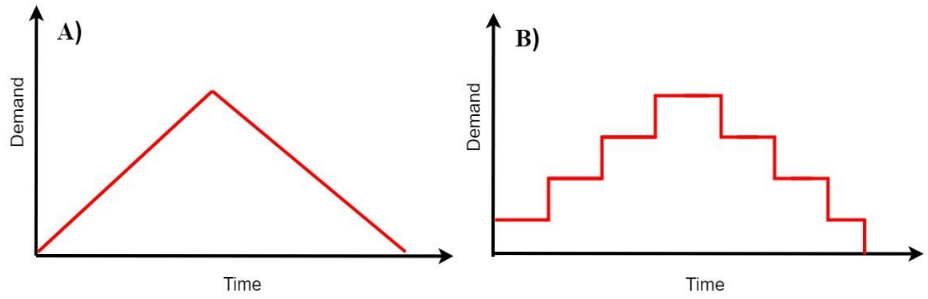

Figure 5.1: Demand scenarios: A) steady rise and fall of demand; B) stepped rise and fall of demand

5.3 Experimental Setup and Results

To validate the volume and quality metrics, experiments on Amazon AWS and Microsoft Azure cloud platforms, using OrangeHRM and Mediawiki as cloud-based software services, have been provided. The purpose is to check the

scalability performance of cloud-based applications using different cloud environments, configuration settings, and demand scenarios. We applied the similar experimental settings for the same cloud-based system (OrangeHRM) in two different cloud environments (EC2 and Azure). We have changed the parameters for Mediawiki, which runs a different type of instance on AWS EC2 environment. Finally, experiments with different auto-scaling polices have been conducted. Table 5.1 illustrates the hardware configurations for both cloud platforms. Here the work follows the testing methodology that has been presented in Chapter 3.

Table 5.1: Hardware configrations for cloud platforms

Platform	Type	CPU Credits/hr	V-CPU(s)	RAM	Price ($/ Hr)
Amazon EC2 (London)	t2.micro (Linux)	6	1	1	0.0132
	t2.medium (Linux)	24	2	4	0.052
MS Azure (UK South)	StandardA1 (Linux)	6	1	1.75	0.06

To provide the scaling of the services we relied on the Auto-Scaling and Load-Balancer services provided by both Amazon AWS and Microsoft Azure. Furthermore, Amazon CloudWatch and Azure Monitor services have been configured in order to monitor the parameters. The Auto-scaling polices (the default policies that are offers by the cloud providers when setting up an auto-scaling group) that have been used for the first two set of experiments are given in Table 5.2.

Table 5.2: Auto-Scaling polices

Auto Scaling Policies	
Add Instance	When 80% >= CPUUtilization < +infinity
Remove Instance	When 30% <= CPUUtilization > -infinity

In this study, we perform three kinds of comparisons, one between the same cloud-based software hosted on two different cloud platforms (EC2 and Azure). The second comparison is between two different cloud-based software services hosted on the same cloud platform (EC2). The third is between the same cloud-based software service hosted on the same cloud platform (EC2) with different Auto-scaling polices. The parameters of these experiments are listed in Table 5.3.

Table 5.3: Cloud-based services, workload, and cloud platform

System service	Cloud provider / Instance type	Workload generator
OrangeHRM	Amazon EC2 / t2.micro	JMeter script run by Redline13 services.
OrangeHRM	Microsoft Azure / Standard A1	JMeter script run by Redline13 services.
Mediawiki	Amazon EC2 / t2.medium	Redline13

For OrangeHRM experiments (hosted on EC2 and Azure), we simulate the workload using an Apache JMeter script (http://jmeter.apache.org/) and run through Redline13 services after connecting our cloud accounts to the service (https://www.redline13.com).

We used Redline13 services by uploading the test script into our account; which allows us to easily deploy JMeter test plans inside our cloud domain and repeat the tests without the need to reset the test parameters again. This allows efficient extraction of the data. The experimental data has been collected through both

Redline13 management portal and the monitoring services from EC2 and Azure. The service requests consisted of an HTTP request to all pages and links of OrangeHRM by gaining login access using the Apache JMeter script. The Redline13 Pro services used to test Mediawiki, which allows us to test the targeted application by covering HTTP requests for all pages and links, including getting authentication (log in) to the application's admin page.

5.3.1 Experimental Process

The cloud resources must be adequately configured to measure up to the workload in order to achieve efficient performance and scalability. We considered two demand scenarios as shown in Figure 5.1. The first scenario follows the steady rise and fall of demand pattern (see Figure 5.1A). The second scenario consists of a series of stepwise increases and falls in demand as shown in Figure 5.1B. Examples of the two kinds of experimental demand patterns are shown in Figure 5.2A is an example of experiments on Mediawiki in AWS EC2 and Figure 5.2.B is an example of experiments on OrangeHRM in Microsoft Azure. The volume of demand and experimented were varied with four demand sizes, with 100, 200, 400 and 800 service requests in total.

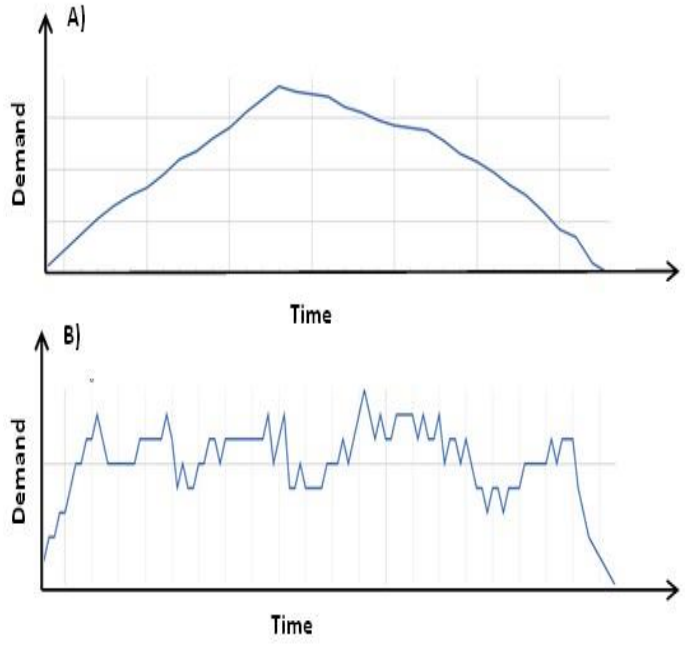

Figure 5.2: Typical experimental demand patterns: A) Mediawiki/EC2 - Steady rise and fall of demand; B) OrangeHRM/Microsoft Azure - Series of step-wise increases and decreases of demand

All experimental settings were repeated 20 times, in total 640 experimental were conducted. The average number of simultaneously active software instances and the average response time for all service requests for each experimental run has been calculated. In this study, the system average response time was measured as the average time that the targeted system takes to process an HTTP request once it was received. The averages and standard deviations of simultaneously active software instances and average response times over the 20 experimental runs have been calculated. It is to be noted that the standard deviations are small relative to the averages over the 20 runs, for this reason, we do not show them in the figures to avoid cluttering.

5.3.2 Measured Cloud-Based Software Services Results

5.3.2.1 Results for The Same Cloud-Based Software System On EC2 and Azure

To achieve fair comparisons between two public clouds, we used similar software configurations, hardware settings, and a workload generator in the experiments. To measure the scalability for the proposed demand scenarios for the first cloud-based software service (OrangeHRM) hosted in EC2 and Azure. The average number of OrangeHRM instances for both scenarios and for the four demand workloads are shown in Figure 5.3. The average response times for both scenarios and four demand workloads are shown in Figure 5.4. In both figures, the 'Ideal' lines show the expected value of average response time, assuming that the scaling of the software service works perfectly. The 'Real' curves show the actual measured average response times.

Notice that there are variations in average response times for the same cloud-based application hosted on two different cloud platforms (EC2 and Azure). So all configurations for instances, Auto-Scaling, and Load-Balancer services for both cloud accounts have been checked, to make sure that all configuration settings match. A number of tests have been re-run to make sure that the variations in results are not caused by configuration differences.

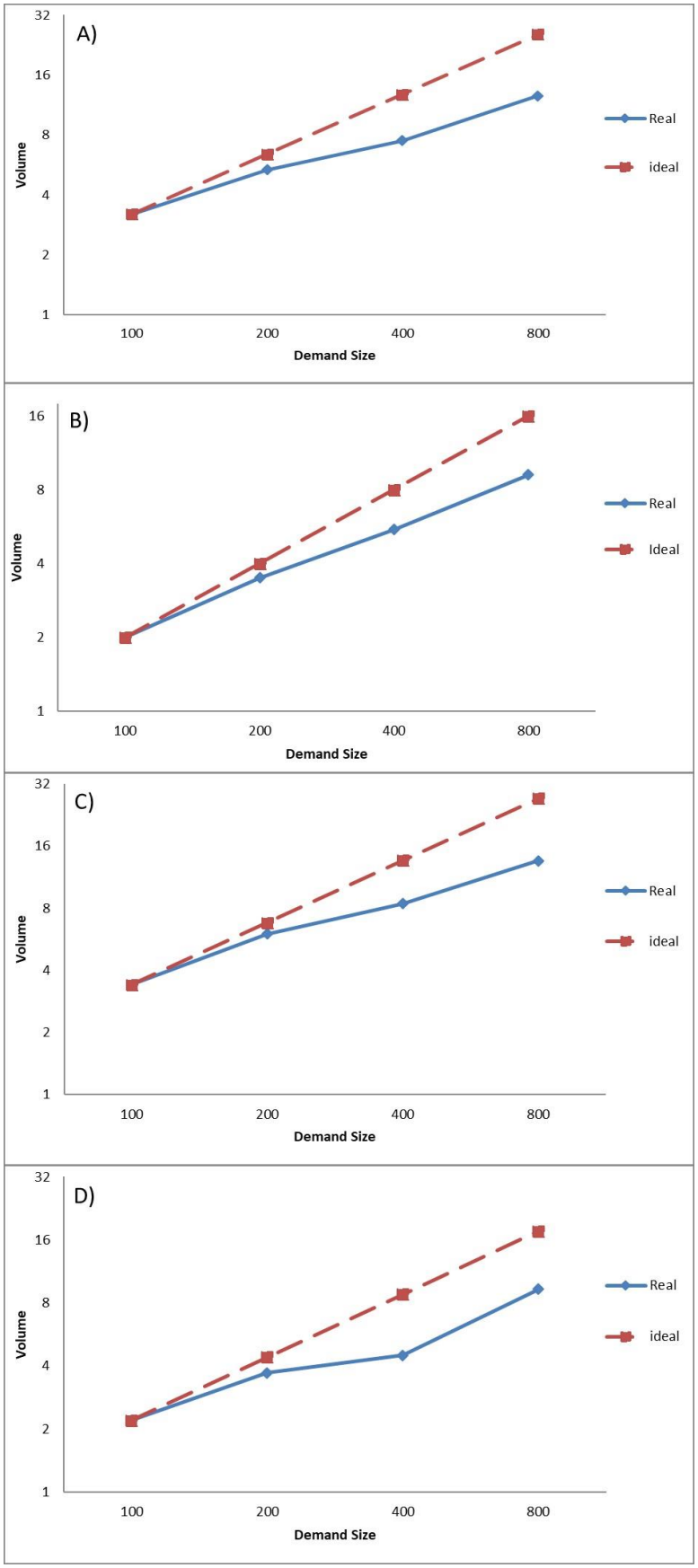

Figure 5.3: The average number of software instances. A) OrangeHRM/EC2 – Steady rise and fall of demand scenario. B) OrangeHRM/Azure - Steady rise and fall of demand scenario. C) OrangeHRM/EC2– Series of step-wise increases and decreases of demand scenario. D) OrangeHRM/Azure– Series of step-wise increases and decreases of demand scenario

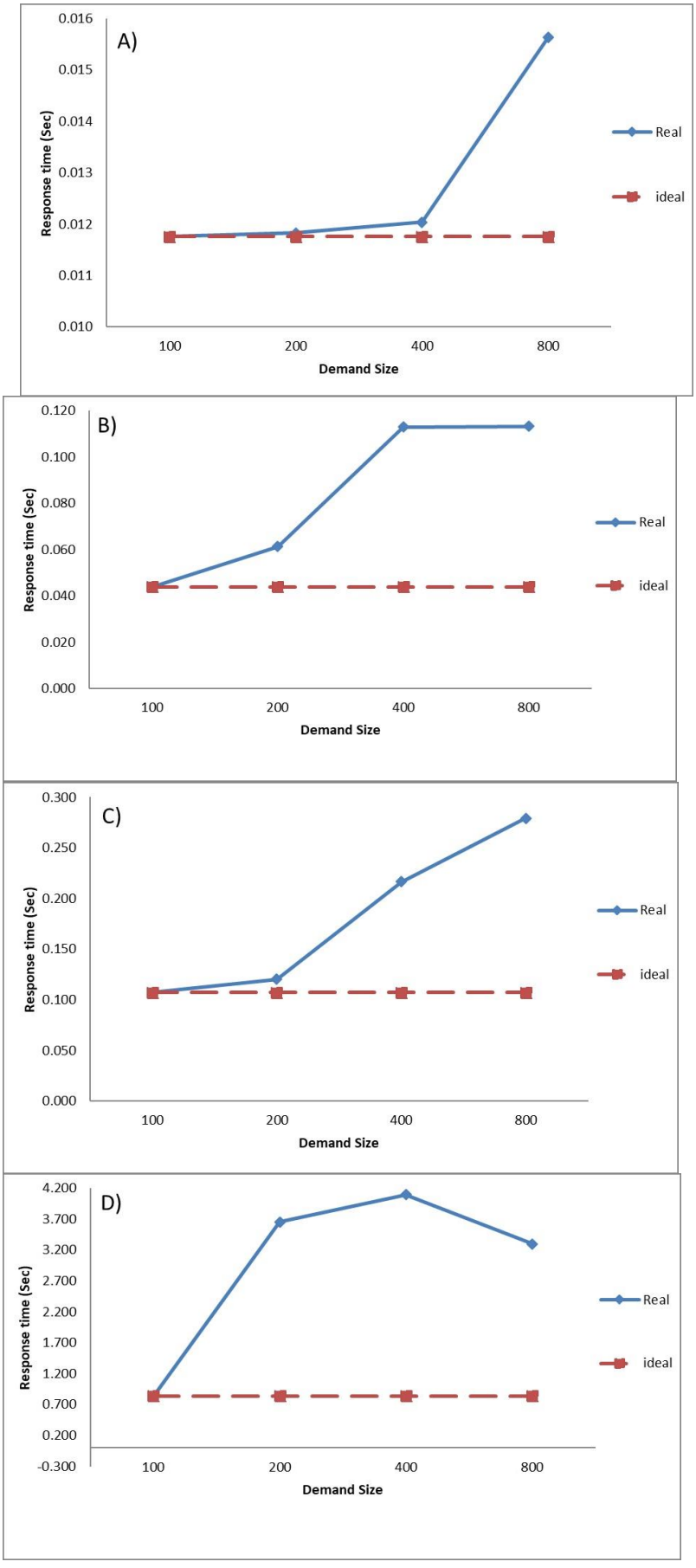

Figure 5.4: The average response times. A) OrangeHRM/EC2 – Steady rise and fall of demand scenario. B) OrangeHRM/Azure - Steady rise and fall of demand scenario. C) OrangeHRM/EC2– Series of step-wise increases and decreases of demand scenario. D) OrangeHRM/Azure– Series of step-wise increases and decreases of demand scenario

There have been other investigations about variations in average response times for cloud-based applications by [155], [156]. There are a number of factors that could cause variations such as: bursty workload, software component management strategies, bursts in system consumption of hardware resources, and network latency. However, all software configurations, hardware settings, and workload generator are similar in our experiments.

The observed average response time values for Azure for the stepped rise and fall of demand scenario are shown in Figure 5.4D. Starting from the demand size of 200 the response time increases significantly. Once the demand size reached 800 the average response time began to decline significantly. In contrast, response time values for EC2 for the same scenario which shown in Figure 5.4C, have increased gradually with less variation.

The scalability metrics η_I and η_t for the two demand scenarios for the cloud-based application for both cloud platforms have been calculated. The values of the scalability metrics are shown in Table 5.4. The calculated metrics for EC2 show that in terms of volume scalability the two scenarios are similar, the scaling being slightly better in the context of the step-wise increase and decrease of demand scenario. In contrast, Azure shows better volume scaling in the first scenario (Steady rise and fall) with around 0.65, while in the second scenario the volume scaling performance for the Azure is slightly less than the corresponding performance for the EC2.

Table 5.4: Scalability metrics values

Cloud Provider	Scenario	Metric	
		η_I	η_t
Amazon EC2	Steady rise and fall	0.5687	0.9041
	Step-wise increase and decrease	0.5882	0.5201
Microsoft Azure	Steady rise and fall	0.6532	0.4526
	Step-wise increase and decrease	0.5592	0.2372

In terms of quality scalability, the EC2 hosted system scales much better in the context of the first scenario, steady rise and fall of demand, than in the case of the second scenario with step-wise increase and decrease of demand. In contrast, Azure shows lower quality scalability than EC2 in this respect, with the metric being 0.45 in the first scenario, and 0.23 for the second scenario.

The values of both metrics η_I and η_t for both clouds that software system performed better with respect to both volume and quality in the first scenario, steady rise and fall of demand, which is more realistic and simpler demand scenario for many cloud-based software services. In general, OrangeHRM performed better in Amazon EC2, in the terms of quality scalability, while performed slightly better in Azure in the terms of volume scalability for the steady rise and fall demand scenario. In the case of the variable rise and fall of demand, the OrangeHRM performs considerably better on the EC2 than on the Azure.

The big difference in the average response times for the software system running on the two cloud platforms indicates that either the software system is tailored better to the provisions of the EC2 system or that the Azure might have issues with the speed of service delivery for the kind of service software systems like the

OrangeHRM (or for some particular kind of technical aspect of this software system). Both options raise interesting questions and opportunities for further investigation of the technical match between a software system and the cloud platforms on which it may run.

5.3.2.2 Results for Different Cloud-Based Software Systems On EC2

Different software configurations, hardware settings, and workload generator in this set of experiments have been used to measure the scalability of the two scenarios for both cloud-based software services that have been hosted in EC2. We changed the instance type and the workload generator in order to see the changes in scalability performance when using different and larger experimental settings. The purpose of this kind of comparison is to see the effects on the scalability performance using the same cloud platform while using different types of instances and workload generators. The average number of OrangeHRM instances for both scenarios and for the four demand workload levels are shown in Figure 5.3A and Figure 5.3C. The average numbers of MediaWiki instances for both scenarios and for the four workload levels are shown in Figure 5.5A and Figure 5.5B. The average response times of OrangeHRM for both scenarios and four demand workload levels are shown in Figure 5.4A and Figure 5.4C. The average response times of MediaWiki for both scenarios and for the four workload levels are shown in Figure 5.5C and Figure 5.3D.

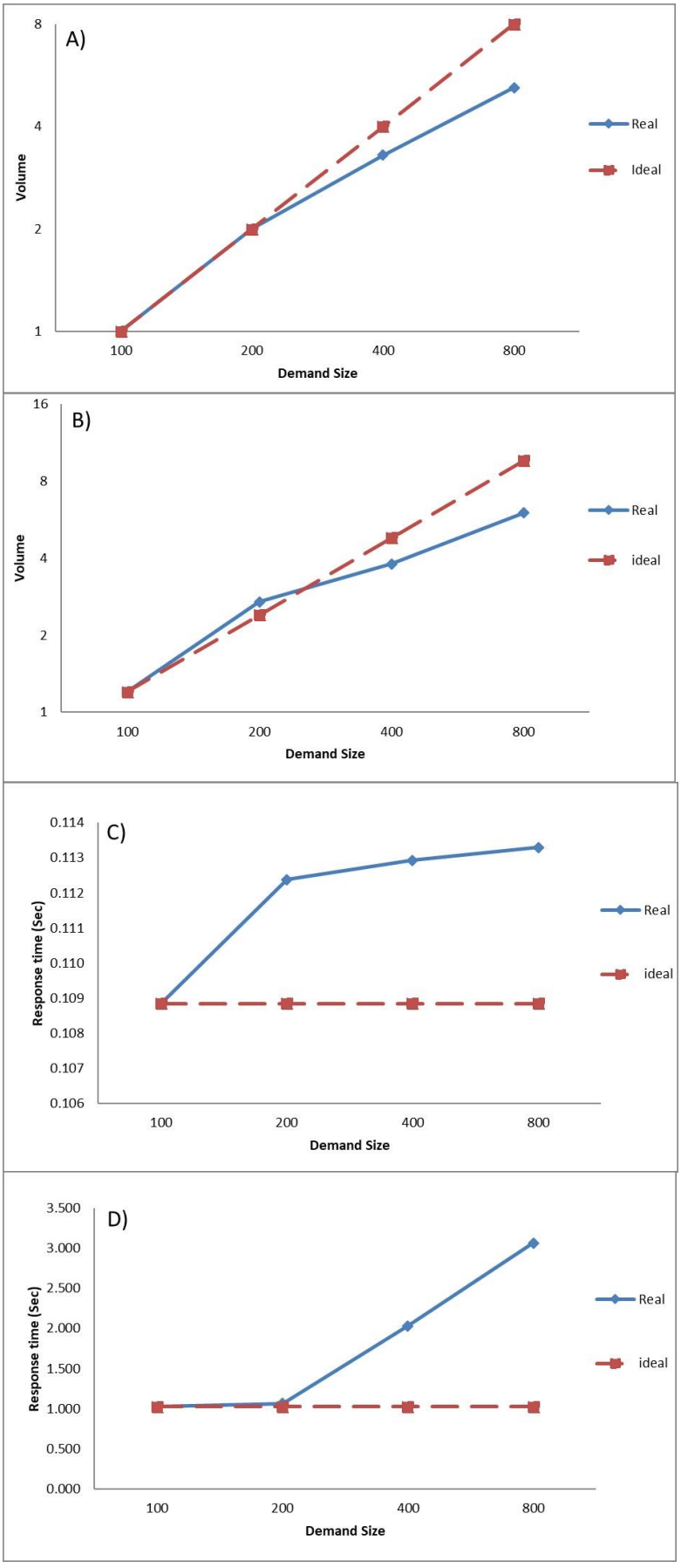

Figure 5.5: The average response times and number of software instances for MediaWiki in EC2. A,B) Average number of software instances- Steady rise and fall of demand scenario, Series of step-wise increases and decreases of demand scenario respectively. C,D) Average response times – Steady rise and fall of demand scenario, Series of step-wise increases and decreases of demand scenario respectively

It is to be noted that in the case of the MediaWiki a case of over-provisioning of software instances has been found, i.e. when the measured average number of software instances is larger than what would be expected as ideal performance according to equation (1) – see Figure5.5B. Given that this finding applies to a scenario with many stepwise up and down changes of the demand, a possible reason for what we found is the slow or delayed down-elastic response of the cloud platform. The volume performance metric does not account for over-provision as it assumes by default under-provision. Consequently, the over-provision, in this case, distorts somewhat the performance metric (increases it). One way to correct for this is to include a penalty for over-provisioning. Considering the symmetric nature of the deviation from the idea (downward or upward) in terms of its impact on the performance and on the geometric calculations in equation (2), this equation can be modified as follows:

$$A = \sum_{k=1,\ldots,n} (D_k - D_{k-1}) \cdot (I_k - 2 \cdot [\, I_k - I^*_k]_+ + I_{k-1} - 2 \cdot [\, I_{k+1} - I^*_{k+1}]_+) / 2 \qquad (7)$$

where $[x]_+$ represents the value of x if it is positive and 0 otherwise. This change of the calculation avoids the distortion of the metric caused by potential over-provision.

Table 5.5 shows the calculated values for the scalability metrics η_I and η_t for the two demand scenarios for both OrangeHRM and MediaWiki cloud-based systems. The corrected volume scalability performance metric, according to equation (7), for the MediaWiki for the second scenario is reported in Table 5.5 in italics.

Table 5.5: Scalability metrics values

Cloud-Based System	Scenario	Metric	
		η_l	η_t
OrangeHRM	Steady rise and fall	0.5687	0.9041
	Step-wise increase and decrease	0.5882	0.5201
MediaWiki	Steady rise and fall	0.7556	0.9664
	Step-wise increase and decrease	0.7421 *0.7183*	0.5012

The calculated metrics show that in terms of volume scaling the two scenarios give similar performance metrics for both systems. The scaling is slightly better in the context of the scenario with step-wise increase and decrease of demand for OrangeHRM. In contrast, for MediaWiki, the performance metrics indicate that the software performs slightly better in the first scenario, steady rise and fall of demand than in the second scenario. In terms of quality scalability, both systems scale much better in the context of the first scenario, steady rise and fall of demand, than in the case of the second scenario with step-wise increase and decrease of demand.

Comparing the two software systems running on the EC2, the metrics show that the MediaWiki runs at a considerably higher volume scalability performance than the OrangeHRM in both demand scenarios. The quality scalability metrics show at the MediaWiki has higher performance than the OrangeHRM in this respect in the first scenario and the performances are relatively close in this sense in the case of the second scenario. One possible factor behind the different volume scalability performance is that we ran the MediaWiki on t2.medium virtual machines, while the OrangeHRM was run on t2.micro virtual machines. Interestingly this

difference in the virtual machines made no major difference to the quality scaling of the two software systems. In principle, the difference in the volume scalability performance may point to the possibility that technical solutions in the MediaWiki system support more the volume scaling of the system than the corresponding solutions in the OrangeHRM. A deeper insight and investigation into the components of these systems responsible for the performance difference could deliver potentially significant improvements to the system with the weaker scalability performance metrics.

5.3.2.3 Results for The Same Cloud-Based Software System On EC2 with Different Auto-Scaling Policies

The same software configurations, hardware settings, and workload generator in this set of experiments have been used, to measure the scalability of the two scenarios for the same cloud-based software services that have been hosted in EC2, with different Auto-Scaling policies. The first set of policies are the default policies that are provided by EC2 cloud when setting up an Auto-Scaling group (option 1). For the second set of experiments, we select random scaling policies (option 2). The Auto-scaling policies that have been used for this set of experiments are given in Table 5.6.

Table 5.6: Auto-Scaling polices

Auto Scaling Policies		
Option 1	Add Instance	When 80% >= CPUUtilization < +infinity
	Remove Instance	When 30% <= CPUUtilization > -infinity

Option 2	Add Instance	When 70% >= CPUUtilization < +infinity
	Remove Instance	When 10% <= CPUUtilization > -infinity

The purpose of this kind of comparison is to see the effects on the scalability performance using the same cloud platform while using same types of instances and workload generators, with different auto-scaling policies. The average number of MediaWiki instances (Option 2) for both scenarios are shown in Figure 5.6A,B. The average response times of MediaWiki (Option 2) for both scenarios shown in Figure 5.6C,D. The average response times and number of software instances for MediaWiki in EC2 (Option 1) - see Figure 5.5.

As noted, there are two cases of over-provisioning of MediaWiki software instances for both 200 and 400 demand size, when we used new set of auto-scaling policies – see Figure 5.5B. Table 5.7 shows the calculated values for the scalability metrics η_I and η_t for the two demand scenarios for MediaWiki cloud-based systems for both auto-scaling policies options. The corrected volume scalability performance metric, according to equation (7), for the second scenario is reported in Table 5.7 in italics.

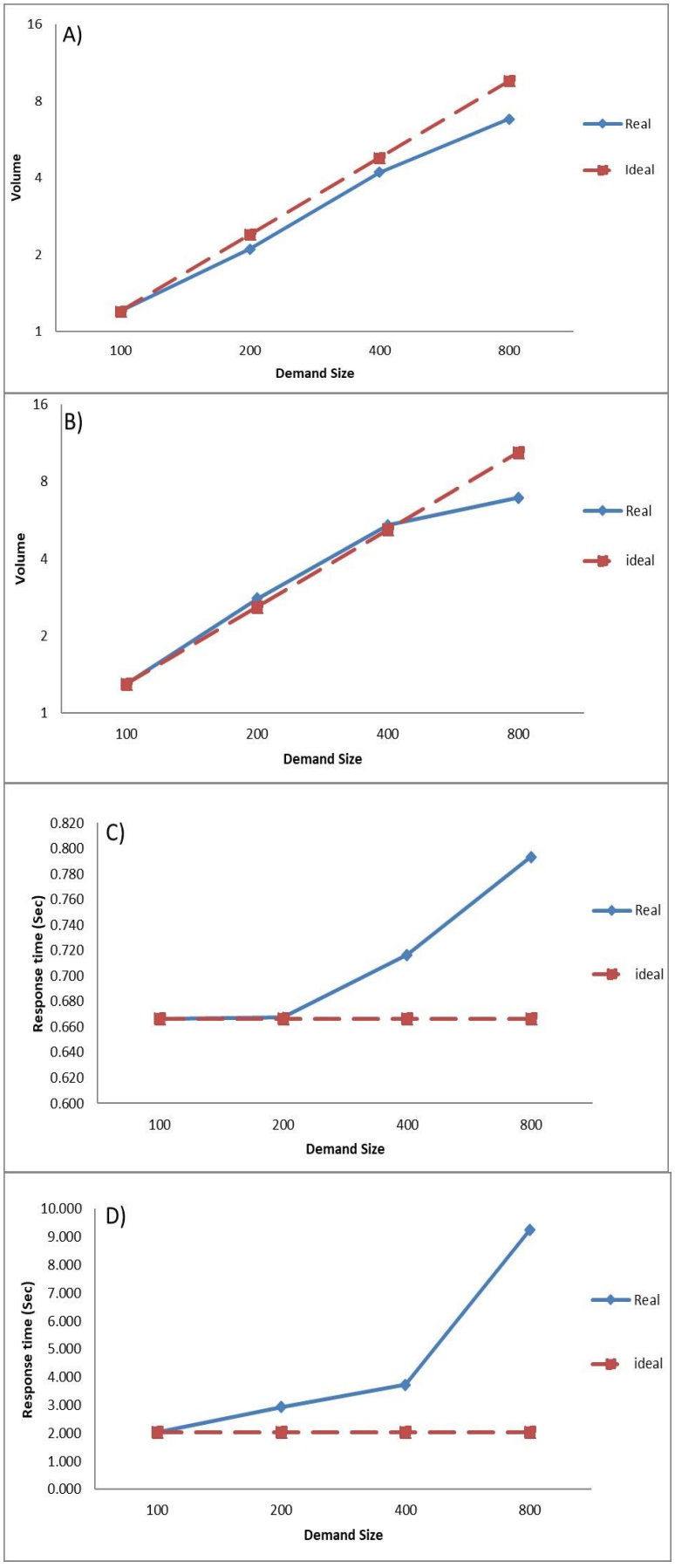

Figure 5.6: The average response times and number of software instances for MediaWiki in EC2 (Option 2). A,B) Average number of software instances- Steady rise and fall of demand scenario, Series of step-wise increases and decreases of demand scenario respectively. C,D) Average response times – Steady rise and fall of demand scenario, Series of step-wise increases and decreases of demand scenario respectively

In the terms of average response time, it is to be noted that there are big differences in the average of response times for the second scenario as it gradually changes from 2.035 seconds for demand size 100 to 9.24 seconds for demand size 800. While it gradually changes from 1.02 seconds for demand size 100 to 3.06 seconds for demand size 800, for the second scenario- Step-wise increase and decrease.

Table 5.7: Scalability metrics values

Cloud-Based System	**Scenario**	**Metric**	
		η_I	η_t
MediaWiki (Auto-Scaling policies option 1)	Steady rise and fall	0.7556	0.9664
	Step-wise increase and decrease	0.7421 *0.7183*	0.5012
MediaWiki (Auto-Scaling policies option 2)	Steady rise and fall	0.7923	0.9202
	Step-wise increase and decrease	0.8510 *0.8217*	0.4060

In terms of volume scaling that the experiments of MediaWiki with the second option of auto-scaling policies, increased 4% and 11% for the first and second scenarios respectively. While in terms of quality scaling the values decrease 4.5% and 10% for the first and second scenarios respectively. Comparing between the two options of auto-scaling policies, it is to be noted that the efficiency is increased when we used the default auto-scaling policies (option 1).

5.4 Discussion

In this Chapter an experimental analysis of scalability performance on two public clouds: Amazon EC2 and Microsoft Azure. Two cloud-based software services (MediaWiki and OrangeHRM) have been employed in order to demonstrate the scalability metrics that address both volume and quality scaling of such services and provide a practical measure of these features of such systems. Two demand scenarios to demonstrate the effect of demands patterns on scaling metrics have been used. Using more than one scenario can help to improve cloud-based software services to fit specified demand scenario expectations.

Three set of comparisons have been undertaken, this makes possible using the metrics to show differences in the system behaviour based on different scaling scenarios, configuration settings, or cloud platforms. This kind of comparisons provides the platform to construct the metrics as a basis that can be used effectively to compare the scalability delivery of cloud-based software services on different public clouds, and consequently supporting deployment decisions with technical arguments. In this chapter, the volume scalability metric has been altered to considers the over-provision case (see Subsection 5.3.2.2; equation number 7).

An interesting scalability behaviour has been noted through the analysis, such as big variations in average response time for similar experimental settings hosted in different clouds; OrangeHRM hosted on two different cloud platforms (EC2 and Azure), with the same hardware and experimental configurations.

In the third comparison group, a full analysis using the same software configurations, hardware settings, and workload generator to measure the scalability of the two scenarios for the same cloud-based software services (MediaWiki) that have been hosted in EC2, with different Auto-Scaling policies. It has been concluded there was no real impact caused by changing the auto-scaling polices on using on the scalability metrics.

The integrated scalability metric (see costs in Table 5.1) for the two demand scenarios for all cloud-based applications for both cloud platforms have been calculated using the formula introduced in Section 4.4.

$$P(\Lambda) = \eta I\,(\Lambda) \cdot \eta t\,(\Lambda) / c(\Lambda) \qquad (8)$$

by adopting $p(\Lambda) = \eta_I(\Lambda)$ and $\omega(\Lambda) = \eta_t(\Lambda)$ (see 4.4 for more details).

The values of the integrated scalability metrics are shown in Table 5.8 – note that the MediaWiki experiments used more powerful and more expensive virtual machines than the experiments with the OrangeHRM on the EC2. Our utility oriented scalability calculations show that in the case of the systems that we compared the best choice is to use smaller and cheaper virtual machines on the EC2. The corrected integrated scalability metric, based on equation (10), for the MediaWiki for the second scenario, is reported in Table 5.8 in italics.

Table 5.8: Integrated scalability metric values

Cloud-Based System / Cloud provider	Scenario	Values	
OrangeHRM / EC2	Steady rise and fall	38.95	
	Step-wise increase and decrease	23.18	
OrangeHRM / Azure	Steady rise and fall	4.93	
	Step-wise increase and decrease	2.21	
MediaWiki (Auto-Scaling policies option 1)	Steady rise and fall	14.04	
	Step-wise increase and decrease	7.15	*6.92*
MediaWiki (Auto-Scaling policies option 2)	Steady rise and fall	14.02	
	Step-wise increase and decrease	6.64	*6.42*

We believe that the technical-based scalability metrics can be used in designing and performing scalability testing of cloud-based software systems, in order to identify system components that critically contribute to the technical scaling performance.

5.5 Summary and Conclusions

In this chapter, we demonstrate the use of two technical scalability metrics for cloud-based software services for the comparison of software services running on the same and also on different cloud platforms. The underlying principles of the metrics are conceptually very simple and they address both the volume and quality scaling performance and are defined using the differences between the real

and ideal scaling carves. We used two demand scenarios, two cloud-based open source software services (OrangeHRM and MediaWiki) and two public cloud platforms (Amazon AWS and Microsoft Azure). Our experimental results and analysis show that the metrics allow clear assessments of the impact of demand scenarios on the systems, and quantify explicitly the technical scalability performance of the cloud-based software services. The findings of this chapter aim to address the thesis objective number 5, and also emphasis the findings of the previous chapter, in order to achieve objectives 2-4 (see Section 1.2).

Some interesting scalability behaviour has been noted through the analysis, such as big variations in average response time for similar experimental settings hosted in different clouds. A case of over-provision state has been accrued when using higher capacity hardware configurations in the EC2 cloud. This has been addressed by introducing a revised calculation for the scalability metrics that we use.In the next chapter, an in-depth investigation using application-level fault injection to measure the scalability behaviour of cloud-based software services under faults.

Chapter 6 Application-Level Fault Injection for Cloud-based Software Services

This chapter presents a preliminary investigation into the effect of faults on the scalability of cloud-based software services. The study introduces an experimental approach for Application-Level Fault Injection (ALFI) to investigate the how the faults at the application level affect the scalability behaviour of cloud-based software services. The previous chapters provided a baseline of the scalability behaviour of software services, which allows the researchers to conduct a more in-depth scalability investigation of such services. An experimental analysis on the EC2 cloud environment with a real-world cloud-based software service is used to demonstrate the approach, considering one type of software faults with two varied settings, and one demand scenario. The results of this preliminary study show how the proposed methodology can be used to assess the impact of injected faults on the scalability behaviour of cloud-based services.

6.1 Introduction

As cloud-based software services have become more popular and dependable, evaluating the performance of such services is more critical than before. Previous research studies have focused on the performance and scalability of such services to collect the right measurements and set up specific metrics such as technical evaluation metrics and infrastructure-monitoring metrics. These metrics are important to set a baseline for the performance behaviour of these services.

Performance and scalability assessment by using the fault injection technique allows evaluation of the impact of faults on the quality aspects of cloud-based software services, such as performance, scalability and security [19]. However, most studies focused on injecting the faults on the IaaS and PaaS level [125], [126], or introducing a test environment system that injects faults into hardware devices or VM levels [104].

Fault injection is an approach to test the performance of software systems [124], [157]. Fault injection can take place at different times: at runtime, compile-time or the loading time of external components [158]. Fault injection approaches have been used extensively to characterise the behaviour of systems under faults [125]. Fault injection has been used to analyse the dependability and reliability of cloud-based software systems [129], [130], [159].

Application-level fault injection (ALFI) is one of the most common techniques to study the application's resilience to faults [19]. It has been used to evaluate the

application's vulnerability [19] based on its application responses. Moreover, the ALFI technique is used for testing the application's resilience to ascertain how applications tolerate random instance failures [160], which is a discipline of experimenting on software systems' ability to tolerate failures in unexpected conditions that has been referred to as "chaos engineering"[161]. In this work, the focus will be on injecting the faults into the running cloud-based application by using fault injection tools to emulate potential problems at the application level to assess how the faults influence the scalability behaviour of the cloud-based software service.

The remainder of the chapter is structured as follows. Section 6.2 discusses the preliminary concepts and the approach of fault injection at the application level. Section 6.3 presents the result of an application example. This is followed by a discussion of the study in Section 6.4, including the implications, limitations, and importance of the work. The final section, Section 6.5, presents the conclusions and future directions.

6.2 Preliminary Concepts

This chapter aims to investigate the effect of runtime fault injection at the application level on the scalability performance of cloud-based software services. An Auto-Scaling service is used to support the software services to deal with the sudden workload, and a Load-Balancing service is used to determine the fault

tolerance of software services by ensuring that the incoming application's traffic is distributed across multiple application instances [143]. In Chapters 3, 4 and 5, studies investigate the scalability performance of cloud-based software services, which set a baseline for the scalability behaviour of those services. In the study reported in this chapter, the use of ALFI will provide data to compare the scalability performance with the baseline performance following the proposed scalability metric discussed in Chapters 4 and 5.

In general, the aim here is not to crash the application at runtime. In the construct, the methodology is measuring and evaluating the effect of the injected faults on the cloud-based software services' scalability over a sustained period. Here, the researcher will collect the measurements that have been defined in Section 4.2, the number of instances and average response time, to calculate the volume and quality scalability metrics. This will provide fair comparisons of the calculated average number of instances and average response time under normal operation and the behaviour of the two measurements during fault injection. This will provide useful behaviour benchmarking about the scalability performance that can be used to assess the impact of faults in the delivery of the cloud-based software service from a scalability perspective. Figure 6.1 illustrates the general concepts of the experimental approach.

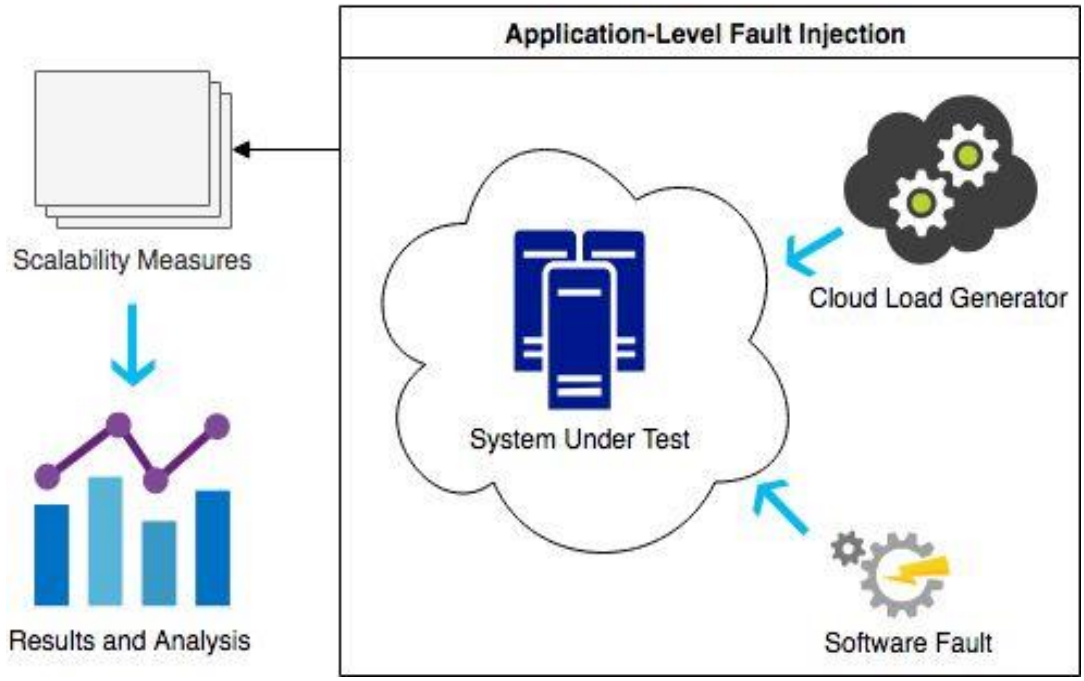

Figure 6.1: Experimental approach for application-level fault injection

This approach incorporates four main components: workload generator, software fault, scalability measures, and the system under test and its environment. A workload generator (such as JMeter or/and Redline13) is used to simulate a realistic demand scenario. A set of software faults should represent a repeatable and generally accepted set of faults (such as adding latency/bandwidth, HTTP traffic, database traffic, or terminating requests). The software fault is defined as *"An error is that part of the system state which is liable to lead to subsequent failure: an error affecting the service is an indication that a failure occurs or has occurred. The adjudged or hypothesised cause of an error is a fault."*[162]. Scalability measures are the indicators that are used to quantify the scalability of cloud-based software services. Finally, the system under test and its environment includes connecting both Auto-Scaling and Load-Balancing services to ensure the scaling provision of services.

One type of demand scenario is used in this chapter, the stepped increase and decrease scenario, with a set peak level of demand. With this scenario, the aim is to start with 10% of the demand size, then increase 10% stepwise over time, followed by a 10% stepped decrease over time. Figure 6.2 illustrates the demand scenario pattern.

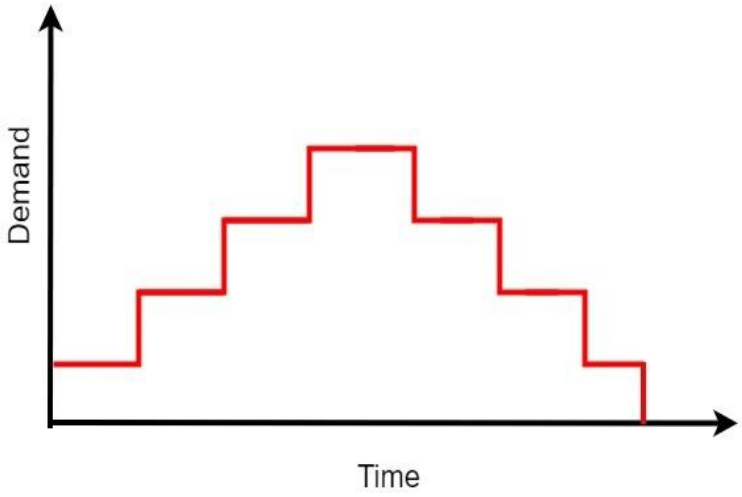

Figure 6.2: The stepped rise and fall of demand

6.3 Application Example and Result

To demonstrate the applicability of the ALFI experimental approach that was explained above, the approach was prepared in two stages. The first is preparing the workload scenario, scalability measures, and the system under test and its environment; the second is preparing the set of the software fault(s) which will be injected in parallel with the workload on the system under test.

6.3.1 The First Stage

In this stage, the testing methodology that has been presented in Chapter 3 is followed. The OrangeHRM service was hosted in the Amazon EC2 environment. An EC2 instance was set up and configured to host the targeted application through the EC2 management console. Both Auto-Scaling and Load-Balancing services were connected to the application instance, and the CloudWatch service to monitor the scaling parameters was attached to the software service. The parameters of the VM and the Auto-Scaling polices that were used for the experiments are shown in Table 6.1.

Table 6.1: EC2 virtual machine parameters and Auto-Scaling policies

Virtual Machine Parameters			
Instance type: t2.micro			
vCPUs	RAM (GiB)	CPU Credits/hr	Storage (GB)
1	1.0	6	10
Auto-Scaling Policies			
Add Instance	When 80% >= CPUUtilization < +infinity		
Remove Instance	When 30% <= CPUUtilization > -infinity		

To measure the scalability, the user demand scenario was simulated using the Apache JMeter script and run through Redline13 services after connecting the researchers' Amazon account to the service. The experimental data was collected through both Redline13 and Amazon's CloudWatch services. The service requests consisted of HTTP requests to the main page of the application by gaining login access using the Apache JMeter script (for more details, see Section 3.2).

Only one demand scenario was used in this chapter. The scenario consists of a series of stepwise increases and decreases in demand, conceptually similar to the demand pattern shown in Figure 6.2. Example of the experimental demand patterns (users running at runtime) are shown in Figure 6.3. These patterns were captured after applying the two-stage experimental approach. The volume of demand and experimented were varied with four demand sizes, with 100, 200, 400 and 800 service requests in total.

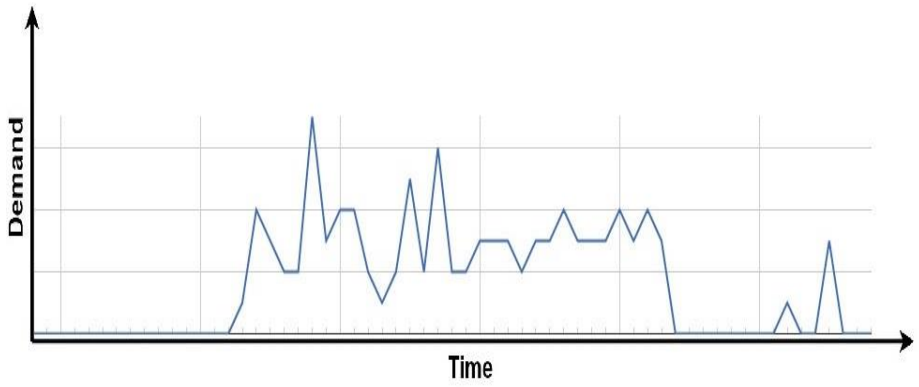

Figure 6.3: Typical experimental demand patterns: OrangeHRM/EC2 – series of stepwise increases and decreases of demand

6.3.2 The Second Stage

To simulate the injected faults, the experiment used Charles (https://www.charlesproxy.com/), which is an HTTP proxy; an HTTP monitor; a reverse proxy; and a web traffic simulator to simulate application latency (in milliseconds [ms]). The latency delay simulates the latency experienced on slower connections, which is the delay between making an HTTP request from the application side and the request being received at the cloud server side. In the

experiment, the delay latency times were varied: 800 ms and 1600 ms. For our purposes it was sufficient to simulate the latency delay using Charles, also we found this HTTP proxy easy to use, free and available. However, it should be noted that there are some other HTTP proxy alternatives include James, Fiddler, TinyProxy, mitmproxy etc.

6.3.3 The Measured Scalability Results

This section will present the scalability measures that were collected following the scalability technical measurements approach that was proposed and demonstrated in Chapter 4, Section 4.2. The baseline benchmark data was collected from the experiments without fault injection (this was conducted as part of Chapter 4). Both 800 service requests for 800 ms and 1600 ms delay latency experiments crashed as a result of "connection timed out". Table 6.2 shows the successful and failed experiments. In the detailed set of experiments, each fault injection experiment was conducted once, and this indicates that the values are not statistically significant. However, this was done to illustrate the impact of fault injection at runtime.

Table 6.2: The successful/failed experiments

	100	**200**	**400**	**800**
Baseline (without fault injection)	Successful	Successful	Successful	Successful
800 ms delay latency	Successful	Successful	Successful	Failed
1600 ms delay latency	Successful	Successful	Successful	Failed

The average number of software instances for each of the four demand levels is shown in Figure 6.4. The average response times for the four demand levels are shown in Figure 6.5.

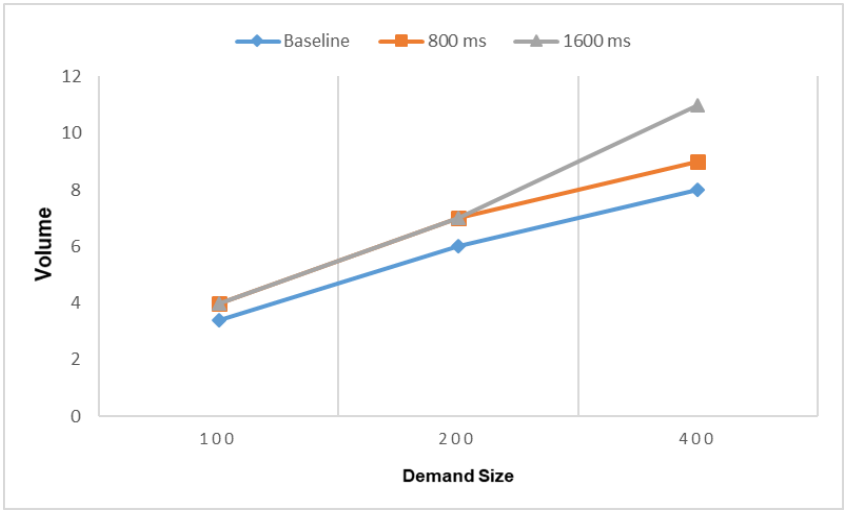

Figure 6.4: The average number of software instances for the baseline, 800 ms, and 1600 ms delay latency experiments

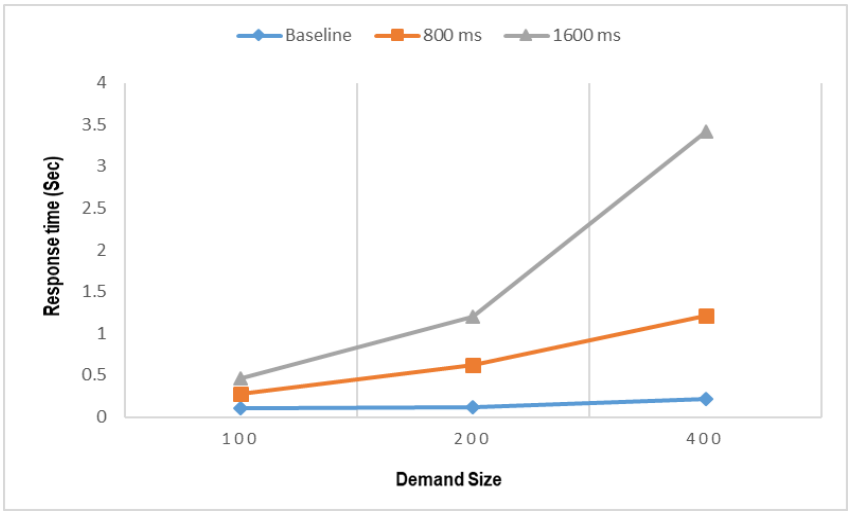

Figure 6.5: The average response times for the baseline, 800 ms, and 1600 ms delay latency experiments

It was noted that the average number of instances for the 800 ms experiments caused a similar scaling behaviour to the baseline, while in the case of the 1600 ms experiments, the behaviour changed by increasing the number of provisioned instances at the 400 service requests. In terms of quality, there was not a big variation in average response times in both cases (800 and 1600 ms). In the case of 800 ms, the scaling started increasing significantly from the demand size of 200, and then once the demand size reached 400, the average response time began to stabilise around the same pattern as the baseline. In contrast, the response time values for the 1600 ms experiment, which are shown in Figure 6.5, increased gradually with bigger variations.

This investigation of scalability performance is designed to determine the impact of using other ways to study the performance of cloud-based software services, such as using the fault-injection technique. Figure 6.6 shows the actual scaling of the 800 ms latency injection experiments compared with the ideal scaling behaviour in relation to both technical scalability metrics: volume (η_l) and quality (η_t) calculation (see Section 4.2 for more details), while Figure 6.7 illustrates the ideal and actual scaling behaviour of the 1600 ms latency injection experiments. Both technical scalability measures are shown in terms of the average number of instances and average response times. Here, here we compare the ideal scaling of the baseline experiments with the actual scaling behaviour of the latency injection experiments.

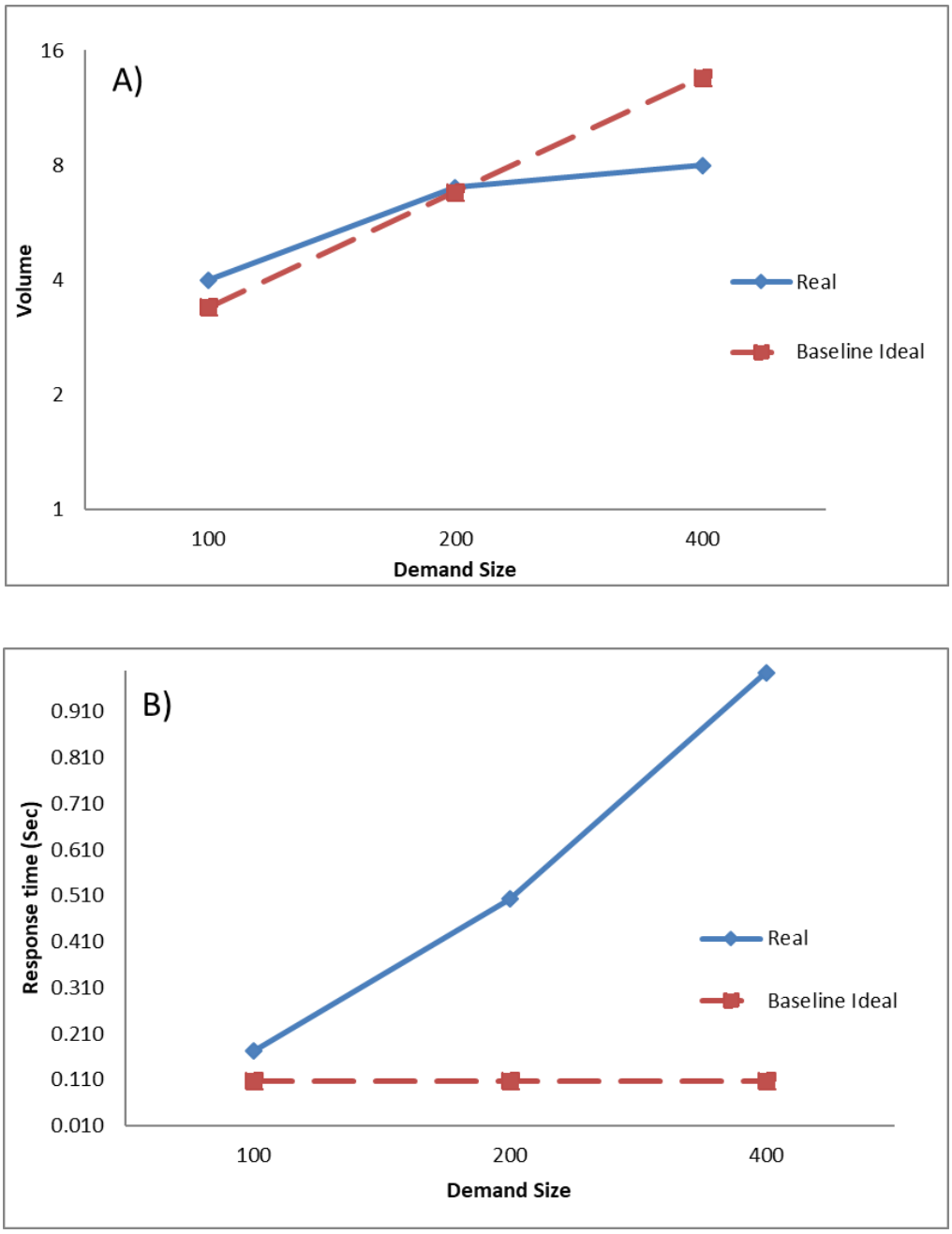

Figure 6.6: The average number of software instances and response times for 800 ms experiments. A) Average number of software instances. B) Average response times

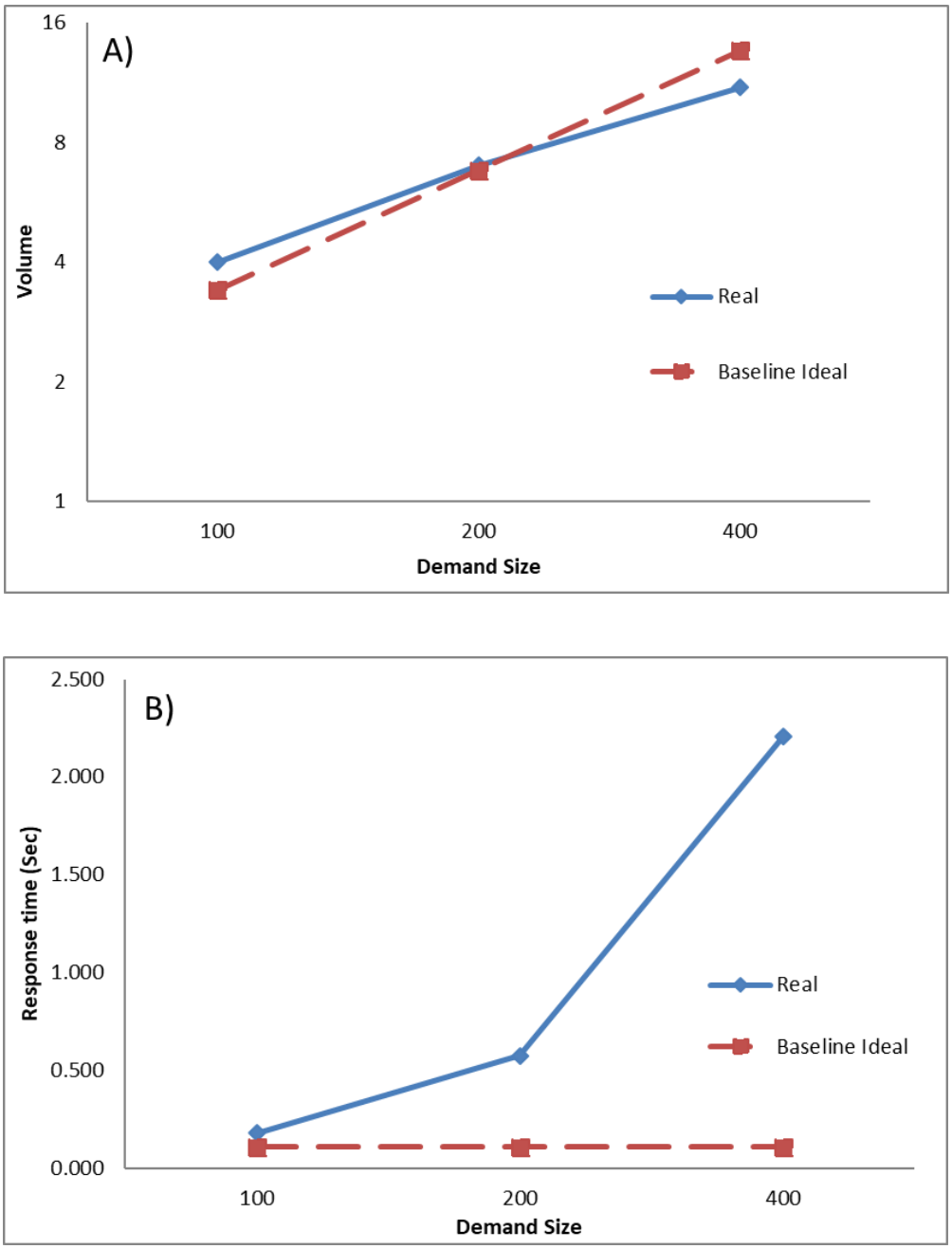

Figure 6.7: The average number of software instances and response times for 1600 ms experiments. A) Average number of software instances. B) Average response times

The values for the scalability metrics η_I and η_t for the baseline (see Section 4.3 for more details) and the two fault injection experiments that were conducted are shown in Table 6.3. The calculated metrics show that in terms of volume scalability, the fault injection experiments display over-provisioning behaviour, with notably decreased in the volume performance in the 1600 ms experiment. The calculation of the volume metric values for the fault injection scaling behaviour is based on the altered metric that considers the over-provision (see Subsection 5.3.2.2; equation number 7). The reason for this is that part of the volume results are equivalent of over-provision according to our definition of this (i.e. see Figures 6.6A and 6.7A for demand size 100).

Table 6.3: Scalability metrics values

Scenario	**Metric**	
	η_I	η_t
Baseline (without fault injection)	0.5882	0.5201
800 ms delay latency	0.4706	0.1752
1600 ms delay latency	0.2353	0.1019

In terms of quality scalability, the system scales much better in the context of the baseline compared to the fault injection experiments. It was noted that as a result of the variations in response times for the 1600 ms experiments, the quality metric value dropped by 0.4182, a percentage decrease of 80.41%. It was also noted that by using 1600 ms latency injection, the volume decreased as expected; however, the quality dropped significantly. If the decrease in quality and volume scaling is taken into account, this shows that the overall performance of the scalability has dropped.

It should be noted that there is a negative impact caused by the latency faults in terms of quality. While volume is decreased between 19.99% and 60% in relation to the baseline, the quality indicator shows that there is a significant drop in the performance of the services between 66.31% and 80.41%.

6.4 Discussion and Limitations

In this chapter, a preliminary experimental analysis of the impact of fault injection on the scalability performance of cloud-based software services has been presented. The experimental approach based on the use of the ALFI, has been explained, combining four components: workload generator, software fault, scalability measures, and the system under test and its environment. Previous studies on the scalability performance of cloud-based software services provide a baseline for the scalability behaviour of those services. An example using Amazon EC2 and a cloud-based software service (OrangeHRM) has been employed to demonstrate the approach using delay latency injection with two different times, 800 and 1600 ms, and the data has been compared with the baseline data from the previous studies (see Chapter 4). This is important to determine whether the fault injection experiments have a significant impact on the scalability performance of the software service. It should be noted that a negative impact is caused by the delay latency faults in terms of quality. Moreover, while the volume scaling is decreased in relation to the baseline, the quality indicator shows a significant drop in the performance of the service in terms of quality.

In this chapter, the fault injection is considered through the injection of delay latency into the software service at runtime. Other faults at the application level could also be considered to assess the true impact of faults on the scalability of cloud-based software services, and to ascertain the type of impact on the scalability based on the nature of the fault. This would provide useful behaviour benchmarking in relation of the scalability performance that can be used to assess the impact of faults on the delivery of the cloud-based software service from a scalability perspective. This could help to identify likely problems with the software or the cloud environment that deliver the cloud-based software service.

Expanding the range of faults provides better benchmark data and a more comprehensive picture of the performance of the scalability of cloud-based software services under fault scenarios and techniques. Here, only one demand scenario was used to demonstrate the effect of demand patterns in the fault injection approach. In principle, various demand scenarios may be used to fine-tune the cloud-based software service to fit particular demand scenario expectations. Similarly, considering a set of fault injection incorporated with different demand scenarios can also be used to identify changes in such scenarios or faults that trigger interventions in terms of software upgrades or maintenance for the targeted system.

The limitations of the results presented here stem from the limited nature of the experimental investigation. First, the fault injection experiments were conducted once, and this indicates that the values are not statistically significant (i.e. tests

should be repeated 20 times to say that the result can be considered as benchmark data; if the tests rely on collecting a performance indicator, the value of one test should be obtained and compared exactly with previous tests). However, the experiments were conducted to illustrate the impact of fault injection at runtime. Furthermore, only one cloud platform (EC2) was used with OrangeHRM to demonstrate the application and usefulness of the proposed ALFI approach. Naturally, expanding the experiments to cover multiple cloud platforms and multiple cloud-based software services would provide a better picture of the impact of the fault on the scalability of such services. Moreover, only one demand scenario and fault were used, whereas a wider range of these would offer a deeper understanding of how the proposed approach varies depending on the demand scenarios and the nature of faults. Finally, one particular setting of the cloud service (i.e. VM specification), one load and one fault generators were used to implement the ALFI approach to investigate the scalability of cloud-based software services. Alternative load and fault generators might have an impact on the values of the calculated metrics due to their implementation details, although in principle, it is not expected that these would have a major impact on the reported results.

6.5 Conclusions and Future Directions

In this chapter, a preliminary experimental approach of ALFI to investigate the scalability of cloud-based software services is presented. The experimental

approach is explained, combining four components: workload generator, software fault, scalability measures, and the system under test and its environment. The proposed approach was demonstrated using a cloud-based software service run on the Amazon EC2 and considering one demand scenario and one type of fault. The preliminary results show that the proposed approach allows clear assessment of the impact of a fault scenario on the cloud-based software service's scalability performance. The findings of this chapter aim to address objective number 6 of the thesis (see Section 1.2).

A major part of the method implemented in the ALFI approach is derived from the findings of previous studies reported in this thesis. For instance, scalability analysis was used in both Chapter 4 and Chapter 5, the first stage of the methodology of the approach. Furthermore, the results of the studies are used as a baseline to draw comparisons with the result of the fault injection experiments to assess the impact of this methodology.

Future work will include the consideration of other cloud platforms (e.g. Azure, Google Cloud, and IBM), other demand workload generators and fault generators, more software faults and other cloud-based software services to obtain a wider range of measurements of the proposed approach, extending the practical validity of the work. Moreover, the researchers aim to consider further demand patterns incorporated with faults to show how these impact on the scalability performance of cloud-based software services. This could help to establish volume and quality scalability metrics conditional on fault injection patterns.

Chapter 7 Discussion

In this chapter, the findings from all the research reported in this thesis are brought together and discussed in relation to the original research objectives and questions.

7.1 Introduction

Although there are cloud-monitoring tools, such as CloudWatch in Amazon EC2 and Azure Monitor in Microsoft Azure, which enable consumers to collect some indicators about the usage of cloud computing resources, there is a lack of analytic metrics supporting the technical analysis of cloud-based software services' performance and scalability. As discussed in this chapter, existing results address the scalability measurements and metrics issue from a technical perspective. Such research is important to support the SLA and the future optimisation of cloud computing, especially to support the delivery of the software services model SaaS in the cloud.

A novel approach to measure and quantify the scalability of such services has been proposed, and the explanation of two technical metrics based on the measurement

approach is detailed in Chapter 4. The metrics address both the volume and quality scalability of the services, and provide a practical measure of these features of cloud-based software services. Thus, these metrics are distinct from elasticity-oriented metrics proposed in the field. The volume (η_l) and quality (η_t) scalability metrics are based on the number of software instances and the average response time. Employing an application on EC2 using a real-world software service (OrangeHRM), 240 experiments were conducted and analysed to demonstrate the applicability of the scalability metrics, which is presented in Chapter 4. To extend the applicability of the metrics, another 640 experiments were conducted on two public clouds (EC2 and/or Azure) using two cloud-based software services (OrangeHRM and/or MediaWiki) considering three kinds of compression scenarios, which is presented in Chapter 5. This provides the baseline analysis of the scalability performance of cloud-based software services to determine the impact of using other ways to study the performance of such services, such as using the fault-injection technique. Consequently, without collecting the right technical measurements incorporated into the right metrics, it will be difficult to determine if the fault injection experiments have exerted a real impact on the scalability performance. A case study of fault injection at the application level is presented in Chapter 6.

This thesis investigates scalability performance from the perspective of cloud-based software services delivery without aiming to analyse, design or improve the underlying cloud infrastructure technology or cloud software development

platforms. Four research questions were listed in Chapter 1 to be answered to achieve the objectives of this research.

The discussion illustrates that the contribution of this thesis is to investigate the area of scalability performance of cloud-based software services. In this chapter, using the novel approach of evaluating the scalability of such software systems will be discussed to answer the research questions. In Section 7.2, the response to RQ1 is based on the findings from this research. The work undertaken in relation to the technical measurements to respond to RQ2 is discussed in Section 7.3. In response to RQ3, the work undertaken to propose and demonstrate the technical scalability metrics is presented in Section 7.4. A discussion in relation to RQ4 is presented in Section 7.5. Comparing the proposed metrics with clostest related work is outlined in Section 7.6. A summary of research contribution is presented in Section 7.7. The technical scalability metrics deployment challenges is presented in Section 7.8. The research limitations are outlined in Section 7.9. Finally, the chapter concludes with a summary in Section 7.10.

7.2 Testing the Scalability of Cloud-based Software Services

RQ1: How can we test the scalability of cloud-based software services?

In this section, a discussion of the work undertaken to investigate the current scalability testing of cloud-based software services is presented. The literature review reported in Chapter two was undertaken in two stages. In Section 2.1, a

systematic mapping study was undertaken to identify the testing methods involved in the area of cloud software testing, whether these methods were applied, and what was being tested. This provides the overview map of the differences of using the cloud as a tool for testing, testing the cloud, and how to test different cloud services (infrastructure, platform, and software). The second stage reviews the current practice of scalability testing of cloud-based software services (see Section 2.2). These two stages established the need for an investigation into the scalability of cloud-based software services. The result of the two-stage literature review, show that most work presents early results and their authors expect to carry their work forward on to more extensive studies.

Subsequent to the literature review, a test plan was developed to outline the operational aspects of executing the scalability testing strategy to collect the right measurements of scalability performance. This plan follows IEEE 829 standards, as outlined in Section 3.2. The plan describes the test process in detail, including test items, approach, tasks, deliverables, and the environment required. The plan was implemented to ensure that we collect the right measurements of scalability performance of such services.

The test strategy that has been set out here focuses on combining the services that cloud providers offer to support the scalability performance of application services, such as Auto Scaling and Load Balancing services, in the test plan. The reason for using these two cloud-based software services (OrangeHRM and MediaWiki) is based on the REST-based nature of the applications, which is highly

adopted by cloud and application providers. As the architecture of these applications support REST caching to improve performance; by caching the data and the code, which will reduce the amount of time required to execute each HTTP request and therefor reducing the CPU usage. Different demand (workload) scenarios help to highlight the difference in scalability behaviour.

To have a plan that contains details and the scope of the scalability testing, will help to conduct the test activities in a more efficient and effective way. This will provide more reliable results that are more consistent over time and more representative to the real scalability behaviour of cloud-based software services.

7.3 Technical Scalability Measurements of Cloud-based Software Services

RQ2: What do we measure in relation to the technical scalability of cloud-based software services?

Technically oriented measurements for cloud-based software scalability research are limited, as shown in the literature review updates (see Section 2.2). This was investigated further to answer what we measure in relation to the technical scalability of cloud-based software services in Chapter 4 and Chapter 5. Section 4.2 presented a novel approach to measure and quantify the scalability of such services based on the number of software instances and average response time. The first measurement is based on an average number of software instances that

have been deployed over a sustained period of service provision. The other measurement is based on the system quality scalability by measuring the service average response times corresponding to the demand level(s). This means that the measurements in this research based on an average number of software instances and average response time were measured regularly during the execution of a demand scenario following a particular pattern of demand variation. These measurements were collected as a result of the testing methodology described in Chapter 3.

In the thesis the system quality scalability has been measured by the service average response times, however, other aspects of quality scaling could be also used to define further similar but functionally distinct technical quality scaling or a combination between two or more quality measurements.

Measuring and quantifying the scalability of cloud software services from a technical perspective is important to understand the system's components that affect and contribute to the scalability performance of the software service. This could help to design suitable test scenarios and provide a basis for future studies aiming to maximise the scalability performance. Collecting the right measurements is important in order to incorporate into the right metrics; this will ensure a consistent interpretation of the fine-grained scalability measurements data through the lenses of relevant scalability metrics. This interpretation will enable better understanding of the factors that influence the scalability of cloud-

based software services and will help practitioners and consumers to fine-tune such services to achieve better performance.

7.4 Technical Scalability Metrics of Cloud-based Software Services

RQ3: How do we interpret the technical scalability performance measurements?

Despite the need to interpret the technical scalability measurements into the right metrics, our literature review (Section 2.2) has signified that most studies do not provide any specific technical scalability metric. However, Chapter 4 of the thesis provides an explanation of two technical metrics based on the measurements approach: the volume (η_l) and quality (η_t) scalability metrics based on the number of software instances and the average response time. The underlying principles of the metrics are conceptually very simple. They address both the volume and quality scaling performance and are defined using the differences between the real and ideal scaling behaviour curves. The original volume scalability metric explained in Section 4.2 only considers the under-provision case of scaling. However, in Subsection 5.3.2.2, we altered the metric to consider over-provisioning behaviour of scaling (see Figure 7.1).

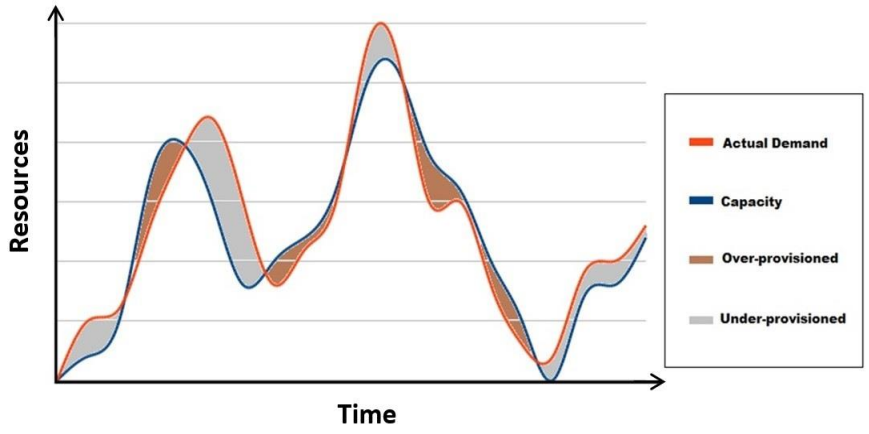

Figure 7.1: Scalability over-provisioning case

To validate the applicability of the metrics and measurements, in Section 4.3 we used a cloud-based software service (OrangeHRM) hosted in Amazon EC2 and considered three demand scenarios. The results show that the metrics quantify explicitly the technical scalability performance of the software service and that they allow clear assessment of the impact of demand scenarios on this service.

In section 5.3, a group of experimental analysis was undertaken to use the metrics to highlight differences in the cloud-based software services' behaviour based on different cloud platforms, scaling scenarios, hardware settings, and auto-scaling polices using two demand scenarios, two cloud-based open source software services (OrangeHRM and MediaWiki) and two public cloud platforms (Amazon AWS and Microsoft Azure). We performed three comparisons, with the first comparing the same cloud-based software service hosted on two different public cloud platforms. The second compares two different cloud-based software services hosted on the same cloud platform. The third compares the same cloud-based

software service hosted on the same cloud platform with different auto-scaling policies. Such comparisons will not only provide an extension of the practicality of the metrics, but also provide the platform to construct technical metrics that can be used effectively to compare the scalability delivery of cloud-based software services in different public cloud environments and support deployment decisions.

To achieve the research objective (see Section 1.2) to enable the scalability analysis from technical and utility-oriented perspectives, Sections 4.4 and 5.4 show how to integrate the technical scalability metrics with an earlier utility-oriented scalability metric and calculate the values for each demand scenario.

Such analysis of scalability performance of those systems, can drive the design of scalability tests, system revision and upgrade focused on improvement of scalability, or development of fine-grained monitoring of system' scalability performance. This investigation of realistic scalability performance can help to estimate the expectations of the system depending on demand scenarios and cloud platforms.

7.5 Cloud Software Services Scalability Assessment using Fault injection

RQ4: How can faults affect the scalability of cloud-based software services?

The findings from the literature review in Chapter 2, the scalability of cloud-based software services assessment in Chapter 4 and the comparisons assessment in Chapter 5 contributed to the type of information collected and made available to provide the baseline analysis for more in-depth investigation into the scalability issue of cloud-based software services.

Chapter 6 presents a preliminary experimental analysis of application level fault injection (ALFI) to investigate the scalability performance of cloud-based software services has been presented. In section 6.2 the experimental approach has been explained, combining four components: workload generator, software fault, scalability measures, and the system under test and its environment. A major part of the methods implemented in the ALFI approach is informed by the findings of studies reported in this thesis. For instance, scalability analysis was used in both Chapter 4 and Chapter 5, the first stage of the methodology of the approach. Furthermore, the results of the studies are used as a baseline to draw comparisons with the result of the fault injection experiments to assess the impact of this methodology.

In Section 6.3, The proposed approach was demonstrated using a case study by hosting OrangeHRM on the Amazon EC2, considering one demand scenario, and one type of fault. We simulate a delay latency injection with two different times; 800 and 1600 ms, and compared the data with the baseline data. The preliminary results show that the proposed approach allows clear assessment of the impact of a fault scenario on the cloud-based software service's scalability performance.

Furthermore, the results illustrate that a negative impact is caused by the delay latency faults in terms of quality. Moreover, while the volume scaling is decreased in relation to the baseline, the quality indicator shows a significant drop in the performance of the service in terms of quality.

This kind of investigation is important in order to determine how the software systems behave in the face of the injected deliberate faults. However, to formalize a full picture regarding the fault injection impact on the scalability of software services running on the cloud, the need for more experimental analysis involves other fault types, demand scenarios, and cloud platforms are necessary. Thus, the experiments presented in the case study can be considered as an example to illustrate the impact of injected fault on scalability. Therefore, the need for further investigation and the extension of the experiments is necessary to fully answer the question about *how can faults affect the scalability of cloud-based software services*.

7.6 Compare the Technical Scalability Metrics against Related Work

Software metrics that concern the scalability of cloud-based software services are limited (see Section 2.2), especially those specializing in measuring such services from a technical perspective. As appears in Table 2.4 of Section 2.1, scalability testing has been used in 17 empirical work, which this work is considers as the testing method applied, however, most of these studies were related to test the

scalability in/on the cloud/cloud services, or using the cloud as a tool for testing. The majority performed scalability testing for PaaS, IaaS, mobile applications, or web applications. Also, as shown in Table 2.5, only five studies focus on testing on cloud-based software services or SaaS, which is the focus of this thesis. However, only two studies were focusing on scalability testing, those studies were discussed further in Section 2.2, other related literature. As shown in the literature review discussion section 2.3, we found that most of related work did not clearly formulate a specific technical metric of scalability, and only presented scalability measurements relying on some scalability indicators (i.e. Jayasinghe et al. [71], [72] provides a technical scalability measure in terms of throughput and CPU utilization of the VMs, but the work does not provide a specific metric).

On the other hand, during the additional literature review, we have located other related work concerning the scalability of cloud-based software services or SaaS. The review shows that most work presents early results and their authors expect to carry their work forward on to more extensive studies, or the work focuses on measuring the elasticity of cloud software services from a technical perspective. Furthermore, an alternative utility-oriented approaches found in the literature for the measurement of the scalability of cloud-based software services. We have located an earlier utility-perspective scalability metric [15], [18], this work compares the proposed technical scalability metrics with their scalability metric, as a result of this comparison and the importance of the utility analysis of scalability, we integrated the metrics into our metrics, this is explained in the discussion section of Chapter 4 (see Section 4.4). In relation to the other related work, there

were no scalability metrics found for measuring cloud-based software services from a technical perspective.

7.7 Thesis Contributions

This thesis reports a novel investigation into the issues of scalability measurements and testing of cloud-based software services from technical perspective. Specifically, the technical measurements that contribute to the scalability performance of such services, and how to quantify and interpret those measurements into the right technical metrics. More details about how specific units of the work have contributed to knowledge in this area were discussed in details in Section 1.6. 1. Therefore, the list below summarizes and identifies the contributions individually:

- A mapping study, reported in Chapter 3 was the first in the field to investigate the empirical studies on software cloud testing methods. The study examined the method and application of functional and non-functional cloud software testing methods, 69 primary studies were analysed for building of classification scheme.

- The novel quantifying and measuring approach is used to investigate the scalability issues of cloud-based software services. The work explains the volume and quality scaling metrics for evaluating cloud-based software services' scalability performance based on the measurement approach. This

work introduces the demand scenarios and demonstrates a practical example of the metrics. This work established the need to determine how the technical scalability metrics can be integrated into an earlier utility-oriented metric of scalability. (Chapter 4)

- Three sets of experimental analysis in order to extend the practicality of the measurement approach and metrics, by comparing the scalability performance in two cloud platforms: Amazon EC2 and Microsoft Azure using two cloud-based software systems. The work not only provides an extension of the applicability of the metrics, but also provides the platform to construct the technical scalability metrics as a basis to effectively comparing the scalability of software on cloud environments, and supporting deployment decisions with technical arguments. (Chapter 5)

- A case study of application level fault injection (ALFI) testing for measuring the scalability of cloud-based software system, using Amazon EC2. An experimental approach has been explained. Here we simulate delay latency injection with two different times; 800 and 1600 ms, and compared the results with the baseline data. (Chapter 6)

7.8 Technical Scalability Metrics Deployment Challenges

The experimental analysis reported and conducted in this thesis, were based on real cloud environments and using real applications, however, any proposed

metrics should consider real cloud deployment platforms. Therefore, there are some potential challenges in regards to the deployment of the metrics in a cloud production environment:

One of the main challenges is to convince cloud providers that these metrics are useful and help them to understand and fine-tune better their services; another challenge is that the proposed metrics require a scenario-based testing of the service, which may not be commonly accepted at present, so the service providers need to be convinced that this is useful for them. Another further challenge is if the software services provider is different than the cloud IaaS provider, so some of the information needed for applying the proposed metrics could be not available to the software service provider.

On the other hand, there are issues that should be taking into account if any instrumentation is suggested or required for further testing that may be prohibitive because of the extra work and slow down effect on the service, however, we can ensure that the slowdown is minimized and the extra work can be done in a well-planned efficient way, so the extra cost is minimized.

7.9 Research Limitations

In this section, the limitations of this research are presented and classified into two subcategories: literature review limitations; and experimental execution limitations.

The initial literature reviewed in this project is presented as a systematic mapping study, using mainly a manual search, and therefore, there is a possibility that not all relevant studies were located. Using specific inclusion and exclusion criteria could also affect the location of the relevant studies, as some studies will be excluded in the last stage of the study selection process. However, every effort was made to ensure that the review covers all available literature up to the writing of this thesis by checking continually for any possible new publication. Snowballing technique was employed to check the reference list of the new articles identified in the updated literature search for any missed publication. We checked that all the primary studies reported in the identified published reviews were located by the search process and either complied with the inclusion criteria or were excluded based on the exclusion criteria. The same process was adopted in the case of the additional literature review.

In this thesis, as all the experiments were conducted based on real cloud environments and using real applications, each experiment needs on average of one hour, without considering the management time for uploading the software services to the cloud accounts, setting up the auto-scaling and load-balancing management settings. This limited our options to expand our experiments to more public cloud environments, software services and more instance types (VMs), which reflect on the decision to run the experiments on two public clouds and using two real-world software services only.

7.10 Summary

This chapter aims to clarify the extent to which the different work undertaken in this research has been able to answer the research questions, and how the findings address the research objectives set out in Chapter 1, Section 1.2. The testing of the scalability (Chapter 3), the technical measurement approach (Chapter 4), and the technical scalability metrics of cloud-based software services (Chapter 4 and Chapter 5) have been explained. These studies formed a baseline to undertake more in-depth studies related to the scalability issue of SaaS. Chapter 6 introduces an initial investigation using the fault injection technique, which still requires further work to establish volume and quality scalability metrics conditional on fault injection patterns. The comparison between the proposed metrics and related work in relation of the systematic literature review (section 2.1) and the additional literature were discussed. The research contributions were summarised and identified, some possible challenges for technical metrics on cloud environment were discussed. The research limitations were discussed. These can be categorised into those involving the literature review process and those related to the experimental execution. In the next chapter, a summary of the work and its conclusions is provided, and the chapter concludes with suggestions for future work.

Chapter 8 Conclusions and Future Directions

This chapter provides a summary and the conclusions of the research into scalability measurements and testing of cloud-based software services. Following this, some future directions for research are suggested.

8.1 Summary and Conclusions of the Research

The main aim of this research was to investigate the scalability measurements and testing of cloud-based software services. A novel approach of measuring and quantifying the scalability of cloud-based software services from a technical perspective was proposed. Two scalability metrics were introduced and an experimental analysis was conducted to demonstrate the applicability of the approach.

The first stage of this PhD project is the literature review that was undertaken to identify the current empirical practice of cloud software testing methods and the mapping study that was conducted to identify and classify cloud testing methods, the application of these methods, and the purpose of testing using these methods. The mapping study located 75 papers with related studies, which indicates the

growing interest across the field of cloud-related testing and the potential for much more research to follow the early results. As a result of the mapping study, and the business and technological importance of the scalability of cloud-based software services, the decision was made to investigate this area in detail. An additional search of related studies on technical scalability analysis of cloud-based services shows that most works present early results and their authors expect to continue their work in more extensive studies.

Following the literature review, a scalability test plan was developed to collect and monitor the right indicators from the scaling behaviour of the system under test. The plan details the testing items, approach, suspension criteria, test deliverables and tasks, and environmental needs.

A novel approach to measure and quantify the scalability of cloud-based software services based on the number of software instances and average response time was discussed. Two scalability metrics were explained depending on the technical approach: the volume (η_l) scalability metric based on the number of software instances, and the quality (η_t) scalability metric based on the average response time. The underlying principles of the metrics are conceptually very simple. They are defined using the differences between the real and ideal scaling behaviour curves. To demonstrate the use of the metrics, an application on EC2 and using OrangeHRM, using three demand scenarios, has been discussed. The results show that the proposed metrics quantify explicitly the technical scalability performance

of the system, and that they allow clear assessment of the impact of demand scenarios on the cloud-based software service.

To extend the applicability of the metrics and measurements approach, another set of experimental analyses were undertaken, considering three sets of comparisons, using two public clouds (Amazon and/or Azure) and two real-world software services (OrangeHRM and/or MediaWiki), and considering two demand scenarios. The results show that the metrics can be used effectively to compare the scalability of software on public clouds and consequently to support deployment decisions with technical arguments.

The technical metrics were integrated with an earlier utility-oriented metric to enable analysis of the scalability behaviour and delivery from both economic and technical viewpoints.

The results reported for the undertaken studies indicate some interesting scalability behaviour, such as big variations in average response time for similar experimental settings hosted in different clouds. A case of over-provisioning occurred when using higher-capacity hardware configurations in the EC2 Cloud. This was addressed by introducing a revised calculation for the volume scalability metrics.

The above research has provided a valuable insight into the scalability delivery of cloud-based software services. This helps to understand better the scalability behaviour of these services. The application-level fault-injection technique presented in Chapter 6 provides an initial assessment of how faults affect the

software services' scalability behaviour. The results show an impact of the injected faults on the behaviour of the scalability of those software systems.

8.2 Future Research Directions

This thesis identifies a number of future directions to further consolidate the effectiveness of testing and measuring the scalability of cloud-based software services and the technical scalability metrics. These are described below.

1- Future work will include the consideration of other public cloud platforms (e.g. Google Cloud, IBM), private cloud platforms, demand workload generators, VM specifications, and other cloud-based software services to extend the practical validity of the work. The work will consider further demand patterns (such as variable width sudden peaks in demand, seasonal demand) to determine the impact of these scenarios on the scalability performance of cloud-based software services.

2- The cloud serverless execution model (i.e. AWS Lambda, and Azure Functions) can be used to assess the scalability of cloud-based software using this delivery model, which may become more widely used in the future.

3- The fault-injection analysis of the cloud-based software services will be extended by considering more faults. This will provide useful behaviour benchmarking in relation with the scalability performance. This will be used

to assess the impact of faults in the delivery of the cloud-based software service from a scalability perspective. This can help to identify likely problems with the software or the cloud environment that deliver the cloud-based software service, which can then be followed by the addressing of the estimated problems.

4- Another aspect of future work will focus on using the whole code instrumentation technique to identify the software system or cloud platform components that contribute critically to variations in average response times for the same cloud-based software service with similar experimental settings in different public clouds. Using code instrumentation will enable monitoring of the scalability delivery behaviour of such services to support the identification of scalability bottlenecks within the software services.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[2] C. Jia, Y. Cai, Y. T. Yu, and T. H. Tse, "5W+1H pattern: A perspective of systematic mapping studies and a case study on cloud software testing," *J. Syst. Softw.*, vol. 116, pp. 206–219, 2016.

[3] IBM Cloud, "Defining IaaS, PaaS and SaaS." [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/iaas-paas-saas. [Accessed: 05-Feb-2019].

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] H. H. Liu, *Software performance and scalability: a quantitative approach*. Hoboken, N.J: John Wiley & Sons, 2011.

[6] T. Atmaca, T. Begin, A. Brandwajn, and H. Castel-Taleb, "Performance Evaluation of Cloud Computing Centers with General Arrivals and Service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2341–2348, 2016.

[7] K. Blokland, J. Mengerink, and M. Pol, *Testing Cloud Services: How to Test SaaS, PaaS & IaaS*. Rocky Nook, 2013.

[8] H. Aljahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu,

"Multi-tenancy in cloud computing," in *Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*, 2014, pp. 344–351.

[9] B. Jennings and R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *J. Netw. Syst. Manag.*, vol. 23, no. 3, pp. 567–619, Jul. 2015.

[10] J. Gao, X. Bai, W. T. Tsai, and T. Uehara, "SaaS testing on clouds - Issues, challenges, and needs," in *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, 2013, pp. 409–415.

[11] M. Becker, S. Lehrig, and S. Becker, "Systematically Deriving Quality Metrics for Cloud Computing Systems," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15*, 2015, pp. 169–174.

[12] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is , and What It Is Not," in *Presented as part of the 10th International Conference on Autonomic Computing*, 2013, pp. 23–27.

[13] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of*

Software Architectures, 2015, pp. 83–92.

[14] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud : Utility-Oriented Federation of Cloud Computing Environments for Scaling of," in *Algorithms and Architectures for Parallel Processing (10th International Conference, ICA3PP 20)*, 2010, pp. 13–31.

[15] K. Hwang, Y. Shi, and X. Bai, "Scale-out vs. scale-up techniques for cloud performance and productivity," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2015, vol. 2015-Febru, no. February, pp. 763–768. [S62]

[16] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15*, 2015, pp. 83–92.

[17] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*, 2012, p. 85.

[18] K. Hwang, X. Bai, Y. Shi, M. Li, W. G. Chen, and Y. Wu, "Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 130–143, 2016.

[19] L. Guo, J. Liang, and D. Li, “Understanding Ineffectiveness of the Application-Level Fault Injection,” *Poster in ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016.

[20] A. A. Ahmad, P. Brereton and P. Andras, "A Systematic Mapping Study of Empirical Studies on Software Cloud Testing Methods," *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, 2017, pp. 555-562. doi: 10.1109/QRS-C.2017.94

[21] A. Al-Said Ahmad and P. Andras, "Measuring the Scalability of Cloud-Based Software Services," *2018 IEEE World Congress on Services (SERVICES)*, San Francisco, CA, 2018, pp. 5-6. doi: 10.1109/SERVICES.2018.00016

[22] A. A. Ahmad and P. Andras, "Measuring and Testing the Scalability of Cloud-based Software Services," *2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)*, Amman, 2018, pp. 1-8. doi: 10.1109/ISIICT.2018.8613297

[23] A. Al-Said Ahmad and P. Andras, “Cloud-based software services delivery from the perspective of scalability,” *Int. J. Parallel, Emergent Distrib. Syst.*, pp. 1–16, 2019. doi: 10.1080/17445760.2019.1617864

[24] A. Al-Said Ahmad and P. Andras, “Scalability Analysis Comparisons of Cloud-based Software Services,” *In revision cycle, Journal of Cloud Computing,*

2019.

[25] B. Kitchenham, *Evidence-Based Software Engineering and Systematic Literature Reviews*, vol. 4. CRC Press, 2006.

[26] B. Kitchenham, "Procedures for Performing Systematic Reviews," 2004.

[27] F. Q. B. Da Silva, A. L. M. Santos, S. Soares, A. C. C. Frana, C. V. F. Monteiro, and F. F. MacIel, "Six years of systematic literature reviews in software engineering: An updated tertiary study," *Inf. Softw. Technol.*, vol. 53, no. 9, pp. 899–913, 2011.

[28] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.

[29] RTI, "The economic impacts of inadequate infrastructure for software testing," *Natl. Inst. Stand. Technol. RTI Proj.*, p. 309, 2002.

[30] T. Parveen and S. Tilley, "When to migrate software testing to the cloud?," in *ICSTW 2010 - 3rd International Conference on Software Testing, Verification, and Validation Workshops*, 2010, no. Vm, pp. 424–427.

[31] T. Hanawa, T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, and M. Sato, "Large-scale software testing environment using cloud computing

technology for dependable parallel and distributed systems," in *ICSTW 2010 - 3rd International Conference on Software Testing, Verification, and Validation Workshops*, 2010, pp. 428–433. [S36]

[32] H. Liu and D. Orban, "Remote network labs," in *ACM SIGCOMM Computer Communication Review*, 2010, vol. 40, no. 1, p. 83.

[33] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia, "VATS: Virtualized-aware Automated Test Service," in *Proceedings - 5th International Conference on the Quantitative Evaluation of Systems, QEST 2008*, 2008, pp. 93–102.

[34] C. Jia and Y. T. Yu, "Using the 5W+1H model in reporting systematic literature review: A case study on software testing for cloud computing BT - 13th International Conference on Quality Software, QSIC 2013, July 29, 2013 - July 30, 2013," in *Quality Software (QSIC), 2013 13th International Conference on*, 2013, pp. 222–229.

[35] K. Inçki, I. Ari, and H. Sözer, "A survey of software testing in the cloud," in *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability Companion, SERE-C 2012*, 2012, vol. 46, no. 6, pp. 18–23.

[36] S. Nachiyappan and S. Justus, "Cloud testing tools and its challenges: A comparative study," *Procedia Comput. Sci.*, vol. 50, pp. 482–489, 2015.

[37] X. Bai, M. Li, B. Chen, W. T. Tsai, and J. Gao, "Cloud testing tools," in

Proceedings - 6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011, 2011, no. Sose, pp. 1–12.

[38] J. Gao, X. Bai, W. T. Tsai, and T. Uehara, "Testing as a service (TaaS) on clouds," in *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, 2013, pp. 212–223.

[39] L. Riungu-Kalliosaari, O. Taipale, and K. Smolander, "Software Testing as a Service," in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, 2013, pp. 196–215.

[40] K. Sunitha, "A Survey on Securing the Virtual Machines in Cloud Computing," *IJISET-International J. Innov. Sci. Eng. Technol.*, vol. 1, pp. 1–9, 2014.

[41] J. Mukherjee, M. Wang, and D. Krishnamurthy, "Performance testing web applications on the cloud," in *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014*, 2014, pp. 363–369. [S18]

[42] M. Vasar, S. N. Srirama, and M. Dumas, "Framework for monitoring and testing web application scalability on the cloud," in *Proceedings of the WICSA/ECSA 2012 Companion Volume on - WICSA/ECSA '12*, 2012, p. 53. [S31]

[43] Y. H. Tung, C. C. Lin, and H. L. Shan, "Test as a service: A framework for web security TaaS service in cloud environment," *Proc. - IEEE 8th Int. Symp. Serv. Oriented Syst. Eng. SOSE 2014*, pp. 212–217, 2014. [S21]

[44] J. Wu, C. Wang, Y. Liu, and L. Zhang, "AGARIC - A hybrid cloud based testing platform," in *Proceedings - 2011 International Conference on Cloud and Service Computing, CSC 2011*, 2011, pp. 87–94. [S63]

[45] N. Snellman, A. Ashraf, and I. Porres, "Towards automatic performance and scalability testing of rich internet applications in the cloud," in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, 2011, pp. 161–169. [S65]

[46] A. Ali and N. Badr, "Performance testing as a service for web applications," in *2015 IEEE 7th International Conference on Intelligent Computing and Information Systems, ICICIS 2015*, 2016, pp. 356–361. [S59]

[47] M. B. Cooray, J. H. Hamlyn-Haris, and R. G. Merkel, "Test reconfiguration for service oriented applications," in *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*, 2011, pp. 300–305. [S58]

[48] H. Sun, X. Wang, M. Yan, Y. Tang, and X. Liu, "Towards a scalable paas for service oriented software," *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, pp. 522–527, 2013. [S66]

[49] M. Yan, H. Sun, X. Wang, and X. Liu, “WS-TaaS: A testing as a service platform for Web Service load testing,” *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, pp. 456–463, 2012. [S67]

[50] L. Zhang, Y. Chen, F. Tang, and X. Ao, “Design and implementation of cloud-based performance testing system for web services,” *Proc. 2011 6th Int. ICST Conf. Commun. Netw. China, CHINACOM 2011*, pp. 875–880, 2011. [S43]

[51] O. Starov and S. Vilkomir, “Integrated TaaS platform for mobile development: Architecture solutions,” *2013 8th Int. Work. Autom. Softw. Test, AST 2013 - Proc.*, pp. 1–7, 2013. [S24]

[52] J. Gao, W.-T. Tsai, R. Paul, X. Bai, and T. Uehara, “Mobile Testing-as-a-Service (MTaaS)--Infrastructures, Issues, Solutions and Needs,” in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE)*, 2014, pp. 158–167. [S48]

[53] S. Zhang and B. Pi, “Mobile functional test on TaaS environment,” in *Proceedings - 9th IEEE International Symposium on Service-Oriented System Engineering, IEEE SOSE 2015*, 2015, vol. 30, pp. 315–320. [S44]

[54] I. K. Villanes, E. A. B. Costa, and A. C. Dias-Neto, “Automated Mobile Testing as a Service (AM-TaaS),” *Proc. - IEEE World Congr. Serv. Serv. 2015*, pp. 79–86, 2015. [S68]

[55] C. Tao and J. Gao, "Cloud-Based Mobile Testing as a Service," in *International Journal of Software Engineering and Knowledge Engineering*, 2016, vol. 26, no. 01, pp. 147–152. [S61]

[56] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012, pp. 22–28. [S23]

[57] H. Turner *et al.*, "Building a Cloud-Based Mobile Application Testbed," in *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, IGI Global, 2013, pp. 382–403. [S53]

[58] O. Rebollo, D. Mellado, E. Fernández-medina, and H. Mouratidis, "Empirical Evaluation of a Cloud Computing Information Security Governance Framework," *Inf. Softw. Technol.*, vol. 58, pp. 44–57, 2015. [S7]

[59] R. Li, D. Abendroth, X. Lin, Y. Guo, H.-W. Baek, E. Eide, R. Ricci, and J. Van der Merwe, "Potassium: penetration testing as a service," in *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*, 2015, pp. 30–42. [S16]

[60] P. Kamongi, M. Gomathisankaran, and K. Kavi, "Nemesis: automated architecture for threat modeling and risk assessment for cloud computing," *Proc. 6th ASE*, pp. 1–10, 2014. [S57]

[61] P. Kamongi, S. Kotikela, K. Kavi, M. Gomathisankaran, and A. Singhal, "VULCAN: Vulnerability assessment framework for cloud computing," *Proc. - 7th Int. Conf. Softw. Secur. Reliab. SERE 2013*, pp. 218–226, 2013. [S33]

[62] S. Ristov, M. Gusev, and A. Donevski, "OpenStack Cloud Security Vulnerabilities from Inside and Outside," *CLOUD Comput. 2013 Fourth Int. Conf. Cloud Comput. GRIDs, Virtualization OpenStack*, no. c, pp. 101–107, 2013. [S49]

[63] S. Kotikela, K. Kavi, and M. Gomathisankaran, "Vulnerability Assessment In Cloud Computing," *2012 Int. Conf. Secur. Manag. (SAM 2012)*, pp. 67–73, 2012. [S33]

[64] L. Compagna, P. Guilleminot, and A. D. Brucker, "Business process compliance via security validation as a service," in *2013 IEEE sixth international conference on software testing, Verification and validation*, 2013, pp. 455–462. [S26]

[65] A. Donevski, S. Ristov, and M. Gusev, "Security assessment of virtual machines in open source clouds," *Inf. Commun. Technol. Electron. Microelectron. (MIPRO), 2013 36th Int. Conv.*, pp. 1094–1099, 2013. [S47]

[66] P. Zech, M. Felderer, and R. Breu, "Towards a model based security testing approach of cloud computing environments," in *Software Security and Reliability Companion, 2012 IEEE Sixth International Conference on*, 2012, pp.

47–56. [S25]

[67] R. Schwarzkopf, M. Schmidt, C. Strack, S. Martin, and B. Freisleben, "Increasing virtual machine security in cloud environments," *J. Cloud Comput.*, vol. 1, no. 1, pp. 1–12, 2012. [S12]

[68] H. C. Li, P. H. Liang, J. M. Yang, and S. J. Chen, "Analysis on cloud-based security vulnerability assessment," in *Proceedings - IEEE International Conference on E-Business Engineering, ICEBE 2010*, 2010, pp. 490–494. [S46]

[69] M. Menzel, R. Warschofsky, I. Thomas, C. Willems, and C. Meinel, "The service Security Lab: A model-driven platform to compose and explore service security in the cloud," in *Proceedings - 2010 6th World Congress on Services, Services-1 2010*, 2010, pp. 115–122. [S41]

[70] A. Tchana, N. De Palma, B. Dillenseger, and X. Etchevers, "A self-scalable load injection service," *Softw. - Pract. Exp.*, vol. 45, no. 5, pp. 613–632, 2015. [S6]

[71] D. Jayasinghe, S. Malkowski, J. Li, Q. Wang, Z. Wang, and C. Pu, "Variations in performance and scalability: An experimental study in IaaS clouds using multi-tier workloads," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 293–306, 2014. [S9]

[72] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, and C. Pu, "Variations

in performance and scalability when migrating n-tier applications to different clouds," in *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, 2011, pp. 73–80. [S9]

[73] Y. Sun, J. White, S. Eade, and D. C. Schmidt, "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization," *J. Syst. Softw.*, vol. 116, pp. 146–161, 2016. [S10]

[74] A. Turner, A. Fox, J. Payne, and H. S. Kim, "C-MART: Benchmarking the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1256–1266, 2013. [S13]

[75] Q. Gao, W. Wang, G. Wu, X. Li, J. Wei, and H. Zhong, "Migrating load testing to the cloud: A case study," in *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, 2013, pp. 429–434. [S20]

[76] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa, "DS-Bench Toolset: Tools for dependability benchmarking with simulation and assurance," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2012, pp. 1–8. [S36]

[77] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 143. [S37]

[78] M. A. El-Refaey and M. A. Rizkaa, "CloudGauge: A dynamic cloud and virtualization benchmarking suite," in *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2010, pp. 66–75. [S40]

[79] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Adaptable, model-driven security engineering for SaaS cloud-based applications," *Autom. Softw. Eng.*, vol. 21, no. 2, pp. 187–224, 2014. [S5]

[80] F. Chauvel, H. Song, N. Ferry, and F. Fleurey, "Evaluating robustness of cloud-based systems," *J. Cloud Comput.*, vol. 4, no. 1, 2015. [S11]

[81] A. O. Portillo-Dominguez, M. Wang, J. Murphy, and D. Magoni, "Automated WAIT for cloud-based application testing," in *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014*, 2014, pp. 370–375. [S19]

[82] J. Gao, K. Manjula, P. Roopa, E. Sumalatha, X. Bai, W. T. Tsai, and T. Uehara, "A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation," in *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 2012, pp. 464–471. [S28]

[83] H. Srikanth and M. B. Cohen, "Regression Testing in Software as a Service," in *Conference On Software Maintenance*, 2011, pp. 372–381. [S30]

[84] S. Scherzinger, E. C. De Almeida, F. Ickert, and M. D. Del Fabro, "On the necessity of model checking NoSQL database schemas when building SaaS applications," in *Proceedings of the 2013 International Workshop on Testing the Cloud - TTC 2013*, 2013, pp. 1–6. [S38]

[85] D. Jayasinghe *et al.*, "Expertus: A generator approach to automate performance testing in IaaS clouds," in *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 2012, pp. 115–122. [S39]

[86] W.-T. Tsai, Q. Shao, Y. Huang, and X. Bai, "Towards a scalable and robust multi-tenancy SaaS," in *Proceedings of the Second Asia-Pacific Symposium on Internetware - Internetware '10*, 2010, pp. 1–15. [S42]

[87] J. Gao, P. Pattabhiraman, X. Bai, and W. T. Tsai, "SaaS performance and scalability evaluation in clouds," in *Proceedings - 6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011*, 2011, pp. 61–71. [S35]

[88] T. M. King, A. S. Ganti, and D. Froslie, "Enabling automated integration testing of cloud application services in virtualized environments," *Proc. 2011 Conf. Cent. Adv. Stud. Collab. Res.*, pp. 120–132, 2011. [S64]

[89] D. Preuveneers, T. Heyman, Y. Berbers, and W. Joosen, "Systematic scalability assessment for feature oriented multi-tenant services," *J. Syst.*

Softw., vol. 116, pp. 162–176, 2016. [S3]

[90] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar, "Testing of data-centric and event-based dynamic service compositions," *Softw. Test. Verif. Reliab.*, vol. 23, no. 6, pp. 465–497, 2013. [S14]

[91] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar, "Test coverage of data-centric dynamic compositions in service-based systems," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, 2011, pp. 40–49. [S14]

[92] J. Zhou, B. Zhou, and S. Li, "Automated model-based performance testing for PaaS cloud services," in *Proceedings - IEEE 38th Annual International Computers, Software and Applications Conference Workshops, COMPSACW 2014*, 2014, pp. 644–649. [S29]

[93] K. Ravindran and A. Adiththan, "Verification of non-functional properties of cloud-based distributed system services," *Proc. 9th Int. Work. Autom. Softw. Test - AST 2014*, pp. 43–49, 2014. [S15]

[94] L. Qu, Y. Wang, M. A. Orgun, L. Liu, H. Liu, and A. Bouguettaya, "CCCloud: Context-aware and credible cloud service selection based on subjective assessment and objective assessment," *IEEE Trans. Serv. Comput.*, vol. 8, no. 3, pp. 369–383, 2015. [S8]

[95] W. Jenkins, S. Vilkomir, P. Sharma, and G. Pirocanac, "Framework for testing cloud platforms and infrastructures," in *Proceedings - 2011 International Conference on Cloud and Service Computing, CSC 2011*, 2011, pp. 134–140. [S32]

[96] L. Yu, W. T. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a service over cloud," in *Proceedings - 5th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2010*, 2010, pp. 181–188. [S22]

[97] C. S. Wu and Y. T. Lee, "Automatic SaaS test cases generation based on SOA in the cloud service," in *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 2012, pp. 349–354. [S27]

[98] W. T. Lo, X. L. Liu, R. K. Sheu, S. M. Yuan, and C. Y. Chang, "An architecture for cloud service testing and real time management," in *Proceedings - International Computer Software and Applications Conference*, 2015, vol. 3, pp. 598–603. [S60]

[99] G. Sunyé, E. C. De Almeida, Y. Le Traon, B. Baudry, and J. M. Jézéquel, "Model-based testing of global properties on large-scale distributed systems," *Inf. Softw. Technol.*, vol. 56, no. 7, pp. 749–762, 2014. [S1]

[100] Y. H. Tung and S. S. Tseng, "A novel approach to collaborative testing in a crowdsourcing environment," *J. Syst. Softw.*, vol. 86, no. 8, pp. 2143–2153,

2013. [S2]

[101] J. A. Meira, E. C. De Almeida, Y. Le Traon, and G. Sunye, "Peer-to-peer load testing," in *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, 2012, pp. 642–647. [S17]

[102] A. Pakhira and P. Andras, "Leveraging the Cloud for Large-Scale Software Testing–A Case Study: Google Chrome on Amazon," in *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, IGI Global, 2013, pp. 252–279. [S52]

[103] J. H. Hill, "Using Test Clouds to Enable Continuous Integration Testing of Distributed Real-time and Embedded System Applications," in *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, IGI Global, 2013, pp. 174–195. [S51]

[104] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-Cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," *CCGrid 2010 - 10th IEEE/ACM Int. Conf. Clust. Cloud, Grid Comput.*, pp. 631–636, 2010. [S36]

[105] T. Hanawa and M. Sato, "D-cloud: Software testing environment for dependable distributed systems using cloud computing technology," in *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, IGI Global, 2012, pp. 340–355. [S36]

[106] T. Hanawa *et al.*, "Customizing virtual machine with fault injector by integrating with specc device model for a software testing environment D-cloud," in *Proceedings - 16th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2010*, 2010, pp. 47–54. [S54]

[107] Y. Yamato, "Automatic system test technology of virtual machine software patch on IaaS cloud," *IEEJ Trans. Electr. Electron. Eng.*, vol. 10 (1), pp. S165–S167, 2015. [S4]

[108] W. T. Tsai, P. Zhong, J. Balasooriya, Y. Chen, X. Bai, and J. Elston, "An approach for service composition and testing for cloud computing," in *Proceedings - 2011 10th International Symposium on Autonomous Decentralized Systems, ISADS 2011*, 2011, pp. 631–636. [S34]

[109] E. Bower, T. Parveen, and S. Tilley, "Performance Analysis of a Distributed Execution Environment for JUnit Test Cases on a Small Cluster," in *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, IGI Global, 2013, pp. 96–112. [S50]

[110] M. Staats and C. Păsăreanu, "Parallel symbolic execution for structural test generation," in *Proceedings of the 19th international symposium on Software testing and analysis*, 2010, pp. 183–194. [S55]

[111] S. Bucur, V. Ureche, C. Zamfir, and G. Candea, "Parallel symbolic execution for automated real-world software testing," in *Proceedings of the sixth*

conference on Computer systems - EuroSys '11, 2011, p. 183. [S65]

[112] G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC*, 2010, p. 155. [S65]

[113] P. Moura and F. Kon, "Automated scalability testing of software as a service," *2013 8th Int. Work. Autom. Softw. Test, AST 2013 - Proc.*, pp. 8–14, 2013. [S69]

[114] N. R. Herbst, S. Kounev, A. Weber, and H. Groenda, "BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments," in *Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, 2015, pp. 46–56.

[115] A. Bauer, N. Herbst, and S. Kounev, "Design and Evaluation of a Proactive, Application-Aware Auto-Scaler," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17*, 2017, pp. 425–428.

[116] M. Beltran, "Defining an Elasticity Metric for Cloud Computing Environments," in *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2016, pp. 172–179.

[117] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and

elasticity of distributed database systems," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1219–1230, Aug. 2014.

[118] A. Ilyushkin *et al.*, "An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17*, 2017, pp. 75–86.

[119] M. Hasan Jamal, A. Qadeer, W. Mahmood, A. Waheed, and J. Jason Ding, "Virtual machine scalability on multi-core processors based servers for cloud computing workloads," in *Proceedings - 2009 IEEE International Conference on Networking, Architecture, and Storage, NAS 2009*, 2009, pp. 90–97.

[120] S. Lehrig, R. Sanders, G. Brataas, M. Cecowski, S. Ivanšek, and J. Polutnik, "CloudStore — towards scalability, elasticity, and efficiency benchmarking and analysis in Cloud computing," *Futur. Gener. Comput. Syst.*, vol. 78, pp. 115–126, 2018.

[121] G. Brataas, N. Herbst, S. Ivansek, and J. Polutnik, "Scalability Analysis of Cloud Software Services," in *Proceedings - 2017 IEEE International Conference on Autonomic Computing, ICAC 2017*, 2017, pp. 285–292.

[122] M. Woodside, "Scalability Metrics and Analysis of Mobile Agent Systems," in *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, 2001, pp. 234–245.

[123] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *J. King Saud Univ. - Comput. Inf. Sci.*, 2018. doi: 10.1016/j.jksuci.2018.09.021

[124] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing Dependability with Software Fault Injection: A Survey," *ACM Comput. Surv.*, vol. 48, no. 3, pp. 44:1--44:55, Feb. 2016.

[125] L. Feinbube, L. Pirl, P. Tröger, and A. Polze, "Software Fault Injection Campaign Generation for Cloud Infrastructures," in *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017*, 2017, pp. 622–623.

[126] C. Sheridan, D. Whigham, and M. Artač, "DICE Fault Injection Tool," in *Proceedings of the 2Nd International Workshop on Quality-Aware DevOps*, 2016, pp. 36–37.

[127] Y. Xiaoyong, L. Ying, W. Zhonghai, and L. Tiancheng, "Dependability analysis on open stack IaaS cloud: bug anaysis and fault injection," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014, pp. 18–25.

[128] Y. Deng, R. Mahindru, A. Sailer, S. Sarkar, and L. Wang, "Providing fault injection to cloud-provisioned machines." Google Patents, 05-Sep-2017.

[129] L. Herscheid, D. Richter, and A. Polze, "Experimental Assessment of Cloud

Software Dependability Using Fault Injection," in *Doctoral Conference on Computing, Electrical and Industrial Systems*, 2015, pp. 121–128.

[130] K. Ye, Y. Liu, G. Xu, and C.-Z. Xu, "Fault Injection and Detection for Artificial Intelligence Applications in Container-Based Clouds," in *Cloud Computing – CLOUD 2018*, 2018, pp. 112–127.

[131] T. Hacaloglu, P. E. Eren, D. Mishra, and A. Mishra, "A software development process model for cloud by combining traditional approaches," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9416, pp. 421–430.

[132] M. Babar, A. ur Rahman, and F. Arif, "Cloud Computing Development Life Cycle Model (CCDLC)," in *International Conference on Future Intelligent Vehicular Technologies*, 2016, pp. 189–195.

[133] N. S. Chauhan and A. Saxena, "A green software development life cycle for cloud computing," *IT Prof.*, vol. 15, no. 1, pp. 28–34, 2013.

[134] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 45–52, 2011.

[135] "IEEE Standard for Software and System Test Documentation," *IEEE Std*

829-2008. pp. 1–150, 2008.

[136] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, 2014.

[137] JMeter, "JMeter HTTP Request." [Online]. Available: https://jmeter.apache.org/usermanual/component_reference.html#HTTP_Request. [Accessed: 01-Apr-2019].

[138] "Amazon EC2." [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 23-Jan-2019].

[139] Microsoft, "What is Azure." [Online]. Available: https://azure.microsoft.com/en-gb/overview/what-is-azure/. [Accessed: 05-Feb-2019].

[140] Z. Xiao, Q. Chen, and H. Luo, "Automatic scaling of internet applications for cloud computing services," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1111–1123, 2014.

[141] "What Is Amazon EC2 Auto Scaling?" [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html. [Accessed: 23-Jan-2019].

[142] S. R. Seelam, P. Dettori, P. Westerink, and B. B. Yang, "Polyglot application

auto scaling service for platform as a service cloud," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, 2015, pp. 84–91.

[143] "Elastic Load Balancing." [Online]. Available: https://aws.amazon.com/elasticloadbalancing/. [Accessed: 23-Jan-2019].

[144] OrangeHRM Inc., "OrangeHRM." .

[145] "OrangeHRM Github." [Online]. Available: https://github.com/orangehrm/orangehrm. [Accessed: 14-Feb-2019].

[146] OrangeHRM, "OrangeHRM REST APIS." [Online]. Available: https://api.orangehrm.com/?url=/apidoc/index.html. [Accessed: 14-Feb-2019].

[147] mediawiki, "MediaWiki." [Online]. Available: https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F. [Accessed: 14-Feb-2019].

[148] MediaWiki, "RESTBase for MediaWiki." [Online]. Available: https://www.mediawiki.org/wiki/RESTBase. [Accessed: 14-Feb-2019].

[149] "Applications Taxonomy and Methodology," *Apps Run The World*, 2018. [Online]. Available: https://www.appsruntheworld.com/taxonomy/. [Accessed: 25-May-2019].

[150] A. JMeter™, "Apache JMeter." .

[151] M. Autili *et al.*, "CHOReOS Dynamic Development Model Definition (D2. 1)," 2011.

[152] I. Saleh and K. Nagi, "HadoopMutator: A Cloud-Based Mutation Testing Framework," in *14th International Conference on Software Reuse, ICSR 2015*, 2014, pp. 172–187.

[153] H. Jayathilaka, C. Krintz, and R. Wolski, "Performance Monitoring and Root Cause Analysis for Cloud-hosted Web Applications," in *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, 2017, pp. 469–478.

[154] Q. Duan, "Cloud service performance evaluation: status, challenges, and opportunities – a survey from the system modeling perspective," *Digit. Commun. Networks*, vol. 3, no. 2, pp. 101–111, 2017.

[155] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, M. Kawaba, and C. Pu, "Response time reliability in cloud environments: An empirical study of n-tier applications at high resource utilization," in *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 378–383.

[156] B. Butler, "Who's got the best cloud latency?," 2016. [Online]. Available: https://www.networkworld.com/article/3095022/cloud-computing/who-s-got-the-best-cloud-latency.html,. [Accessed: 19-Mar-2018].

[157] A. Avizienis, J.-. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[158] R. Piscitelli, S. Bhasin, and F. Regazzoni, “Fault attacks, injection techniques and tools for simulation,” in *Hardware Security and Trust*, Springer, 2017, pp. 27–47.

[159] N. Huber, F. Brosig, N. Dingle, K. Joshi, and S. Kounev, “Providing Dependability and Performance in the Cloud: Case Studies,” in *Resilience Assessment and Evaluation of Computing Systems,* W. Katinka, A. Alberto, M. Vieira, and M. Aad van, Eds. 2012, pp. 391–412.

[160] “Chaos Monkey.” [Online]. Available: https://github.com/netflix/chaosmonkey. [Accessed: 16-Mar-2019].

[161] “Principles of chaos engineering,” pp. 1–2, 2018. [Online]. Available: http://principlesofchaos.org/. [Accessed: 17-Mar-2019].

[162] A. Avizienis, J.-C. Laprie, and B. Randell, *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

Appendix A: Mapping Study Protocol Details

Change Record version 2.3

Document status	Version	Date	Changes from previous version
Draft	V0.1	18/1/16	- None.
Revision	V1.0	25/1/16	- Correction of typos. - Updated the quality assessment checklist, inclusion criteria, and data extraction strategy.
Major Revision	V1.1	11/2/16	- Correction of typos. - Updated RQs, inclusion criteria, and data extraction strategy.
Revision	V1.2	25/2/16	- Update data extraction strategy, selection criteria, inclusion criteria, and search strategy.
Revision	V1.3	9/3/16	- Update Search Strategy.
Major Revision	V2.0	21/4/16	- Grammar corrections. - Updated the RQs, data extraction strategy, and search strategy.
Revision	V2.1	8/5/16	- Updated the RQs, data extraction strategy, and background.
Revision	V2.2	24/5/16	- Update data extraction strategy, and search strategy.
Revision	V 2.3	10/6/16	- Update data extraction strategy

Reviewing Team:

Name	Description	Role
Amro Al-Said Ahmad	Lead Researcher	Designing the protocol, extracted and reviewing all required data.
Prof. Pearl Brereton	Reviewer	Reviewing the protocol and performing data extraction for assign random sample of studies.
Prof. Peter Andras	Reviewer	Reviewing the protocol and performing data extraction for assign random sample of studies.

Appendix B: IEEE 829 Test Plan Template

- Test Plan Identifier
- Introduction
- Test Items
- Features to Be Tested
- Features Not to Be Tested
- Approach
- Item Pass/Fail Criteria
- Suspension Criteria and Resumption Requirements
- Test Deliverables
- Testing Tasks
- Environmental Needs
- Responsibilities
- Staffing and Training Needs
- Schedule
- Risks and Contingencies
- Approvals