UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

Master of Science in Computer Science

Research Project Report

Cloud Computing: Assessing the Impact of Application Architecture on the Performance of Cloud-Based Applications

Ciirah, Stephen Mburu

P58/9219/2006

Supervisors: Dr. Elisha Abade & Dr. Andrew Kahonge

Submitted in partial fulfilment of the requirements for the award of the Degree of Master of Science in Computer Science of the University of Nairobi.

March 2018

DECLARATION OF AUTHENTICITY

I, Stephen Mburu Ciirah, hereby declare that this project report entitled "**Cloud Computing: Assessing The Impact of Application Architecture on the Performance of Cloud-Based Applications**," submitted to the University of Nairobi, School of Computing and Informatics, is my original work and has not been presented for the award of a degree in any other university.

…… ……

Signature Date

Stephen Mburu Ciirah
P58/9219/2006

This research project has been submitted for examination with our approval as the University Supervisors.

…… ……

Signature Date

Dr. Elisha Abade
University of Nairobi

…… ……

Signature Date

Dr. Andrew Kahonge
University of Nairobi

DEDICATION

Dedicated to my dear wife Elizabeth and our lovely children Joshua, Lily and Wema.

ACKNOWLEDGEMENTS

It is now 10 years since I registered to undertake this degree course at the School of Computing and Informatics, University of Nairobi. To quote Nelson Mandela, it always seems impossible until it is over.

The successful completion of this research and the master's degree program could not have been possible without the profound support and contributions by many people over the years.

First and foremost, I would like to thank God for His grace that has been multiplied upon me to persistently pursue the completion of this course in spite of the many challenges encountered over the years. By Him, I have overcome, to the glory, praise and honor of God the most High.

My most sincere gratitude to my supervisors Dr. Elisha Abade and Dr. Andrew Kahonge, who stepped in to rescue me from despair through timely and candid feedback on my work and for challenging me to be systematic in my approach and insisting on good quality work. Without this intervention, this work would not have been completed and I would have been struck off from the university register as a post graduate student. Thank you very much sirs.

I also wish to thank other members of the faculty who have played a significant supporting role over the years. These include the late Prof. Okello Odongo, Prof. Timothy Waema, Prof. Peter Wagacha, Prof. Muthoni Masinde, Dr. Dan Orwa, Dr. Wanjiku Ng'ang'a and Mr. Christopher Moturi.

Finally, I thank the Director of the School of Computing and Informatics, Dr. Agnes Wausi, and all the support staff at the Director's office for facilitating the approval of extension of my registration with the Board of Post Graduate Studies to enable me complete this course.

ABSTRACT

Cloud Computing, which involves over-the-Internet provision of dynamically scalable and often virtualized computing resources has very quickly become one of the hottest topics for practicing engineers and academics in domains related to engineering, science and art for building large-scale networks and Internet applications.

The goal of this research was to investigate how application architecture impacts the performance of applications in a cloud computing environment. The specific objectives of the study were one: to identify factors driving the adoption of cloud computing for delivery computing services; two: to discover architectures used for the development of cloud based applications; three: to determine the correlation between throughput and scalability of applications and finally, to determine the moderating effect of architecture to the relationship between load and the performance of applications in a cloud computing environment.

Through a detailed literature search and review, both historical and current perspective of cloud computing were examined. A conceptual framework for the research was development based on the Gartner Conceptual Framework for Application Performance Management. The experimental research methodology was adopted for the study. Microsoft Azure cloud platform and Microsoft Visual Studio Team Services was used to conduct graduated load performance tests for a convenience sample of web based applications. Data analysis was conducted by using the Pearson product-moment correlation coefficient and moderation multiple regression analysis.

The literature search and review findings concurred with the observations made by that there has been limited academic research in this area of study. The findings of the study showed that there was a strong positive correlation between throughput and scalability of applications which was statistically significant, therefore the alternative hypothesis was accepted. On the other hand, the results showed that while there was a positive moderating effect of architecture on the relationship between load and performance, the moderating effect was not statistically significant, hence the null hypothesis was accepted.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
3G	Third Generation of wireless mobile telecommunications
ABAP	Advanced Business Application Programming
API	Application Programming Interface
APM	Application Performance Management
Avg.	Average
CIFS	Common Internet File System
CMS	Content Management System
DDD	Domain Driven Design
DSL	Digital Subscriber Line
E1	2.048 Mbps line
EA	Enterprise Architecture
EC2	Elastic Compute Cloud
e-Commerce	Electronic Commerce
EECS	Electrical Engineering and Computer Science
EUE	End User Experience
HQ	Headquarters
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
I/O	Input / Output
IaaS	Infrastructure as a Service
IBM	International Business Machines
ICT	Information and Communication Technology

IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
ISDN	Integrated Services Digital Network
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
LAN	Local Area Network
Mbps	Megabits per second
NFS	Network File Storage
NIST	National Institute of Standards and Technology
No.	Number
OS	Operating System
PaaS	Platform as a Service
REST	Representative State Transfer
SaaS	Software as a Service
SAP	Session Announcement Protocol
SLA	Service Level Agreement
SMB	Server Message Block
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
T1	1,544 Mbps line
TCO	Total Cost of Ownership
TCP/IP	Transmission Control Protocol/Internet Protocol
UI	User Interface

UK	United Kingdom
URL	Universal Resource Locator
VM	Virtual Machine
VMM	Virtual Machine Monitoring
VPN	Virtual Private Network
VSTS	Visual Studio Team Services
WAN	Wide Area Network
WIFI	Wireless Fidelity
WWW	World Wide Web
XML	Extensible Markup Language

1 INTRODUCTION

1.1 Background

Cloud computing is a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g. computer networks, servers, storage, applications and services) which can be rapidly provisioned and released with minimal management effort (Yu and Chen, 2011).

Cloud computing has very quickly become one of the hottest topics – if not the hottest one – for practising engineers and academics in domains related to engineering, science, and art for building large-scale networks and Internet applications (Pallis, 2010).

The increased availability and reliability of the Internet, even in the developing economies like Africa, is driving businesses around the world to consider the cloud computing as the preferred model of delivery of Information Technology (IT) services and solutions. The emergence of Cloud computing in Africa is a natural extension of the deployment of advanced IT technologies by high-end users in both the consumer and enterprise services markets (Research ICT Africa, 2013).

It is predictable that in the immediate and near future, more enterprises will be migrating their existing applications to the cloud and implementing new cloud-based applications. According to Forrester Research predictions for technology trends in 2017, Cloud computing has been the most exciting and disruptive force in the tech market in the last decade, and it will continue to disrupt traditional computing models at least through 2020. From the beginning of 2017, large commercial institutions will move to cloud in a big way, and that will supercharge the market. It is predicted that the influx of industry dollars will push the global public cloud market to $236 billion in 2020, up from $146 billion in 2017 (Dai, 2017).

On the other hand, although the industry has made significant progress in the development of Cloud Computing technologies, products and services, there is very limited academic research on the subject of Cloud Computing.

While Cloud computing is gaining growing popularity in the IT industry; academia appeared to be lagging behind the rapid developments in this field. While the industry has been

pushing the Cloud research agenda at a high pace, academia has only recently joined, as can be seen through the sharp rise in workshops and conferences focussing on Cloud Computing (Sriram and Khajeh-Hosseini, 2010).

1.2 Problem statement

Based on the defined benefits and the risks of cloud computing (Linthcum, 2016), enterprises are considering migrating their existing applications to the cloud or implementing new cloud-based applications.

The cloud computing model is attractive to smaller organisations that are looking to remain flexible in a challenging economic climate and contain costs. Price alone is only one component of the total cost of ownership (TCO). Larger organisations are looking at factors such as adoption costs, training, downtime, regulatory implications, data security risks and how a change might jeopardise trade secrets. As a result, many larger organisations are more reluctant to move to the cloud (Turner, 2012).

Migration to cloud computing is a strategic organisational decision that can affect performance, productivity, growth, as well as increase competitiveness. The decision to migrate is usually complicated and dynamic due to the immaturity and the still evolving nature of the cloud computing environment (Alkhalil, Sahandi and John, 2016).

The problem is that, to the knowledge of the researcher, there is currently no readily available information, methodologies and tools for evaluating the performance of cloud based applications from their architecture point of view. By understanding how architecture relates to the performance of applications, organisations can make better informed decisions on the adoption of cloud computing. It would be possible to identify application architecture patterns that satisfy the performance expectations when enterprises are considering migrating existing applications to the cloud or developing new cloud based applications.

1.3 Research objectives

The overall objective of this project is to assess how the architecture of applications impacts the performance of applications in a cloud computing environment.

Specific objectives were:

i. Identify factors driving the adoption of cloud computing as the new way of delivering computing services
ii. Discover the main application architectures used in the development of cloud based applications
iii. Conduct an experiment to measure and compare the performance of applications when subjected to different levels of load
iv. Analyse the data to determine the correlation between throughput and scalability
v. Analyse the data to determine the moderating effect of architecture on the relationship between load and performance of the cloud-based applications.

1.4 Research questions

i. What factors are driving the adoption of cloud computing as the new way of delivering computing services?
ii. What are the main application architectures used for the development of cloud-based applications?
iii. What is the correlation between the throughput and scalability of application?
iv. What is the moderating effect of architecture on the relationship between load and performance of the cloud-based applications?

1.5 Research outcomes

First, the research findings show how the architecture of applications impacts the performance of cloud based applications by determining the correlation between architecture and the performance of applications in a cloud computing environment.

Then, the results provide a basis for recommending application architecture considerations for migrating existing applications to the cloud and for developing new cloud based applications.

The results will also form a good foundation for further research into this relatively new area of academic study in cloud computing.

1.6 Limitations of the study

Due to the current state of research in this field of cloud computing, the following and highlighted as limitations for this research work:

- Limited academic research in the area of Cloud Computing
- Internal and extenal validity of the experiment design due to the limited control of the cloud computing environment used in the study

Further study and research on this subject should be carried out beyond the partial fulfilment of the requirements for the Master of Science degree program.

1.7 Glossary of terms

i. **Application architecture** - Application architecture is the discipline that guides application design. Application architecture paradigms, such as service-oriented architecture (SOA), provide principles that influence design decisions and patterns that provide proven design solutions.

ii. **Application performance** - Performance refers to the capability of a system to provide a certain response time, serve a defined number of users or process a certain amount of data. So performance is a software quality metric. Unlike to what many people think it is not vague, but can be defined in number (Reitbauer, 2008).

iii. **Application throughput** – measures the volume of requests/responses or volume of transactions in relation to time, for example, average requests per second or number of transactions per second (Haines, 2006).

iv. **Application scalability** – measures the ability of an application to maintain its performance under increasing load (Haines, 2006).

v. **Application Performance Management** - In the fields of information technology and systems management, application performance management (APM) is the monitoring and management of performance and availability of software applications.

vi. **Cloud computing** - Cloud computing is a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal

management effort. Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in either privately owned or third-party data centres that may be located far from the user, ranging in distance from across a city to across the world. Cloud computing relies on sharing of resources to achieve coherence and economy of scale, similar to a utility (like the electricity grid) over an electricity network.

vii. **Internet** - The Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to link devices worldwide. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony, and peer-to-peer networks for file sharing.

viii. **Microservices architecture** - Microservices architecture is an approach to application development in which a large application is built as a suite of modular services. Each module supports a specific business goal and uses a simple, well-defined interface to communicate with other sets of services.

ix. **Monolithic architecture** - A software system is called "monolithic" if it has a monolithic architecture, in which functionally distinguishable aspects (for example data input and output, data processing, error handling, and the user interface) are all interwoven, rather than containing architecturally separate components.

x. **Service Oriented Architecture (SOA)** - A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies

xi. **Correlation** – Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases.

xii. **Inductive reasoning** - Inductive reasoning is a logical process in which multiple premises, all believed true or found true most of the time, are combined to obtain a

specific conclusion. Inductive reasoning is often used in applications that involve prediction, forecasting, or behaviour.

xiii. **Academic research** - the careful study of a given subject, field, or problem, undertaken to discover facts or principles.

xiv. **Best practices** - industry or professional procedures that are accepted or prescribed as being correct or most effective.

xv. **Wikipedia** - is a free online encyclopaedia, created and edited by volunteers around the world and hosted by the Wikimedia Foundation.

xvi. **Grid computing** – can be defined as the use of computer resources from multiple administrative domains to reach a common goal. It can be considered as a distributed system with non-interactive workloads involving a large number of files, yet more loosely coupled, heterogeneous, and geographically dispersed as compared to cluster computing. In its simplest form, grid computing may be represented as a "super virtual computer" composed of many networked loosely coupled computers acting together to perform humongous tasks (Biswas, 2011).

xvii. **Utility computing** - involves the renting of computing resources such as hardware, software and network bandwidth on an as-required, on-demand basis. In other words, what were earlier considered products, are treated as services in utility computing (Biswas, 2011).

xviii. **Cloud infrastructure** - A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer (Mell and Grance, 2011).

xix. **Virtual Machine (VM)** - is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine.

xx. **Application Program Interface (API)** - is code that allows two software programs to communicate with each other. The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required

syntax is described in the documentation of the application being called (Rouse, Nolle and Li, 2017).

xxi. **Experimental research method** - is a systematic and scientific approach to research in which the researcher manipulates one or more variables, and controls and measures any change in other variables (Explorable, 2009).

xxii. **Non-probability sampling** - is a sampling technique where the samples are gathered in a process that does not give all the individuals in the population equal chances of being selected (Explorable, 2009).

2 LITERATURE REVIEW

2.1 Chapter overview

This chapter explores the literature on the three key concepts that underpin this research work, which include cloud computing, application architecture and application performance management. The chapter begins by examining the meaning of cloud computing and defining the key terminologies used in the subject and key developments in the area.

In discussing application architecture and performance, the chapter highlights two architectures, looking at their advantages and disadvantages with respect to performance in a cloud computing environment.

2.2 Cloud computing

2.2.1 What is cloud computing?

The definition of Cloud Computing has been evolving with time and has been a subject of rigorous consultation between government, industry and academia over many years. This section explores the various definitions that have emerged and are now widely used in the contemporary literature on Cloud Computing.

In an attempt to understand the definition of Cloud Computing, it is argued that "Cloud computing" is increasingly becoming one of those buzz words of the moment. As tends to happen with buzz words (or phrases, in this case), it can be confusing to understand exactly what everything is and how the various technologies differ from one another. In an interview with a utility computing industry expert, an old definition emerged which stated that Cloud computing enables users and developers to utilise services without knowledge of, expertise with, nor control over the technology infrastructure that supports them (Danielson, 2017).

This definition appeared in early versions of Wikipedia which defined Cloud computing as location-independent computing, whereby shared servers provide resources, software, and data to computers and other devices on demand, as with the electricity grid. Or more simply, remote computing. Cloud computing is a natural evolution of the widespread adoption of virtualization, service-oriented architecture and utility computing. Details are abstracted from consumers, who no longer have need for expertise in, or control over, the technology infrastructure "in the cloud" that supports them (En.wikipedia.org, 2011).

In yet another early definition of Cloud computing, it was described as a new supplement, consumption and delivery model for IT services based on the Internet, and it typically involves the over-the-Internet provision of dynamically scalable and often virtualized resources (Gartner., 2008) and (Knorr, 2008).

It is a byproduct and consequence of the ease-of-access to remote computing sites provided by the Internet. This frequently takes the form of web-based tools or applications that users can access and use through a web browser as if it were a program installed locally on their computer (The Economist, 2009).

In another effort from the academia, Cloud Computing was described as the long-held dream of computing as a utility, which has the potential to transform a large part of the Information Technology (IT) industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. Developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expertise to operate it. They need not be concerned about over-provisioning for a service whose popularity does not meet their predictions, thus wasting costly resources, or under-provisioning for one that becomes wildly popular, thus missing potential customers and revenue. Moreover, companies with large batch-oriented tasks can get results as quickly as their programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours. (EECS Department, University of California, Berkeley, 2009).

After years in the works and 15 drafts, the American National Institute of Standards and Technology's (NIST) working definition of cloud computing, the 16th and final definition was published as The NIST Definition of Cloud Computing, NIST Special Publication 800-145 (Brown, 2011).

According to NIST, Cloud Computing is defined a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance, 2011).

This definition by NIST has been adopted and expanded in Wikipedia, which defines Cloud computing as a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal management effort. Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in either privately owned or third-party data centres located far from the user–ranging in distance from across a city to across the world. Cloud computing relies on sharing of resources to achieve coherence and economy of scale, similar to a utility (like the electricity grid) over an electricity network (Wikipedia.com, 2017).

In cloud computing, the word cloud (also phrased as "the cloud") is used as a metaphor for "the Internet," so the phrase cloud computing means "a type of Internet-based computing," where different services — such as servers, storage and applications — are delivered to an organization's computers and devices through the Internet (Beal, 2017).

The metaphor of the cloud may be loosely based on the cloud shaped diagram used in marketing and architectural diagrams to denote the Internet. In this sense the metaphor of a cloud is something that is ubiquitous yet obscure; everywhere, yet abstracting its inner technical workings from the less sophisticated, less interested, or less privileged user (Cuttitta, 2013).

In more simplified terms, the Cloud computing metaphor suggests that for a user, the network elements representing the provider rendered services are invisible, as if obscured by a cloud (En.wikipedia.org, 2017).

The figure below illustrates the concept of cloud computing.

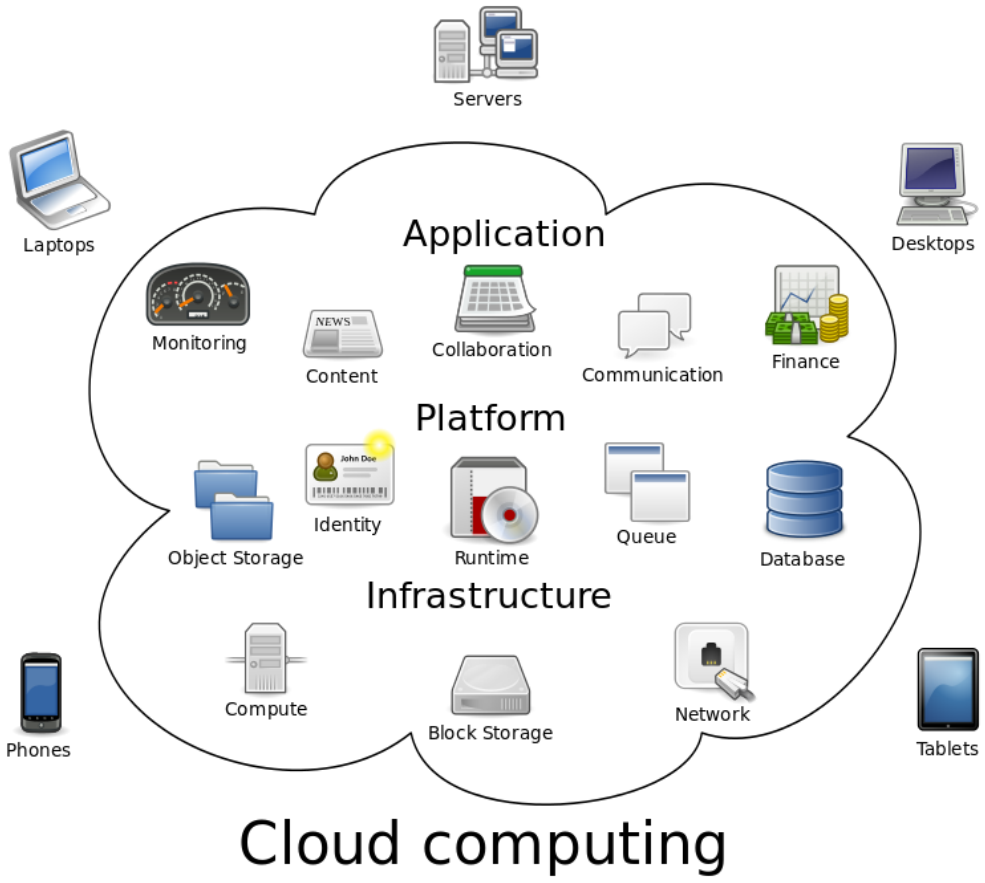

Figure 2.1: Cloud computing conceptual diagram (Johnston, 2017)

While discussing the subject of Cloud Computing, one must distinguish Cloud Computing from other closely related terminology which may easily be confused to be synonymous with cloud computing. These words include grid computing and utility computing, which are described in the glossary of terms.

According to NIST, the Cloud computing model is composed of five essential characteristics, three service models, and four deployment models (Mell and Grance, 2011).

Each of these components of the cloud model is explained in the sections that follow.

2.2.2 Essential characteristics of cloud computing

There are the five key characteristics that are enshrined in the cloud computing model (Mell and Grance, 2011).

These characteristics are illustrated in the figure below:

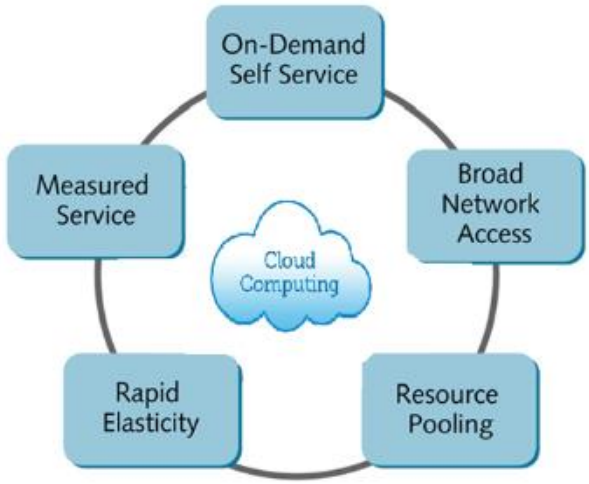

Figure 2.2: Essential characteristics of cloud computing (Somepalle, 2015)

i. **On-demand self-service.** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

ii. **Broad network access.** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

iii. **Resource pooling.** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location-independence in that the customer has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

iv. **Rapid elasticity.** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

v. **Measured service.** Cloud systems automatically control and optimise resource use by leveraging a metering capability at some level of abstraction for the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilised service. Typically, the metering capability is based on a pay-per-use or charge-per-use basis.

2.2.3 Cloud computing service models

The early description of the components of Cloud Computing argued that the Cloud model has layers, with each providing a distinct level of functionality. This stratification of the Cloud's components has provided a means for the layers of Cloud computing to become a commodity just like electricity, telephone service, or natural gas. The commodity that cloud computing sells is computing power at a lower cost and expense to the user (Walker, 2012).

However, in later years, there appears to be a general acceptance to describe the layers of the Cloud model as the Cloud Service Models as outlined below:

i. **Software as a Service (SaaS).** The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings (Mell and Grance, 2011).

   The SaaS model is also described as the top layer of the Cloud service model, and it is the layer that is most visualised as the Cloud. Applications run here and are provided on demand to users. Software as a Service (SaaS) has providers such as Google Pack which includes Internet accessible applications, tools such as Calendar, Gmail, Google Talk, Docs, and much more (Walker, 2012).

ii. **Platform as a Service (PaaS)**. The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using

programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment (Mell and Grance, 2011).

The PaaS model is also described as a middle layer of the Cloud, service model. It provides the application infrastructure. Platform as a Service (PaaS) provides access to operating systems and associated services. It provides a way to deploy applications to the cloud using programming languages and tools supported by the provider. You do not have to manage or control the underlying infrastructure, but you do have control over the deployed applications and, to some degree over application hosting environment configurations.

PaaS has providers such as Amazon's Elastic Compute Cloud (EC2). The small entrepreneur software house is an ideal enterprise for PaaS. With the elaborated platform, world-class products can be created without the overhead of in-house production (Walker, 2012).

iii. **Infrastructure as a Service (IaaS).** The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer can deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls), (Mell and Grance, 2011).

IaaS is also described as the bottom layer and foundation of the Cloud, service model. It consists of the physical assets — servers, network devices, storage disks, etc. Infrastructure as a Service (IaaS) has providers such as the IBM® Cloud. Using IaaS one has no control of the underlying infrastructure, but has complete management of the operating systems, storage, deployment applications, and, to a limited degree, control over select networking components (Walker, 2012).

The figure below depicts the cloud computing layers embedded in the “as Service” components:

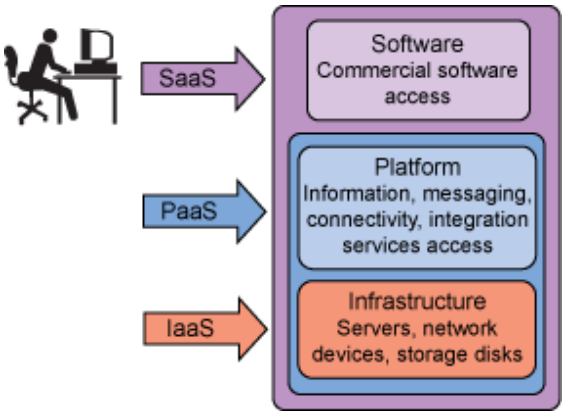

Figure 2.3: Cloud computing layers "as a Service" components (Walker, 2012).

The figure below depicts further details of the Cloud Computing service models:

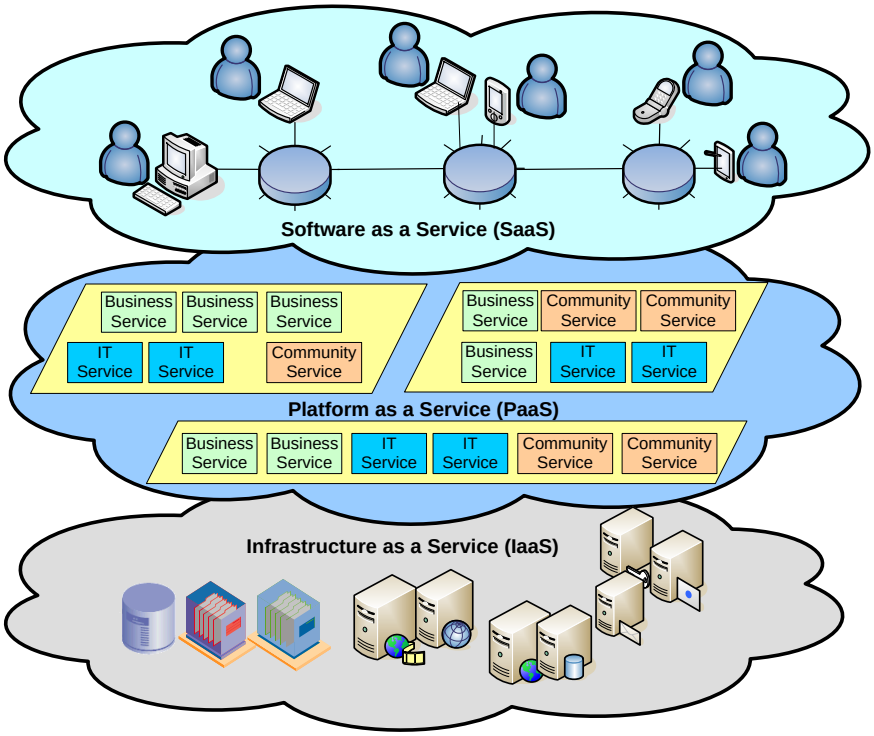

Figure 2.4: Cloud computing service models

2.2.4 Cloud computing deployment models

Whereas most literature discusses three models for deploying Cloud computing, there are four deployment models as described below:

i. **Private cloud.** The cloud infrastructure provisioned for exclusive use by a single organisation comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organisation, a third party, or some combination of them, and it may exist on or off premises.

ii. **Community cloud.** The cloud infrastructure provisioned for exclusive use by a specific community of consumers from organisations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It is owned, managed, and operated by one or more of the organisations in the community, a third party, or some combination of them, and it may exist on or off premises.

iii. **Public cloud.** The cloud infrastructure provisioned for open use by the general public. It is owned, managed, and operated by a business, academic, or government organisation, or some combination of them. It exists on the premises of the cloud provider.

iv. **Hybrid cloud.** The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardised or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

2.2.5 Components of Cloud Computing Infrastructure

This section takes a deeper examination into the Infrastructure as a Service (IaaS) service model and describes the major components of this layer which include: computing, storage and network.

For each of these components, regardless of whether a cloud provider sells services at a low level of abstraction like EC2 or a higher level like AppEngine, computing, storage, and networking must all focus on horizontal scalability of virtualized resources rather than on single node performance (Armbrust et al., 2010).

The sections below discuss the characteristics of these components.

i. Computing (processing)

Cloud infrastructure services, also known as "Infrastructure as a Service (IaaS)", delivers computer infrastructure - typically a platform virtualization environment - as a service.

Rather than purchasing servers, software, data-center space or network equipment, clients instead buy those resources as a fully outsourced service. Cloud infrastructure often takes the form of a Tier-3 data centre with many Tier-4 attributes, assembled from hundreds of virtual machines (Arias, 2011).

A data centre is a facility used to house computer systems and associated components, such as telecommunications and storage systems. It includes redundant or backup power supplies, redundant data communications connections, environmental controls (e.g., air conditioning, fire suppression) and security devices. Large data centres are industrial scale operations using as much electricity as a small town (En.wikipedia.org, 2015).

The figure below shows multiple racks of servers and how a data centre commonly looks.

Figure 2.5: Rows of servers inside an Amazon data centre (Amazon Web Services, 2015)

According to the Uptime Institute, the most stringent level is a Tier 4 data centre, which is designed to host mission critical computer systems, with fully redundant subsystems and compartmentalised security zones controlled by biometric access control methods. Another consideration is the placement of the data centre in a subterranean context, of data security as well as environmental factors such as cooling requirements (Turner, P., Seader, J. and Renaud, V., 2010).

As already pointed out, the cloud infrastructure derives computing power from hundreds of virtual machines in a data centre. A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine.

Virtual machines are classified into two major categories, based on their use and degree of correspondence to any real machine. A system virtual machine provides a complete system

platform which supports the execution of a complete operating system (OS). In contrast, a process virtual machine is designed to run a single program, which means that it supports a single process.

System virtual machines (sometimes called hardware virtual machines) allow the sharing of the underlying physical machine resources between different virtual machines, each running its independent operating system. The software layer providing the virtualization is called a virtual machine monitor or hypervisor. A hypervisor can run on bare hardware (Type 1 or native VM) or top of an operating system (Type 2 or hosted VM).

Below is an outline of the operation of the hypervisor in a cloud computing environment.

Virtual Machine Monitor (VMM)

The virtual machine monitor (VMM) or the hypervisor provides the means for simultaneous use of cloud facilities as illustrated in the figure below.

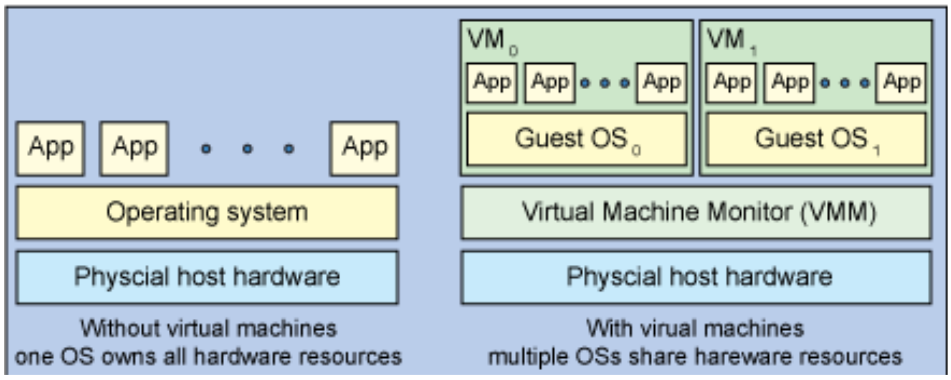

Figure 2.6: How the Virtual Machine Monitor (VMM) works

VMM is a program on a host system that lets one computer support multiple, identical execution environments. From the user's point of view, the system is a self-contained computer which is isolated from other users.

In reality, every user is being served by the same machine. A virtual machine is one operating system (OS) that is managed by an underlying control program allowing it to appear to be multiple operating systems. In cloud computing, VMM enables users to monitor and thus manage aspects of the process such as data access, data storage, encryption, topology, addressing and workload movement.

ii. Storage

Storage is known as the warehouse of cloud computing.

Cloud storage is a model of networked online storage where data is stored on multiple virtual servers, generally hosted by third parties, rather than being hosted on dedicated servers. Hosting companies operate large data centres; and people who require their data to be hosted buy or lease storage capacity from them and use it for their storage needs. The data centre operators, in the background, virtualize the resources according to the requirements of the customer and expose them as storage pools, which the customers can themselves use to store files or data objects. Physically, the resource may span across multiple servers.

Cloud storage services are accessed either through a web service Application Programming Interface (API) or a web-based user interface.

A cloud storage gateway can be optionally used at the customer premises, which expose cloud storage services as if they were local storage devices. Cloud storage gateways are network appliances or servers which translate standard cloud storage APIs such as Simple Object Access Protocol (SOAP) or Representative State Transfer (REST) to either block-based data storage protocols such as iSCSI or Fibre Channel, or file-based network storage protocols such as Network File System (NFS) or Common Internet File System (CIFS) also commonly known as Server Message Block (SMB).

The figure below illustrates the typical architecture of a cloud storage system which includes a master control server (storage gateway) and multiple storage servers (Strickland, J., 2008).

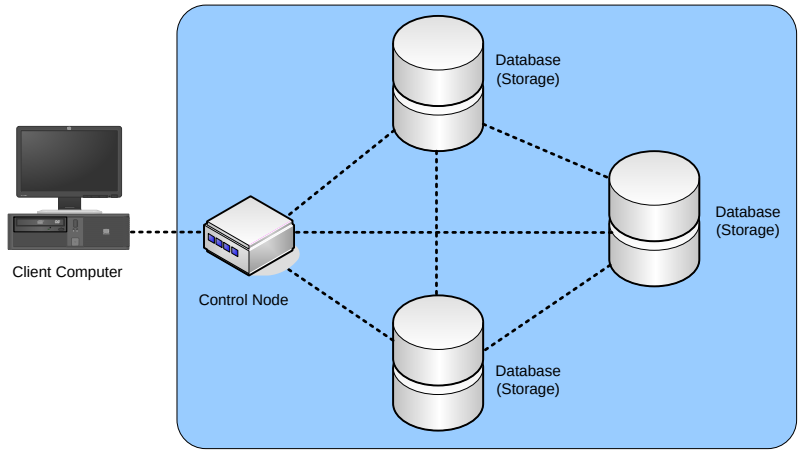

Figure 2.7: Cloud Storage System

iii. Network

Communications in data centres, which house the computing power of cloud computing, are based on networks running the IP protocol suite. Data centres contain a set of routers and switches that transport traffic between the servers and to the outside world. Redundancy of the Internet connection is provided by using two or more upstream service providers.

Network security elements are also usually deployed: firewalls, VPN gateways, intrusion detection systems, etc. Also common are monitoring systems for the network and some of the applications. Additional off-site monitoring systems are also typical, in the case of a failure of communications inside the data centre (Wikipedia, 2011).

Based on the various definitions of cloud computing, ultimately, the goal of cloud computing – regardless of model – is to create a fluid pool of resources across servers and data centres that enable users to access stored data and applications on an as-needed basis. Cloud computing networks, therefore, have two missions:

a) to support the movement of that pool resources as a single virtual resource, and
b) to connect users to these resources regardless of location

To make that happen, cloud computing networks -- whether they support public, private or hybrid clouds – must be able to:

a) Burst up and turn down bandwidth on demand
b) Provide extremely low latency throughput among storage networks, the data centre and the LAN
c) Allow for non-blocked connections between servers for automated movement of virtual machines (VMs).
d) Function within a management plane that stretches across enterprise and service provider networks.
e) Provide visibility despite this constantly changing environment.

Cloud computing networks can be seen as three interdependent structures: the front-end, which connects users to applications; a horizontal aspect, which interconnects physical servers and the movement of their VMs; and storage networks. The larger cloud can be built as either a Layer 2 or a Layer 3 network.

The figure below shows the different levels of cloud computing networks.

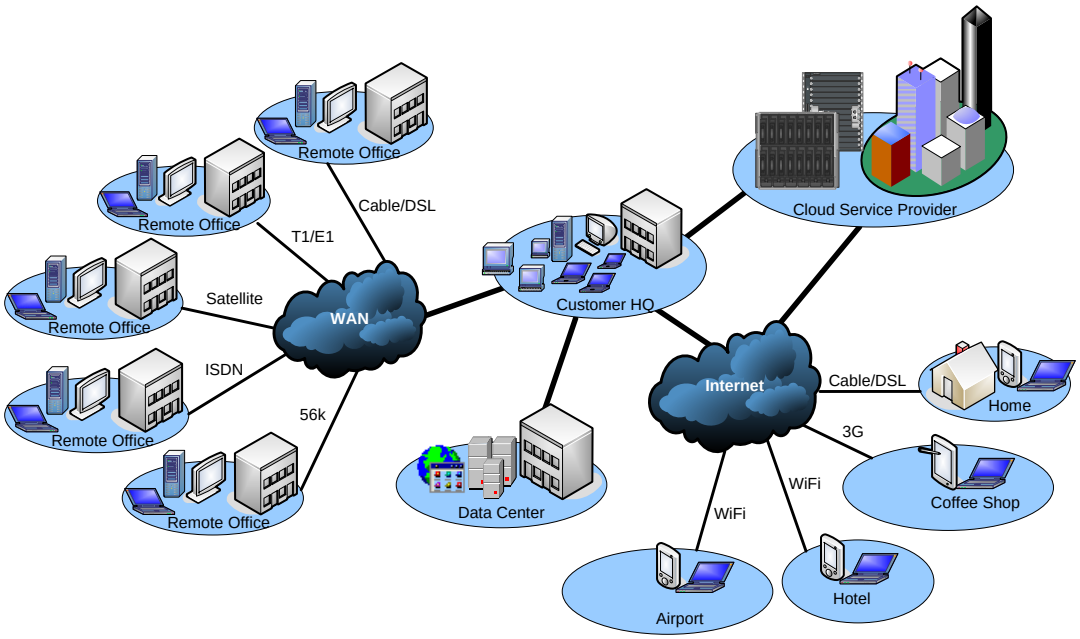

Figure 2.8: Cloud computing networks

2.2.6 Properties of Cloud Computing Infrastructure

Cloud computing is, at its core, about delivering applications or services in an on-demand environment. Cloud computing providers will need to support hundreds of thousands of users and applications/services and ensure that they are fast, secure, and available. To accomplish this goal, they'll need to build a dynamic, intelligent infrastructure with four core properties in mind: transparency, scalability, monitoring/management, and security (MacVittie, 2008).

i. Transparency

One of the premises of Cloud Computing is that services are delivered transparently regardless of the physical implementation of the "cloud". Transparency is one of the foundational concepts of cloud computing, in that the actual implementation of services in the "cloud" are obscured from the user. Transparency, therefore, describes another version of virtualization, where multiple resources appear to the user as a single resource.

It is unlikely that a single server or resource will always be enough to satisfy the demand for a given provisioned resource, which means transparent load-balancing and application delivery will be required to enable the transparent horizontal scaling of applications on-demand. The application delivery solution used to provide open load-balancing services will

need to be automated and integrated into the provisioning workflow process such that resources can be provisioned on-demand at any time.

For example, provisioning a service to a user or an organisation, it may need only a single server (real or virtual) to handle demand at the beginning, but as more users access the service, additional servers (real or virtual) are required. Transparency avails additional servers to the provisioned service without interrupting the service or requiring reconfiguration of the application delivery solution. With an integrated application delivery solution, for example, via a management API with the provisioning workflow system, then transparency is also achieved through the automated provisioning and de-provisioning of resources.

ii. Scalability

Obviously, cloud computing service providers are going to need to scale up and build out "mega data centres". Scalability is easy enough if you've deployed the proper application delivery solution, but what about scaling the application delivery solution? That is often not so easy, and it usually isn't a transparent process; there's configuration work and in many cases, re-architecting of the network. The potential to interrupt services is high and assuming that cloud computing service providers are supporting hundreds of thousands of customers, unacceptable.

The application delivery solution is going to need not only to provide the ability to scale the service infrastructure transparently but itself, as well. That is a tall order and something rarely seen in an application delivery solution.

Making things even more difficult will be the need to scale on-demand in real-time to optimise the use of application infrastructure resources. Analysts suggest that scalability will require a virtualized infrastructure such that resources are provisioned and de-provisioned quickly, easily and one hopes, automatically. The "control node" often depicted in high-level diagrams of the "Cloud Computing mega data centre" will need to provide on-demand dynamic application scalability. The control node will, therefore, require integration with the virtualization solution and the workflow or process responsible for provisioning.

iii. Intelligent Monitoring

To achieve the on-demand scalability and transparency required of a mega data centre in the cloud, the control node, i.e. application delivery solution, will need to have intelligent monitoring capabilities. It will need to understand when a particular server is overwhelmed and when network conditions are adversely affecting application performance. It needs to know the applications and services being served from the cloud and understand when behaviour is outside accepted norms. While this functionality can confidently be implemented externally in a massive management monitoring system, if the control node sees clients, the network, and the state of the applications it is in the best position to understand the real-time conditions and performance of all involved parties without requiring the heavy lifting of correlation that would be required by an external monitoring system.

But more than just knowing when an application or service is in trouble, the application delivery mechanism should be able to take action based on that information. If an application is responding slowly and is detected by the monitoring system, then the delivery solution should adjust application requests accordingly. If the number of concurrent users accessing a service is reaching capacity, then the application delivery solution should be able to not only detect that through intelligent monitoring but participate in the provisioning of another instance of the service to ensure service to all clients.

iv. Security

Security in Cloud Computing is critical, especially when you consider that if the cloud is compromised, potentially all services and associated data in the cloud are at risk. That means that the mega data centre must be architected with security in mind and it must be considered a priority for every application, service, and network infrastructure solution deployed.

The application delivery solution, as the "control node" in the mega data centre, is necessarily one of the first entry points into the cloud data centre and must itself be secure.

It should also provide full application security - from layer 2 to layer 7 - to thwart potential attacks at the edge. Network security, protocol security, transport layer security, and application security should be prime candidates for implementation at the perimeter of the cloud, in the control node. While there certainly will be and should be, additional security

measures deployed within the data centre, stopping as many potential threats as possible at the edge of the cloud will alleviate much of the risk to the internal service infrastructure.

2.3 Application architectures

2.3.1 Definitions

Application architecture is the organizational design of an entire software application, including all sub-components and external applications interchanges. There are several design patterns that are used to define this type of architecture, and these patterns help to communicate how an application will complete the necessary business processes as defined in the system requirements (Holmes, 2017).

The application architecture is used as a blueprint to ensure that the underlying modules of an application will support future growth. Growth can come in the areas of future interoperability, increased resource demand, or increased reliability requirements. With a completed architecture, stakeholders understand the complexities of the underlying components should changes be necessary in the future.

Microsoft on the other hand defines software application architecture as the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application (Microsoft Patterns and Practices Team, 2009).

Application architecture is the discipline that guides application design. Application architecture paradigms, such as service-oriented architecture (SOA), provide principles that influence design decisions and patterns that provide proven design solutions.

One of the realities of application development is that there are a lot of factors that go into its underlying architecture.

2.3.2 Application architecture styles

In their early work, David and Mary introduce a discusson on common architectural styles upon which many systems are currently based and show how different styles can be combined in a single design (Garlan and Shaw, 1994).

The table below shows a summary of architecture styles (Patterns & Practices Team, 2009).

Table 2.1: Summary of architecture styles

Architecture style	Description
Client/Server	Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
Component-Based Architecture	Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
Domain Driven Design	An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.
Layered Architecture	Partitions the concerns of the application into stacked groups (layers).
Message Bus	An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other.
N-Tier / 3-Tier	Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
Object-Oriented	A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.
Service-Oriented Architecture (SOA)	Refers to applications that expose and consume functionality as a service using contracts and messages

2.3.3 Monolithic architecture

In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform. A monolithic application is self-contained and independent from other computing applications.

A monolithic architecture is the traditional unified model for the design of a software program.

Monolithic, in this context, means composed all in one piece. Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled (Rouse and Wigmore, 2016).

In a different approach to the definition of the monilithic architecture, it is argued that a monolith is an architectural style or a software development pattern that fits into three viewtypes that include module, allocation and runtime monoliths (Annett, 2014).

These viewtypes are discussed below.

- Module - The code units and their relation to each other at compile time.
- Allocation - The mapping of the software onto its environment.
- Runtime - The static structure of the software elements and how they interact at runtime.

A monolith could refer to any of the basic viewtypes above.

Module monolith

In the case of a module monolith then all of the code for a system is in a single codebase that is compiled together and produces a single artifact. The code may still be well structured (classes and packages that are coherent and decoupled at a source level rather than a big-ball-of-mud) but it is not split into separate modules for compilation.

Conversely a non-monolithic module design may have code split into multiple modules or libraries that can be compiled separately, stored in repositories and referenced when required. There are advantages and disadvantages to both. However, the module monolith gives little insight into how the code is used as it is primarily done for development management.

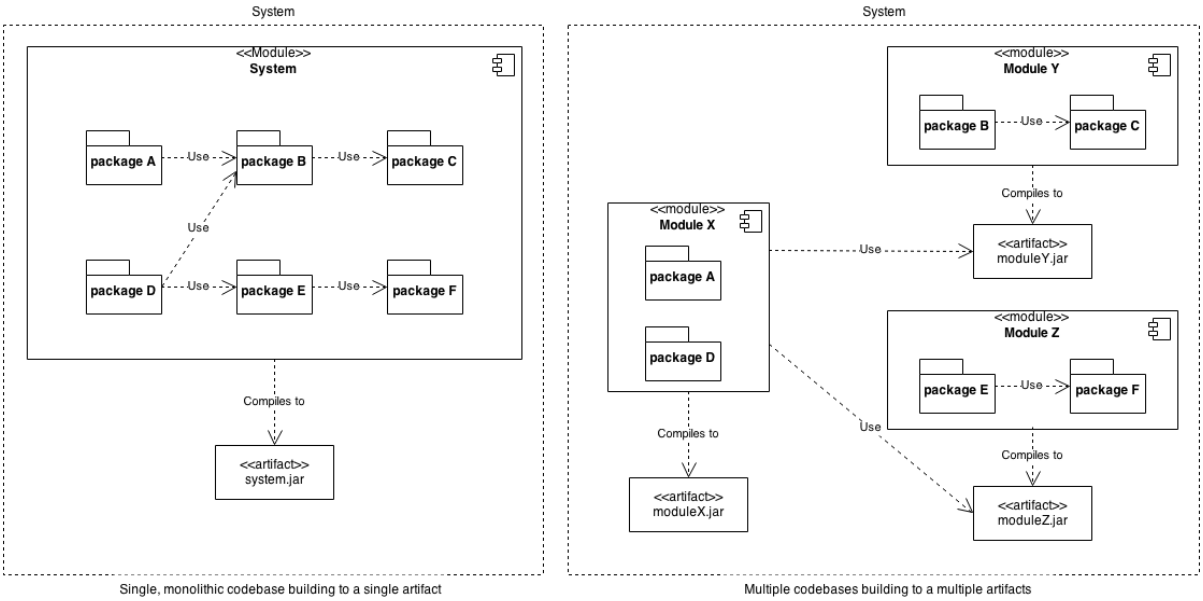

Figure 2.9: Module monolith (Annett, 2014)

Allocation Monolith

For an allocation monolith, all of the code is shipped/deployed at the same time. Once the compiled code is 'ready for release' then a single version is shipped to all nodes. All running components have the same version of the software running at any point in time. This is independent of whether the module structure is a monolith. The entire codebase can either be compiled at once for deployment or may be compiled as a set of deployment artifacts from multiple sources and versions. Either way this version for the system is deployed everywhere at once, often by stopping the entire system, rolling out the software and then restarting.

A non-monolithic allocation would involve deploying different versions to individual nodes at different times. This is again independent of the module structure as different versions of a module monolith could be deployed individually.

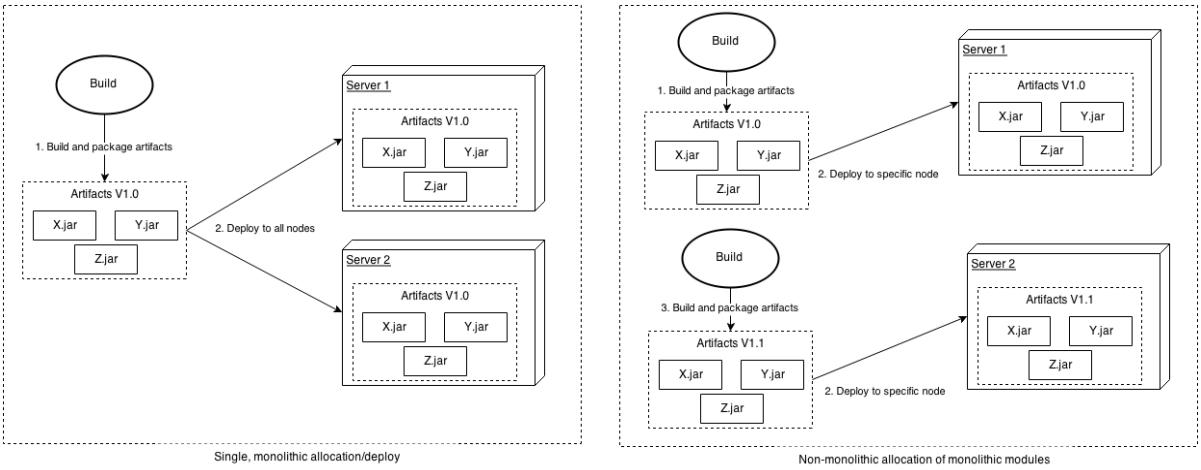

Figure 2.10: Allocation monolith (Annette, 2014)

Runtime Monolith

A runtime monolith will have a single application or process performing the work for the system, although the system may have multiple external dependencies. Many systems have traditionally been written as runtime monolithis, especially for line-of-business systems such as Payroll, Accounts Payable, Content Management Systems (CMS), etc.

Whether the runtime is a monolith is independent of whether the system code is a module monolith or not. A runtime monolith often implies an allocation monolith if there is only one main node/component to be deployed. However, this may not be the case if a new version of software is rolled out across regions, with separate users, over a period of time.

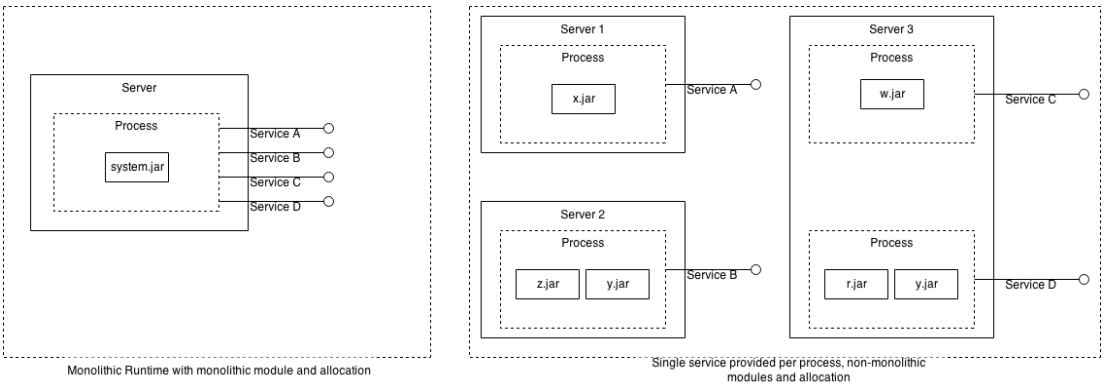

Figure 2.11: Runtime monolith (Annette, 2014)

Note that in these examples above are slightly forced for the viewtypes and it won't be as hard-and-fast in the real world.

Pros and cons of monilithic architecture

Here are the pros and cons of monolithic applications (Gooen, 2014):

Pros

- Faster initial development: With one application, it would be relatively easy to add additional features, especially when the application is relatively small. Several features were added to the 80,000 Hours codebase relatively easily (it would have taken more code to have separate applications for each one).
- Little user confusion: Users wouldn't have to learn about different applications, but would be focussed towards one application. For example, new applications on the same platform would be more easily found by users to the existing applications.
- Improved integration: Features could integrate with each other well and easily, as there is only one user table.
- User interface similarity: All of the pieces of the application would look very similar, so it would be obvious it's all part of one system.
- Power centralization (the good parts): If we have someone who is significantly better than average in charge, and it stays that way, then this would be a way to possibly improve development on average.

Cons

- Substantially less iteration: The larger a website is, the more difficult it is to change it. It would be incredibly tough, for example, to change the theme or UI or a monolith application. The means that we would have significantly less experimentation. This is one reason why the 80,000 Hours social network was put on hold after the redesign; it would have vastly increased the time to actually make that happen.
- Maintenance: The larger a website is, the more difficult it becomes to maintain the entire thing. Maintenance costs may go up exponentially with site size. This is one reason why many startups, with large amounts of funding (1-20 million dollars) have relatively simple websites. Large ones, especially large ones with large feature sets, typically don't have good reputations for stability.
- Power centralization: Obviously if there's one monolith application, it would ultimately be owned by one person or organization. If all application uses would be put into this

application, then this is a lot of power to trust in one organization. There's currently a decent level of distrust and conflicting goals between EA organizations. Thus, to have one EA organization in charge of all EA applications (one large one) would present a situation that could create controversy.

- Bureacracy: Even with power centralization being accepted, what would ownership of the site look like? If one wanted to take control of the 'volunteering' portion, would they have to ask the person in charge of the entire application?
- High set-up costs: In order to get each new volunteer up and running, the larger the application, the more difficult this would be. The volunteer would have to understand the infrastructure of whatever they do, so this could be a pretty big issue.
- Less chance of outside popularity: With a hive of web applications, if any single one does well (money tracker, less-wrong-blog, etc), it can spin out and become popular among more than EAs. With only one, we would have to hope that the entire thing would be popular, and also that this would not have happened anyone.
- Less ownership by volunteers: With an ecosystem of projects, we can encourage individuals to take ownership of their own projects. The projects would be like 'mini-startups' and individual motivation would be very much linked to project success. Motivation would be expected to go down substantially if it's a small part of someone else's project. Typically a good open source project has 1 main maintainer who does 80% of the work.

2.3.4 Microservices architecture

Microservices - also known as the microservices architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservices architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack (Richardson, 2017).

Essentially, microservices architecture is a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal (Huston, 2017).

Thanks to its scalability, this architectural method is considered particularly ideal when you have to enable support for a range of platforms and devices—spanning web, mobile, Internet of Things and wearables or simply when you're not sure what kind of devices you'll need to support in an increasingly cloudy future.

Out of their experience in working with microservices, James and Martin (Lewis and Fowler, 2014) argue that while there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

They therefore propose that the microservices architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

With yet another perspective of microservices architecture, a microservice is defined as a tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable application component. Based on a combination of Services Oriented Architecture (SOA) and domain-driven design (DDD), microservices architecture is a design paradigm that has three core objectives: development agility, deployment flexibility and precise scalability (Thomas and Gupta, 2017).

Pros and Cons

Whether or not microservices architecture is the right architecture depends on the requirements, because they all have their pros and cons. Below is an outline of some of the good and bad:

Pros

- Microservices architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team

- Code for different services can be written in different languages (though many practitioners discourage it)
- Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
- Easy to understand and modify for developers, thus can help a new team member become productive quickly
- The developers can make use of the latest technologies
- The code is organized around business capabilities
- Starts the web container more quickly, so the deployment is also faster
- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
- Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
- Easy to scale and integrate with third-party services
- No long-term commitment to technology stack

Cons

- Due to distributed deployment, testing can become complicated and tedious
- Increasing number of services can result in information barriers
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
- Being a distributed system, it can result in duplication of effort
- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
- Developers have to put additional effort into implementing the mechanism of communication between the services

- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
- The architecture usually results in increased memory consumption
- Partitioning the application into microservices is very much an art

Examples of microservices applications in industry

As Martin Fowler points out, Netflix, eBay, Amazon, the UK Government Digital Service, realestate.com.au, Forward, Twitter, PayPal, Gilt, Bluemix, Soundcloud, The Guardian, and many other large-scale websites and applications have all evolved from monolithic to microservices architecture (Lewis and Fowler, 2014).

Netflix has a widespread microservices architecture that has evolved from monolithic to what has been described by the architect of Netflix as fine grain SOA. It receives more than one billion calls every day, from more than 800 different types of devices, to its streaming-video API. Each API call then prompts around five additional calls to the backend service (Wetherill, 2014).

Amazon has also migrated to microservices. They get countless calls from a variety of applications—including applications that manage the web service API as well as the website itself—which would have been simply impossible for their old, two-tiered architecture to handle.

The auction site eBay is yet another example that has gone through the same transition. Their core application comprises several autonomous applications, with each one executing the business logic for different function areas.

2.4 Application architecture and performance

There are diverse factors that can impact the performance of applications implemented in a Cloud computing environment.

A research conducted by APMdigest (APMdigest - Application Performance Management, 2013), involving many of the Application Performance Management (APM) industry's experts — from analysts and consultants to users and the top vendors — reveals their

perspectives on the root causes of application performance problems. Based on the research, APMdigest compiled a list of 15 factors that impact application performance. In their conclusion, the list provides a broad picture of the many factors out there impacting application performance, which must be must be considered when managing application performance.

In an updated version of the APMdigest research report (APMdigest - Application Performance Management, 2016), 3 years later after the first research, due to the rapid changes in technology and the emergence of more experts in the APM field, the listed number of factors impacting application performance had doubled.

Broadly, these factors that impact application performance fall into one of two major groups: factors that are part of the environment, or factors within the application itself.

Out of the diverse factors, it has been pointed out that the top factor that impacts application performance is the architecture of the application itself. Often times you see this when an application is moved or migrated to another environment. For example, the impact of a "chatty" application can be hidden or mitigated on a high speed local LAN, but once moved to the cloud, the slower telecom speeds expose this design flaw in the form of high latency (APMDigest, 2016).

From yet another perspective, application design/architecture/complexity has been identified as the top factor that impacts application performance. It can be quite difficult to mitigate the effects of poor design, even with a great deal of additional work. Poorly designed applications may suffer from poor performance even with relatively low traffic (APMDigest, 2016).

A bad design/architecture decision will affect the performance of an application throughout its life time. Applications are complex, often comprised of shared services and deployed on shared infrastructure, like the Cloud computing environment. A good architecture requires understanding the relationships and interactions between the various components, and doing so without sacrificing user experience.

2.5 Summary of the Literatures

The literature research shows the complexity of the issues relating to cloud computing, application architectures and application performance. Most of the research work referenced in this study is from industry experts and is complemented by limited academic literature on the subjects.

It can however be inferred that the performance of cloud based applications will be affected by the design of the applications and how the design enables the application to take advantage of the capabilities of the cloud computing environment which provides dynamically scalable and virtualized computing resources.

The intent of this study is to assess the relationship between application architecture and the performance of applications in a cloud computing environment.

2.6 Conceptual Framework

2.6.1 Definition

This research seeks to discover the relationship between application architecture and the performance of applications in a cloud computing environment.

The conceptual framework represents the researcher's synthesis of literature on how to explain a phenomenon. It maps out the actions required in the course of the study to realize the research objectives (Regoniel, 2016). The conceptual framework therefore identifies the variables required in the research investigation and clarifies the relationships among the particular variables (McGaghie, Bordage and Shea, 2001).

In this section therefore, the conceptual framework for the research is presented. To begin with, a conceptual framework for Application Performance Management (APM) is presented. It is followed by a careful examination of various definitions of performance and scalability. The conceptual framework used for this study and the hypotheses tested are then discussed.

2.6.2 Gartner's Application Performance Management Conceptual Framework

In order to develop the conceptual framework for this research, a review of the Gartner APM conceptual framework was conducted.

In this conceptual framework, (Gartner, 2010) has defined five distinct dimensions of, or perspectives on, end-to-end application performance, which are essential to application performance management. Gartner points out that although each of these five dimensions are distinct, and often deployed by different stakeholders, there is a high-level, circular workflow that weaves the five dimensions together (Goldin, 2011).

The conceptual framework is illustrated in the figure below:

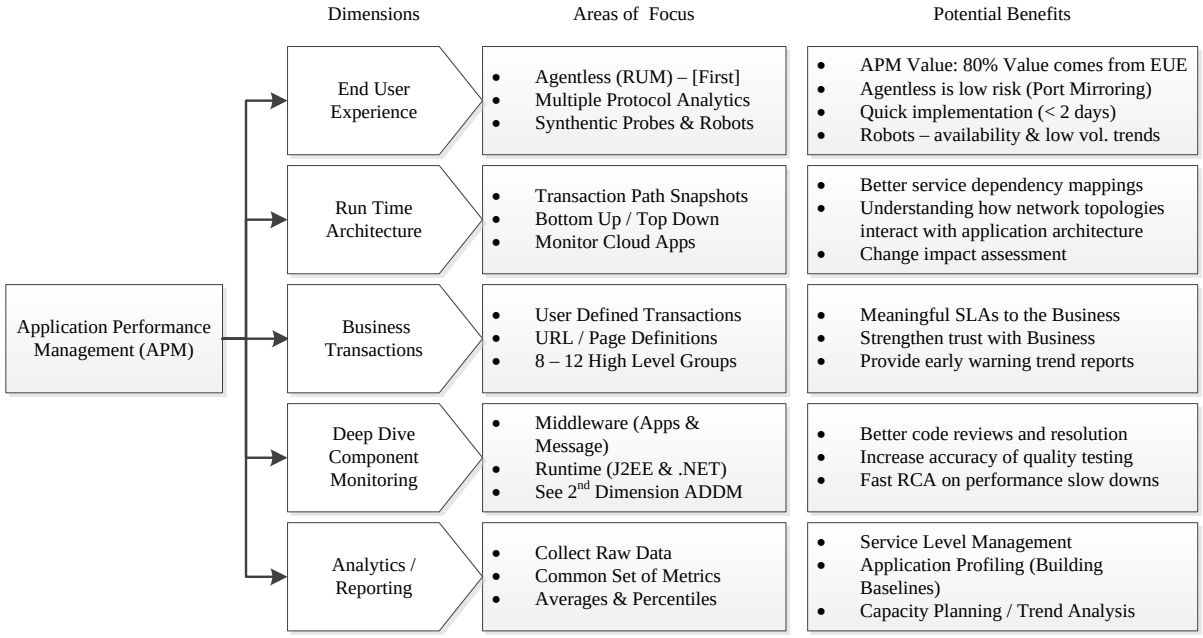

Figure 2.12: Gartner's APM conceptual model

The five dimensions of the framework are discussed below.

End User Experience (EUE)

End-user experience monitoring is the first step, which captures data on how end-to-end performance impacts the user and identifies the problem.

In his view of how to prioritize Gartner's model, (Dragich, 2012) asserts that Real-time Application Monitoring, also referred to as Top Down monitoring, is the cornerstone that gives the EUE its tangible value. Based on experience it is considered that Real User Monitoring (RUM) technology provides at least 80% of the APM value in terms of application visibility for the business and helps lay the foundation for performance trending and predictive analysis.

This approach of Top Down monitoring has two components, Passive and Active. Passive monitoring is usually an agentless appliance and low risk to implement using network port mirroring. In general, this can be up and running providing details of the application performance in less than two days. When considering an agentless solution, a key feature to look for is the ability to support multiple protocol analytics (e.g. XML, SAP ABAP, SQL) since most companies have more than just web-based applications to support.

Active monitoring, on the other hand, consists of synthetic probes and web robots which help report on system availability and predefined business transactions. This is a good complement when used with passive monitoring that together will help provide visibility on application health during off peak hours when transaction volume is low.

In their analysis of the model, (Xangati, 2016), present the passive and active methods of EUE monitoring as Agentless Monitoring and Synthetic Monitoring respectively. They also outline the advantages and disadvantages of each method.

i) Agentless Monitoring

Network traffic is tapped as it passes through switches and load balancers, and analyzed with data probes. The analysis reveals the performance of transactions across the entire IT infrastructure.

Advantages of this method:

- Data is presented quickly, since there are no complex scripts
- It is possible to get user-specific information such as the browser, geographical location, OS, and the like.

Disadvantages:

- Performance can only be tracked from the IT landscape to the end-user and not vice-versa
- Monitoring will not work without users or if there is an interruption just before the point whereby network traffic is being tapped

ii) Synthetic Monitoring

This involves running scripts using robots/probes to create a simulated end-user. One can therefore simulate several users at the same time using multiple monitoring robots, to create a constant flow of monitored traffic. One can also respond to possible faults quickly and in real-time without any real users.

Advantages of this method:

- There is always data available, with or without real users
- Results are never subjected to outside influences, since the pattern you run on the application is fixed
- Application errors can be detected before the application is opened
- Method can also be used to monitor the SLAs for applications

Runtime Application Architecture

This is the second dimension of the model, in which the software and hardware components involved in application execution and their communication paths are studied to establish the potential scope of the problem.

This is also referred to as Bottom Up monitoring. This will become a critical component to build on when working on event correlation to help implement an overall runtime architecture solution.

Providing the transaction path snapshots will also help bring together the Top Down and Bottom Up monitoring. This will give better service dependency mappings and an understanding on how the network topologies interact with the application architecture. Runtime views become an important area to focus on after one has built a solid application profile with the EUE, Business Transactions and Reporting/Analytics dimensions.

From their perspective, (Xangati, 2016), argue that this second dimension of the APM model helps to avoid errors while making changes, by performing a thorough Impact Analysis beforehand. With the ability to monitor the underlying IT infrastructure using a signal, then one can detect and clear faults much faster.

They contend, however, that a thorough understanding of IT infrastructure is required at this stage, or one could set up a chain reaction of unfavorable events. Human error can be minimized with tools that ensure information is accurate, by periodically scanning the IT infrastructure.

Business transaction

This is the third dimension which involves examining user-defined transactions, as they move across the paths defined in second dimension.

Using a subset of this dimension, the focus is on the user defined transactions or the URL page definition that has some meaning to the Business community. There may be 200 to 300 unique pages definitions for any given application however; these can be grouped together into 8-12 high level business transaction categories. This helps begin the process of articulating meaningful SLAs to the business and provides early warning trend reports on performance degradation before it becomes apparent to the majority of the user population.

Since synthetic monitoring uses predefined transactions implemented at known intervals, they are the most suitable data source.

Deep dive component monitoring

This dimension is generally targeted in the middleware space focusing on the Web, application and messaging Servers. It provides the runtime view of the J2EE and .NET stacks, tying them back to the user defined business transactions. Component monitoring covers every element within the IT infrastructure periodically, enabling faults that affect end user experience to be detected and resolved as soon as possible.

A robust solution will give a clear path from the code execution standpoint (e.g. springs, struts, etc.), to the URL rendered, to the user request and where it came from.

Analytics/Reporting

There is a lot of data generated by these tools from the other dimensions. The key to maximizing returns lies in translating this big data correctly. It also enables better forecasting, and more importantly, information about current trends.

A good practice therefore is to collect the raw data from the other tool sets that will enable one to answer a wide variety of performance questions as they arise.

2.6.3 Performance and scalability

To define the conceptual framework for this research, the following definitions and distinction between performance and scalability were considered.

In his book, (Haines, 2006), seeks to clarify the difference between performance and scalability by asserting that the terms “performance” and “scalability” are commonly used interchangeably, but the two are distinct: performance measures the speed with which a single request can be executed, while scalability measures the ability of a request to maintain its performance under increasing load.

He illustrates this definition with an example that the performance of a request may be reported as generating a valid response within three seconds, but the scalability of the request measures the request’s ability to maintain that three-second response time as the user load increases.

Referencing the example above, he adds that scalability asks the following questions about the request:

• At the expected usage, does the request still respond within three seconds?

• For what percentage of requests does it respond in less than three seconds?

• What is the response time distribution for requests that do not respond within three seconds?

In his blog, (Vogels, 2006), proposes that a service is said to be scalable if when there is an increase in the resources in a system, it results in increased performance in a manner proportional to resources added. He further explains that increasing performance in general means serving more units of work, but it can also be to handle larger units of work, such as when datasets grow.

While giving a critique to this definition, (Cecchet, 2006), argues that statement *"A service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added"* is ambiguous. He further explains that, one can have a perfectly scalable system but if no resource is maxed out before adding new resources, it is unlikely that to see any performance improvement.

From his point of view, (Cecchet, 2006), postulates the definition of scalability as constant ratio between workload and throughput. By this definition, he explains that if workload increases proportionally to the resources added, then the throughput should increase in that same proportion.

In yet another effort to define scalability, (Kersey, 2000) posits that:

Scalability for a given application A on a platform P is

$S(A,P) = R(A,P) / C(A,P)$

where

R = Maximum number of requests processed per second by application A on platform P

C = Cost of hardware and software to develop and support application A on platform P

This definition assumes 100% availability for the purposes of the discussion, but adds that availability could be added as an input to the definition, if desired.

By this definition, (Kersey, 2000), states that term displays the expected behavior shown by common usage of the term "scalability" as follows:

- As throughput R increases, scalability increases
- As cost C increases, scalability decreases

In addition, he makes the following assertions:

- Different platforms and different software may be compared using this definition
- The definition can be used to estimate costs of a proposed system, given an anticipated user load.
- Both R and C can be estimated using known techniques.

He summarizes the definition by stating that scalability's dimensions would be "requests processed per second per dollar".

The use of this definition is illustrated in the example below, which compares the scalability of an application on 2 different platforms as follows:

Given the following known values for a single application Z,

Option 1: running on platform X:

R(Z) = 1000 requests/second,

C(Z) = $40,000

S(Z) = 1000 requests/second / $40,000 = 0.025

Option 2: running on not-so-fast but less expensive platform Y:

R(Z) = 500 requests/second,

C(Z) = $10,000

S(Z) = 500 requests/second / $10,000 = 0.05

While platform Y's throughput (performance) is much less than that of platform X, Y is much more scalable than (in fact is twice as scalable as) platform X when running application Z.

In conclusion, (Kersey, 2000), suggests that this definition can also be used to estimate the utility of using various software methodologies. For example, heavy use of components or object technology may or may not change each factor in the definition, however, the degree to which each factor is changed determines whether the resultant system is more or less scalable.

2.6.4 Proposed conceptual framework

Referencing the Gartner conceptual framework for Application Performance Monitoring (APM), the research considers the first and second dimensions of APM model. The research will examine the first dimension of End User Experience (EUE), which captures data on how end-to-end performance impacts the user.

In order to understand the relationship between the architecture of an application and its performance, the research will also examine the second dimension of the APM model which is focused on the runtime architecture of applications.

The conceptual framework is illustrated in the research paradigm diagram below showing the relationship between the independent variable (*Load*), dependent variables (*Performance and Throughput*) and the moderating variable (*Architecture*).

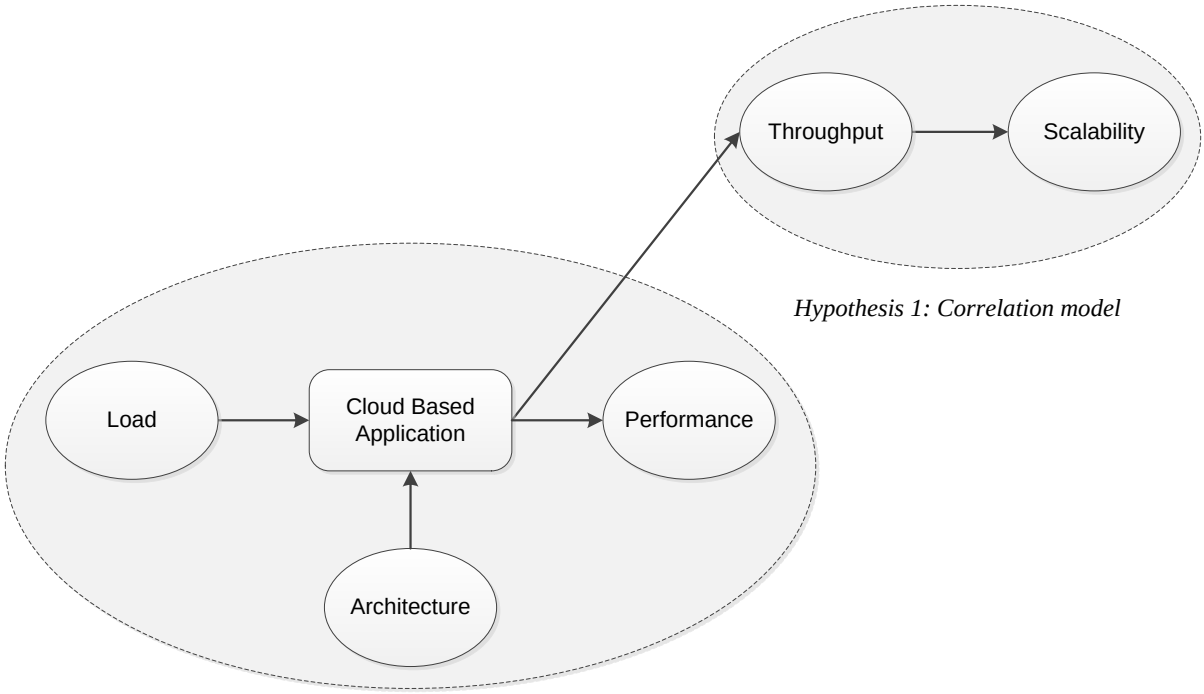

Figure 2.13: Research paradigm diagram for the conceptual model

For this conceptual framework, the definition by (Kersey, 2000) is adopted as follows:

Scalability for a given application A on a platform P is

$$\text{Scalability } S(A,P) = \frac{\text{Throughput } T(A,P)}{\text{Cost } C(A,P)}$$

where

Throughput T(A,P) = maximum number of requests processed per second by application A on platform P

Cost C(A,P) = Cost of hardware and software to develop and support application A on platform P

Using this definition, the variables that were to be measured in the study were:

Independent variable - *Load*

- Load L(A,P), measured by number of concurrent users using application A on platform P

Dependent variable - *Performance*

- Performance R(A,P) measured by the average response time to process a single request by application A on platform P

Dependent variable - *Throughput*

- Throughput T(A,P) measured by the maximum number of requests processed per second by application A on platform P

Moderating variable - *Architecture*

- The application A and platform P together constitute the application architecture in a cloud computing environment which affects the strength of the relationship between load L(A,P) and throughput T(A,P). Application architecture was therefore considered as the moderating variable.

2.6.5 Hypotheses

The following hypotheses were postulated for testing.

Hypothesis 1

Using the definition of scalability by (Kersey, 2000), the following thesis statement was postulated: *As throughput increases, scalability increases. Therefore, there is a positive relationship between throughput and scalability, such that high values of throughput are associated with high values of scalability.*

The study therefore sought to answer the question: *What is the correlation between throughput and scalability of cloud based applications?*

The following hypothesis was tested:

Null hypothesis:

H_0: $\rho = 0$; the correlation coefficient for the population is zero. There is no statistically significant relationship between throughput and scalability of applications.

Alternative hypothesis:

H_1: $\rho \neq 0$; the correlation coefficient for the population is not equal to zero. There is a statistically significant relationship between throughput and scalability of applications.

Hypothesis 2

Based on the conceptual framework, the following thesis statement was postulated: *The architecture of a software application controls how the application utilizes computing resources and therefore impacts the performance of the application when processing load.*

The study therefore sought to answer the question: *Does application architecture moderate the relationship between load and application performance?*

The following hypothesis was tested:

Null hypothesis:

H_0: Application architecture does not moderate the relationship between load and performance.

Alternative hypothesis:

H_1: Architecture does moderate the relationship between load and performance.

2.6.6 Operational definitions

Independent variable - Load

The values of the following indicators control the independent variable *Load*:

i) Number of concurrent users
ii) Number of requests
iii) Number of transactions
iv) Number of database queries

Dependent variable - Performance

The values of the following indicators are examined to show the effect on the dependent variable *Performance*:

i) Response time for a single request
ii) Response time for a single transaction
iii) Response time for a single database query

Dependent variable - Throughput

The values of the following indicators are examined to show the effect on the dependent variable *Throughput*:

i) Number of requests processed within a specific timeframe
ii) Number of transactions processed within a specific timeframe
iii) Number of database queries processed within a specific timeframe

Moderating Variable - Architecture

For the moderating variable *Architecture*, the following indicators show the utilization of computing resources (Haines, 2006) in the duration of the study:

i) CPU utilization
ii) Physical memory utilization
iii) Operating system disk I/O rates

iv) Operating system thread/process utilization

v) Application server thread pool utilization

vi) Application server connection pool utilization

vii) Application server heap utilization and garbage collection rates (frequency and duration)

viii) Application server cache and pool utilizations

ix) Messaging system utilizations

x) Network traffic sent between application nodes

2.6.7 Graduated Load Test

A graduated load tester proposed by (Haines, 2006) is configured to climb to the expected usage in a regular and predefined pattern and then increase load in graduated steps. The purpose behind this configuration is to allow the researcher to capture and analyze performance metrics at discrete units of load. The behavior of graduated load generation is illustrated in the figure below:

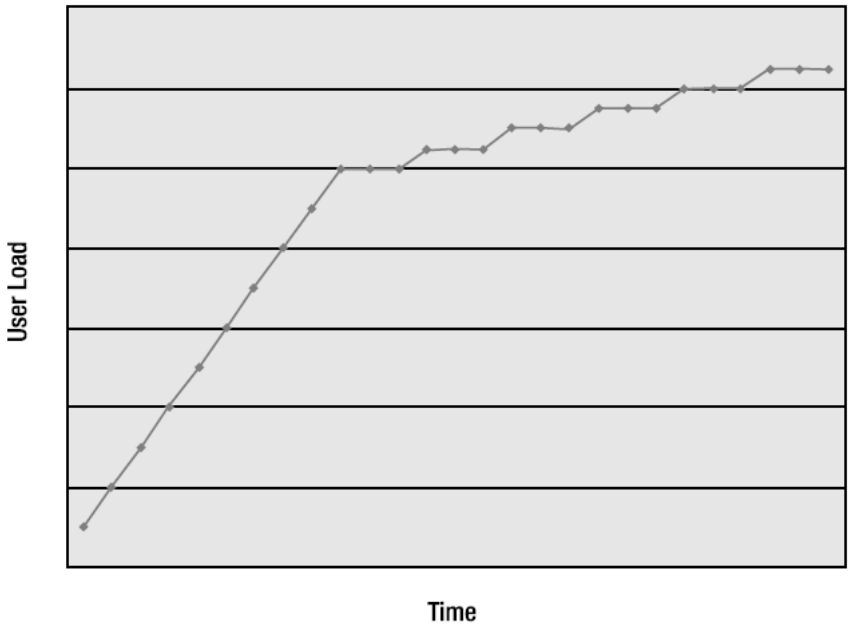

Figure 2.14: Graduated load test

2.6.8 Enterprise application behavior pattern

By following the graduated load test approach, the behavior of a loaded enterprise application follows the typical pattern illustrated in the figure below (Haines, 2006).

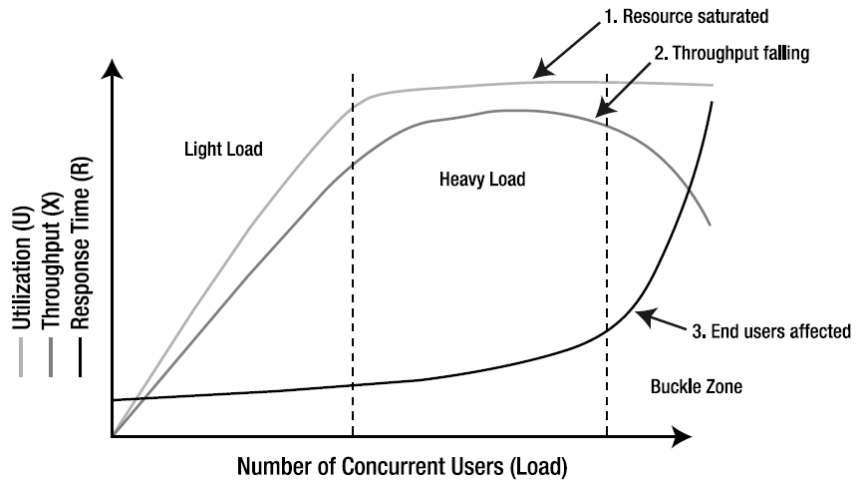

Figure 2.15: A loaded enterprise application follows this typical pattern

3 RESEARCH METHODOLOGY

3.1 Research design

Experimental research method (Explorable, 2008), was used in this study.

This study sought to find answers to the following questions:

- How does changing work load affect the response time of an application request?
- How does changing work load affect the throughput of an application?
- How does changing work load affect utilization of computing resources?

In order to answer these questions, an experiment was setup using a cloud based platform with computing resources to conduct various application performance tests in line with the research hypotheses and the conceptual framework.

Due to the sampling method used to select the test applications and the fact that there was no control group, this research method fell in the category of quasi-experiment design.

3.1.1 Independent variable

The independent variable that was manipulated, or treatment variable was *load*. The treatment was administered by following the graduated load test approach (Haines, 2006).

The actual load conditions that were manipulated in the experiment included:

- Number of concurrent users
- Number of requests
- Number of transactions
- Number of database queries

3.1.2 Dependent variables

The effect of treatment applied to the independent variable was measured by the dependent variable *performance* and *throughput*.

In particular the following factors were measured:

- Response time for a single request
- Response time for a single transaction
- Response time for a single database query
- Number of requests processed within a specific timeframe
- Number of transactions processed within a specific timeframe
- Number of database queries processed within a specific timeframe

3.1.3 Sampling technique

For this experiment, convenience sampling technique was used. This is non-probability sampling technique where subjects are selected because of their convenient accessibility and proximity to the researcher (Explorable, 2009). With this technique, 17 web applications were identified as available and accessible for performing graduated load tests.

3.1.4 Experimental group

This was the set of 17 web based applications that were selected using the convenience sampling technique. These applications were subjected to the same treatment by conducting the graduated load test and observing their performance.

3.1.5 Control group

For this experiment, there was no control group. The effects of the treatment, the graduated load tests, could only be observed in the applications in the sample group.

3.1.6 Factors held constant

The cloud computing test platform, provided a consistent environment for conducting the experiment. The platform provided access to the consistent level of computing resources throughout the experimentation process. These computing resources included CPU, memory, storage and network.

3.1.7 Cause and effect

This experiment was concerned with determining the effect of manipulating the independent variable (*load*) on the dependent variable (*performance and throughput*).

At the same time, the experiment was to determine how the moderating variable (*architecture*) was affecting the relationship between the independent variable (*load*) and the dependent variable (*performance*).

3.1.8 Data collection

The data collection process started by identifying web based applications that were already developed and made available for public testing which formed the treatment group of applications.

For each application, the graduated load test was conducted and the performance data recorded. The graduated load test process was repeated for each application in the sample population.

Throughout the experiment, values of the independent variable were recorded and the effects on the dependent variable observed and recorded using the tools available in the laboratory cloud computing environment.

3.2 Experiment laboratory environment

A cloud simulator platform was used to test the performance of different applications.

The cloud simulator was based on Microsoft Visual Studio Teams Service (www.visualstudio.com) on Microsoft Azure portal (https://azure.microsoft.com). This is a Cloud-based load testing environment leveraging Microsoft Azure cloud computing resources and services. The following performance testing capabilities are provided on this platform.

3.2.1 Cloud scalability testing

Using the platform, one can generate hundreds of thousands of connections in minutes and therefore test the performance of applications before they are launched or before updates are deployed to production.

Test load settings can be configured on the platform as illustrated in the figure below.

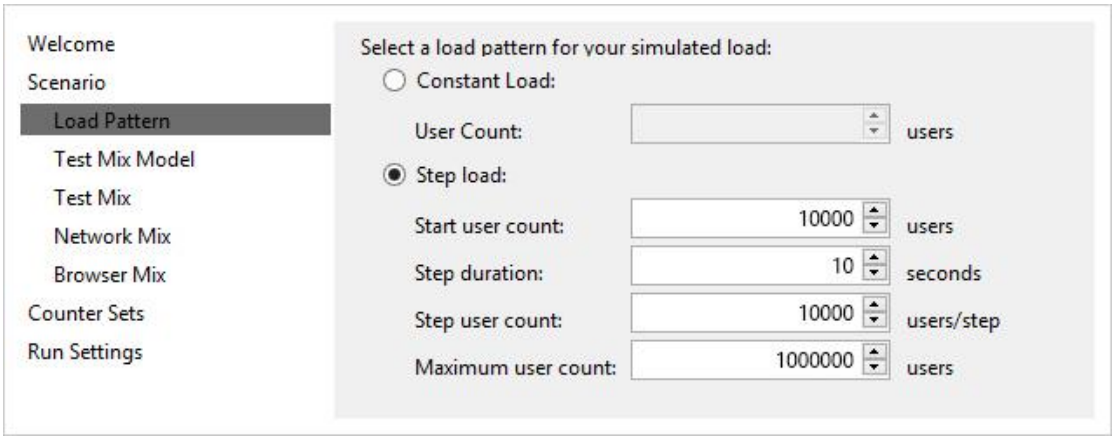

Figure 3.1: Load settings for performance testing

3.2.2 Functionality testing

With authoring experiences in Visual Studio, Azure and VSTS the platform enables one to create load tests by specifying a website, referencing Apache JMeter test file or recording and replaying user actions in an application. The load tests are then run using Visual Studio.

There are also pre-existing unit or functional tests that can be used to generate load for the performance tests as illustrated in the figure below.

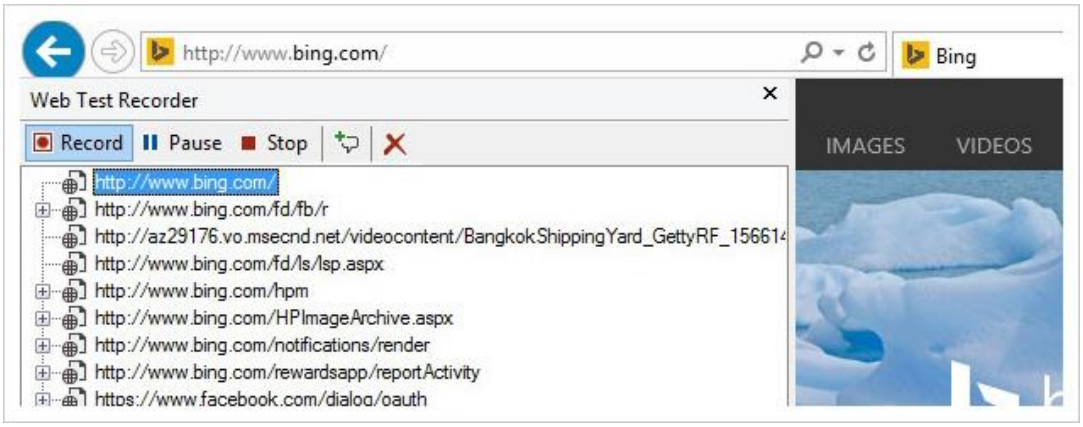

Figure 3.2: Function testing using Apache JMeter test file

3.2.3 Deep reporting and analytics

The platform also enables one to view application performance with real-time charts and graphs as illustrated below.

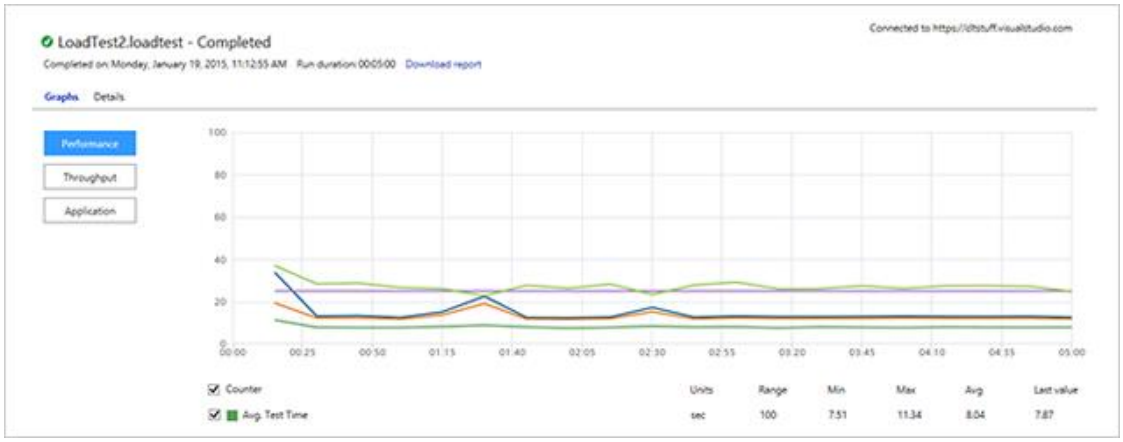

Figure 3.3: Real-time application performance charts and graphs

In addition, the platform provides application insights and correlate test results with server diagnostics.

3.2.4 Location based testing

Using the Azure cloud platform, one can run tests from one of many global Azure data center locations located around the world to minimize latency and simulate users' real-world conditions as illustrated below.

Figure 3.4: Microsoft Azure data centers locations around the world

3.3 Data analysis

3.3.1 Pearson Correlation Coefficient analysis

Data analysis was conducted using the Pearson product-moment correlation coefficient (or Pearson correlation coefficient, for short), which is a measure of the strength of a linear association between two variables and is denoted by r. Basically, a Pearson product-moment correlation attempts to draw a line of best fit through the data of two variables, and the Pearson correlation coefficient, r, indicates how far away all these data points are to this line of best fit (i.e., how well the data points fit this new model/line of best fit) (Statistics.laerd.com, 2013).

3.3.2 Moderation Multiple Regression analysis

The moderation multiple regression analysis by (Hayes, 2017) and discussed by (Cooper, 2015) was used to quantify the effect of the moderating variable (*architecture*) on the strength and/or direction of the relationship between the independent variable (*load*) and the dependent variable (*performance*).

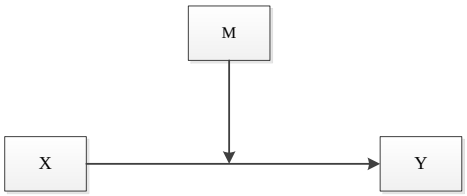

Figure 3.5: Conceptual diagram

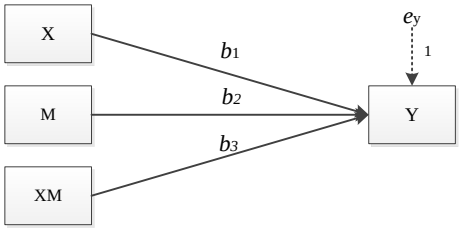

Figure 3.6: Statistical diagram

The conditional effect of X on Y at a given value of M is defined as:

$Y = b_1 + b_3M$

Moderated regression equation is defined as:

$Y = b_0 + b_1X + b_2M + b_3XM + e$

3.4 Strengths and limitations of the methodology

Following is a detailed discussion regarding both the advantages and the limitations or disadvantages of experimental research (Cirt.gcu.edu, 2017).

Strengths

- Experimental research is the most appropriate way for drawing causal conclusions, regarding interventions or treatments and establishing whether or not one or more factors causes a change in an outcome. This is largely due to the emphasis in controlling extraneous variables. If other variables are controlled, the researcher can say with confidence that manipulation independent variable caused a changed in the dependent variable.
- It is a basic, straightforward, efficient type of research that can be applied across a variety of disciplines.
- Experimental research designs are repeatable and therefore, results can be checked and verified.
- Due to the controlled environment of experimental research, better results are often achieved.
- In the case of laboratory research, conditions not found in a natural setting can be created in an experimental setting that allows for greater control of extraneous variables. Conditions that may take longer to occur in a natural environment may occur more quickly in an experimental setting.
- There are many variations of experimental research and the researcher can tailor the experiment while still maintaining the validity of the design.

Limitations

- Experimental research can create artificial situations that do not always represent real-life situations. This is largely due to fact that all other variables are tightly controlled which may not create a fully realistic situation.
- Because the situations are very controlled and do not often represent real life, the reactions of the test subjects may not be true indicators of their behaviors in a non-experimental environment.

- Human error also plays a key role in the validity of the project as discussed in previous modules.
- It may not be really possible to control all extraneous variables. The health, mood, and life experiences of the test subjects may influence their reactions and those variables may not even be known to the researcher.
- The research must adhere to ethical standards in order to be valid. These will be discussed in the next module of this series.
- Experimental research designs help to ensure internal validity but sometimes at the expense of external validity. When this happens, the results may not be generalizable to the larger population.
- If an experimental study is conducted in its natural environment, such as a hospital or community, it may not be possible to control the extraneous variables.
- Experimental research is a powerful tool for determining or verifying causation, but it typically cannot specify "why" the outcome occurred.

4 RESULTS AND DISCUSSIONS

4.1 Chapter overview

The findings of the study are presented and discussed in this chapter.

Attempts have been made to extract common trends that exist in support of or in contradiction to the hypotheses stated in the conceptual framework.

Inferential statistical analysis on the results has also been incorporated.

4.2 Test applications

The table below outlines the web applications that formed the experiment group.

Table 4.1: List of web applications used in the experiment

No.	Application ID	Web application URL	Description
1	WebApp_1	http://automationpractice.com	End-to-end e-commerce website
2	WebApp_2	http://newtours.demoaut.com	Tours & Travel booking web application
3	WebApp_3	http://www.practiceselenium.com	Generic website with static html pages
4	WebApp_4	http://zero.webappsecurity.com	Online banking application
5	WebApp_5	http://demo.nopcommerce.com	Fully functional e-Commerce site
6	WebApp_6	http://www.globalsqa.com	Generic website with HTML Modules
7	WebApp_7	http://store.demoqa.com	Basic e-commerce web application
8	WebApp_8	http://awful-valentine.com	Basic e-commerce web application
9	WebApp_9	http://demo.borland.com	Insurance company web application
10	WebApp_10	http://phptravels.com	Online Travel operations web application
11	WebApp_11	http://demoqa.com	Generic website with rich UI functions
12	WebApp_12	http://thedemosite.co.uk	Generic website
13	WebApp_13	http://www.way2automation.com	Generic website
14	WebApp_14	https://www.ultimateqa.com	Generic website
15	WebApp_15	https://www.qtptutorial.net	Generic website
16	WebApp_16	http://ibm.github.io	Portal for IBM open source at GitHub
17	WebApp_17	http://square.github.io	A simple, static portal

4.3 Graduated load test parameters

For each web application, the graduated load test was conducted with the following parameters:

Table 4.2: Test parameters used in the graduated load tests

Test Parameter	Value
Run duration (minutes)	5
Load pattern	Step
Max v-users	200
Start user count	10
Step duration (seconds)	10
Step user count (users/step)	10
Warmup duration (seconds)	0
Browser mix	IE – 60%, Chrome – 40%
Geo-location	West US (California)

4.4 Graduated load test pattern

The graph below shows the graduated load test pattern achieved with the test parameters above. From the graph, it was observed that the number of concurrent users increased steadily by 10 users every 10 seconds from the initial 10 users to the set maximum of 200 concurrent users. The maximum number of concurrent users was achieved after 3 minutes and 10 seconds. The test continued up to the set period of 5 minutes.

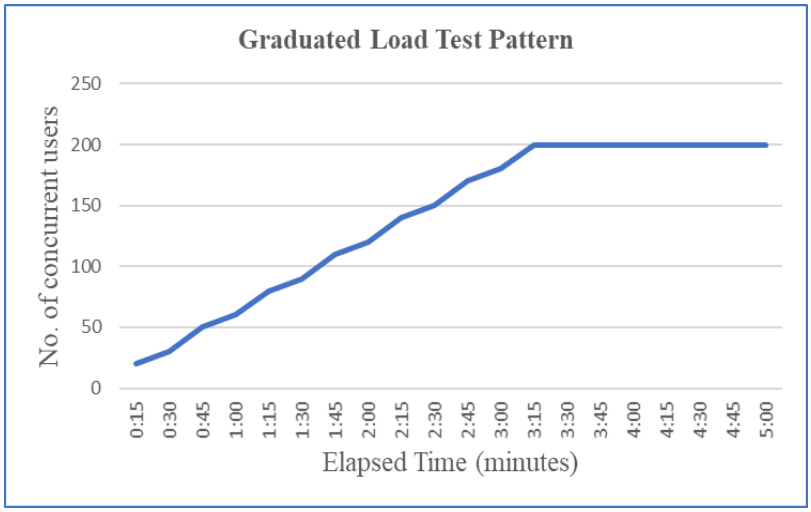

Figure 4.1: Graduated load test pattern achieved

4.5 Application test results

The section below discusses the test results for each of the web applications that were tested. The results show 4 dimensions of the results that include: performance, throughput, errors and the tests conducted.

4.5.1 Test Application WebApp_1

This website is an end-to-end e-commerce web application. Its operation provides back and forth interactions between server and client.

The table below shows the test results for the graduated load test for this application:

Table 4.3: Graduated load test results for WebApp_1

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	6.000000	1.922222	-	23.437500	2,834
0:30	30	13.733330	1.796116	-	26.875000	2,833
0:45	50	22.066670	1.858006	-	38.125000	2,839
1:00	60	24.133330	1.872928	-	29.895830	2,874
1:15	80	29.933330	2.167038	0.066667	42.812500	2,858
1:30	90	37.266670	1.919499	-	36.666670	2,846
1:45	110	44.533330	1.697605	-	37.604170	2,813
2:00	120	49.533330	1.776581	0.066667	37.500000	2,778
2:15	140	55.933330	1.722288	-	35.520830	2,774
2:30	150	59.466670	1.791480	-	22.187500	2,766
2:45	170	62.533330	1.826226	-	19.479170	2,782
3:00	180	66.866670	1.794616	-	23.020830	2,768
3:15	200	75.933330	1.828797	0.200000	18.020830	2,750
3:30	200	78.000000	1.815385	0.066667	20.520830	2,752
3:45	200	76.866670	1.869037	-	16.041670	2,751
4:00	200	77.333340	1.859483	-	15.625000	2,749
4:15	200	72.400000	1.988950	0.066667	11.145830	2,746
4:30	200	75.133330	1.913931	-	13.437500	2,752
4:45	200	81.800000	1.740016	-	16.458330	2,758
5:00	200	74.800000	1.850267	-	13.541670	2,759

These test results are further represented in the graphs below and discussed in the respective sections next to each graph.

Performance

It was notable that the average response time remains relatively steady throughout the test in spite of the increasing user load. Similarly, the average page load time increases only marginally through the test.

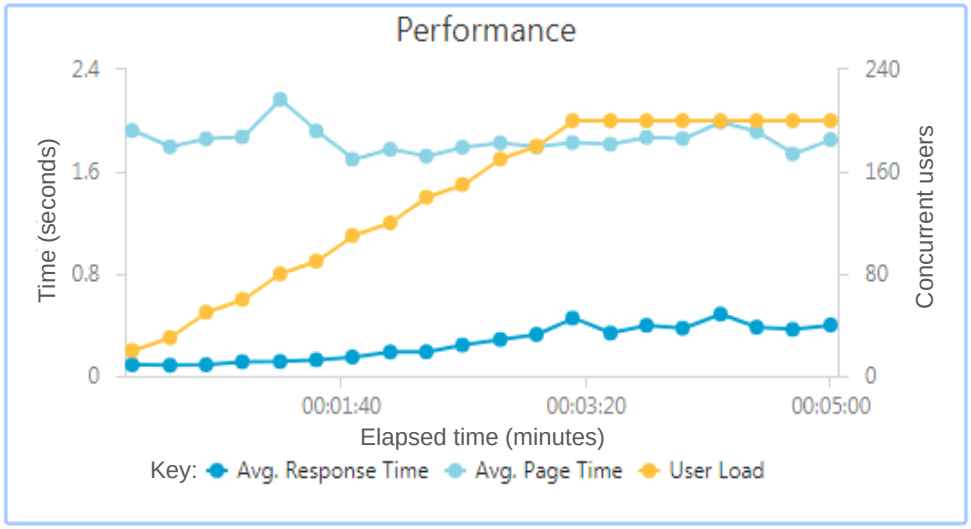

Figure 4.2: Graduated load test "Performance" data for WebApp_1

Throughput

It was notable that the number of requests processed per second increased at the initial stage of the test then decreased consistently as the user load increased. On the other hand, the number of pages loaded per second increased marginally throughout the test period.

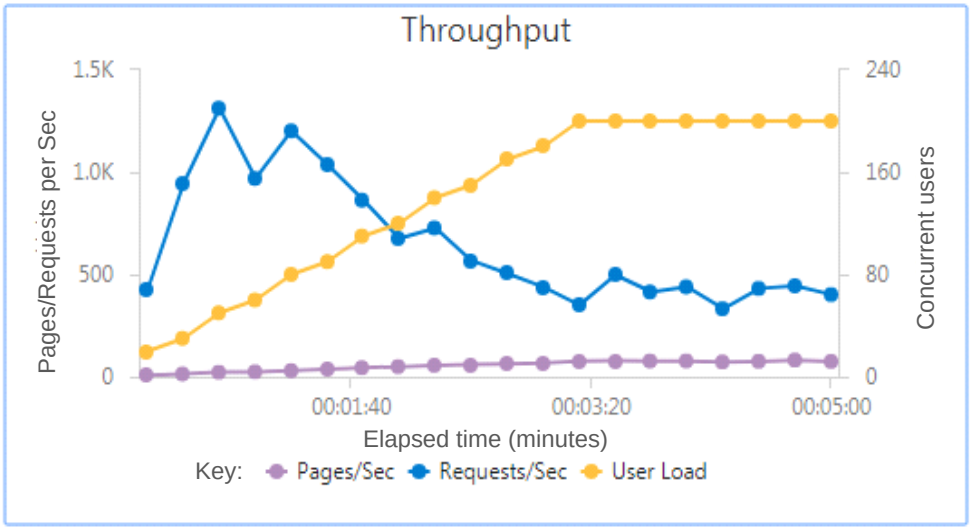

Figure 4.3: Graduated load test "Throughput" data for WebApp_1

Errors

From the errors graph below, it was notable that failed requests were recorded at 5 points during the test, though the rate remained relative low.

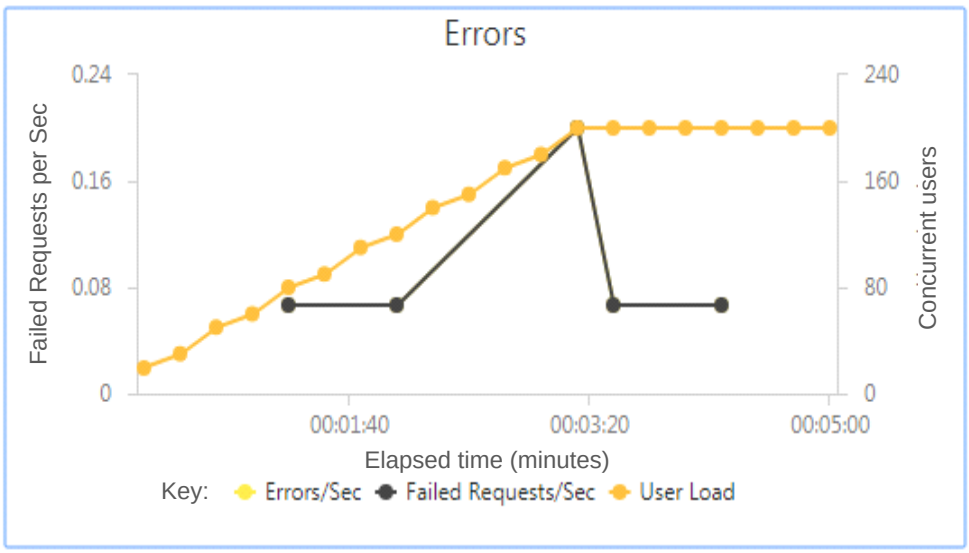

Figure 4.4: Graduated load test "Errors" data for WebApp_1

Tests

From the graph below, it was notable that the number of tests processed per second was consistent with the increasing user load throughout the test period. At the same time, the average test time remained consistently low throughout the test period.

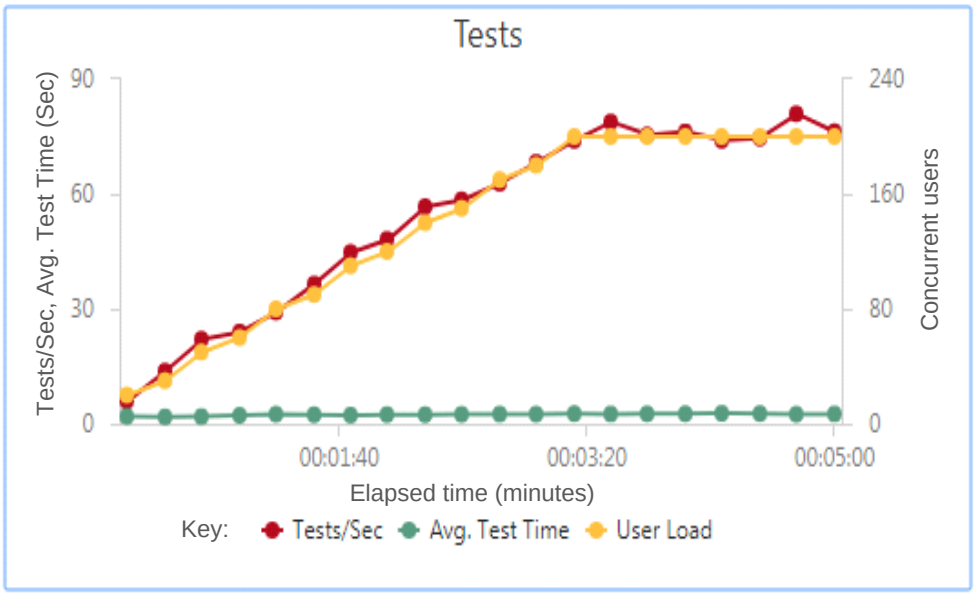

Figure 4.5: Graduated load test "Tests" data for WebApp_1

Discussion

The results showed that the test application, WebApp_1 (http://automationpractice.com), which is an end-to-end e-commerce web application, has a constrained architectural design that provided high average page load time throughout the test period, even though the average response time increased only marginally with increase in user load.

The application design exhibited low scalability as throughput decreased with increasing user load and page load errors were recorded at several points during the test.

4.5.2 Test Application WebApp_2

The table below shows the test results for the graduated load test for the Tours & Travel booking web application:

Table 5.4: Graduated load test results for WebApp_2

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	9.800000	0.530612	19.866670	7.812500	2,748
0:30	30	20.266670	0.378290	40.600000	8.229167	2,738
0:45	50	30.000000	0.426667	60.666670	11.875000	2,708
1:00	60	30.333330	1.032967	61.733330	11.458330	2,704
1:15	80	24.000000	2.322222	47.466670	8.333333	2,709
1:30	90	23.800000	2.378151	48.800000	9.270833	2,704
1:45	110	21.533330	3.690403	43.333330	6.458333	2,704
2:00	120	19.866670	4.104027	40.866660	9.687500	2,700
2:15	140	22.800000	4.514620	47.400000	8.229167	2,693
2:30	150	17.800000	6.101124	36.933330	6.979167	2,692
2:45	170	21.866670	5.923780	43.066670	14.270830	2,688
3:00	180	21.266670	5.789968	44.400000	8.229167	2,690
3:15	200	21.400000	7.461059	44.466670	10.208330	2,682
3:30	200	19.466670	8.890411	39.800000	7.916667	2,681
3:45	200	23.000000	7.965218	44.733330	8.229167	2,684
4:00	200	25.466670	7.664921	49.933330	12.083330	2,683
4:15	200	21.133330	7.842271	44.400000	11.145830	2,686
4:30	200	22.866670	8.865890	45.000000	9.270833	2,691
4:45	200	26.866670	6.848635	53.266670	9.687500	2,692
5:00	200	25.600000	7.541667	51.866660	10.937500	2,693

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that the average response time increased with the increase in user load. Similarly, it was observed that the average page load time increased consistently with the increase in user load.

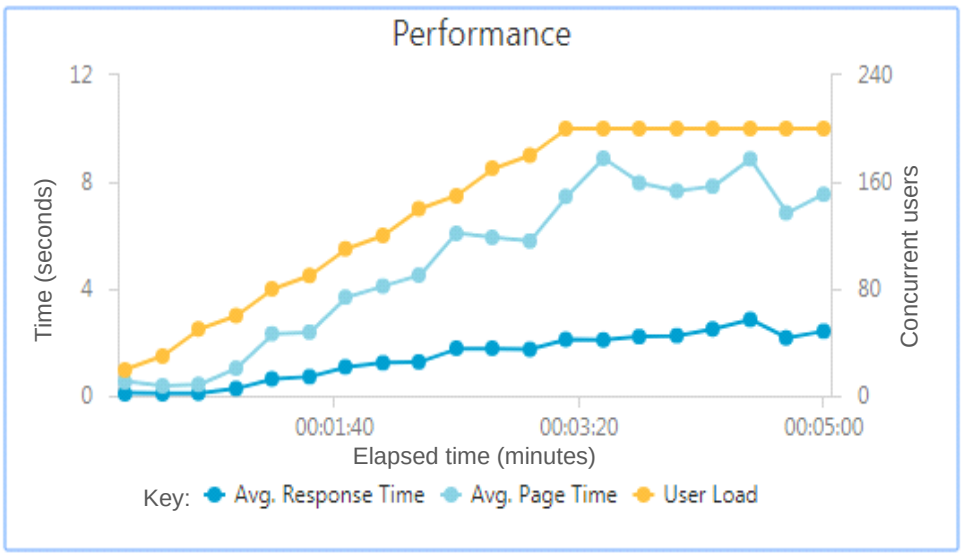

Figure 4.6: Graduated load test "Performance" data for WebApp_2

Throughput

From the throughput results graphed below, it was notable that the number of requests that are successfully processed per second increases steadily from an initial rate of about 160 requests per second to a peak of about 300 requests per second, when the user load of about 50 concurrent users. However, from this point, the rate of requests drops steadily with every increase in the user load. It can be argued that that the success rate of processing requests decreases with increase in user load.

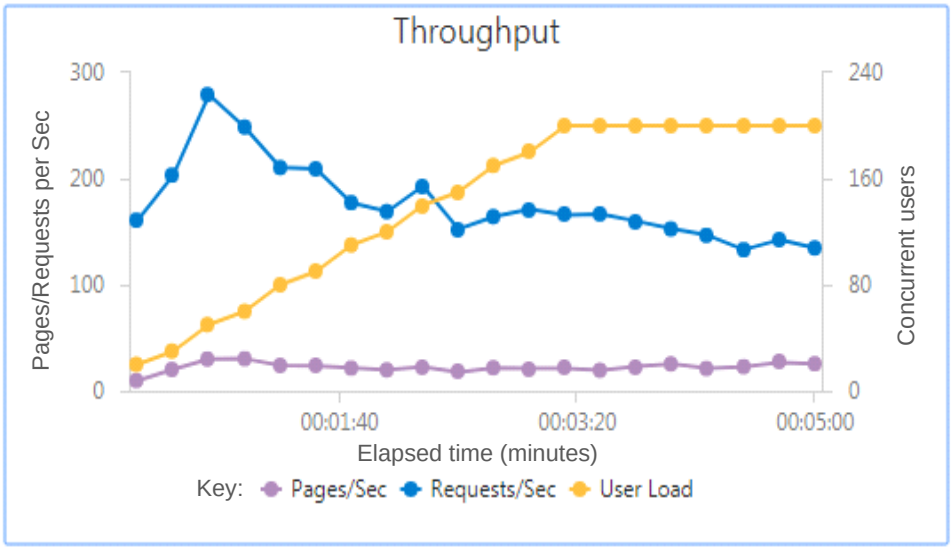

Figure 4.7: Graduated load test "Throughput" data for WebApp_2

On the other hand, the number of pages loaded per second increases marginally from the starting rate of about 9 pages per second to a peak rate of about 30 pages per second.

Errors

From the errors graph below, it was notable that failed requests were recorded right from the start of the test and the rate remained high throughout the test period.

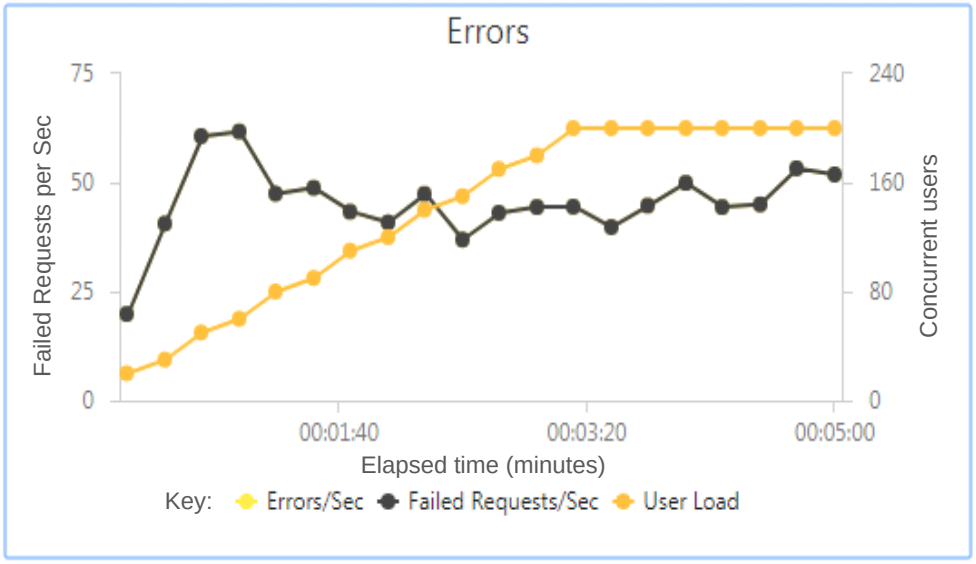

Figure 4.8: Graduated load test "Errors" data for WebApp_2

Tests

From the graph below, it was notable that the number of tests processed per second was not consistent with the number of concurrent users as observed with the previous test application. On the other hand, the average test time increased marginally with the increasing user load.

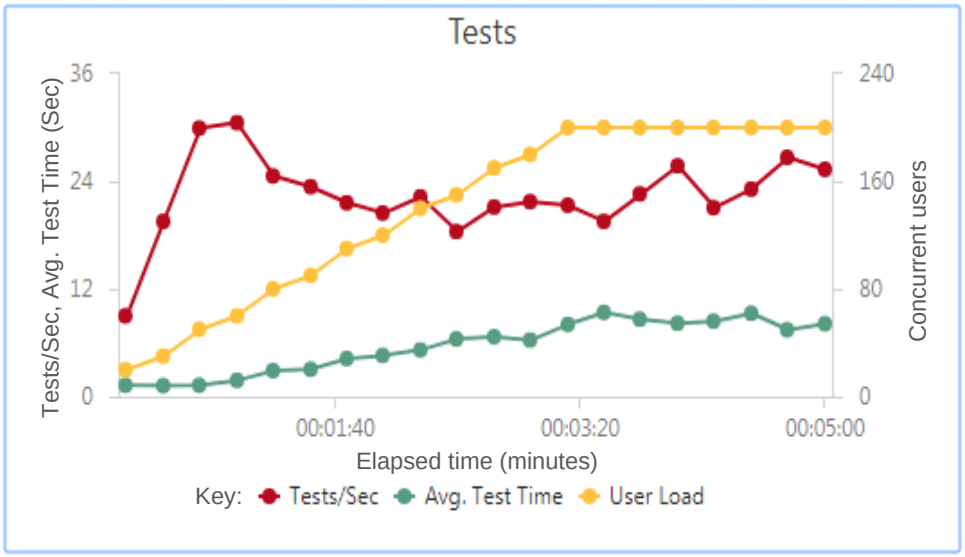

Figure 4.9: Graduated load test "Tests" data for WebApp_2

Discussion

The results showed that the test application, WebApp_2 (http://newtours.demoaut.com), which is a Tours & Travel booking web application, has a constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

The application design exhibited low scalability as throughput decreased with increasing user load and page load errors were recorded at several points during the test.

With increasing user load, the number of failed requests increased and remained high throughout the test period.

4.5.3 Test Application WebApp_3

This website has static html pages, so it was expected to handle load very effectively. The application does not have any back and forth interactions between client and server.

The table below shows the test results for the graduated load test for this application:

Table 4.5: Graduated load test results for WebApp_3

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	11.800000	0.237288	-	17.916670	2,846.0
0:30	30	24.000000	0.177778	-	16.770830	2,853.0
0:45	50	40.200000	0.182421	-	23.229170	2,856.0
1:00	60	51.733330	0.179124	-	34.583330	2,882.0
1:15	80	67.000000	0.183085	-	40.625000	2,837.0
1:30	90	77.733330	0.187822	-	50.312500	2,717.0
1:45	110	91.600000	0.194323	-	60.520830	2,750.0
2:00	120	103.466700	0.192655	-	66.666660	2,707.0
2:15	140	116.133300	0.228473	-	71.458340	2,699.0
2:30	150	126.066700	0.231095	-	70.208340	2,626.0
2:45	170	139.133300	0.263057	-	75.520840	2,646.0
3:00	180	141.400000	0.327204	-	80.208340	2,632.0
3:15	200	149.600000	0.408645	-	87.708340	2,634.0
3:30	200	155.666700	0.425268	-	84.791660	2,619.0
3:45	200	159.733300	0.417780	-	85.000000	2,618.0
4:00	200	155.000000	0.434409	-	84.687500	2,612.0
4:15	200	151.800000	0.458498	-	83.645840	2,623.0
4:30	200	152.133300	0.444785	-	85.312500	2,660.0
4:45	200	155.000000	0.447742	-	84.895840	2,654.0
5:00	200	148.000000	0.416667	-	83.437500	2,649.0

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained consistently low throughout the test period, as depicted by the Y-Axis scale of between 0 and 0.6 seconds. The average page load time increased consistently with the increase in user load in a similar pattern like the average response time.

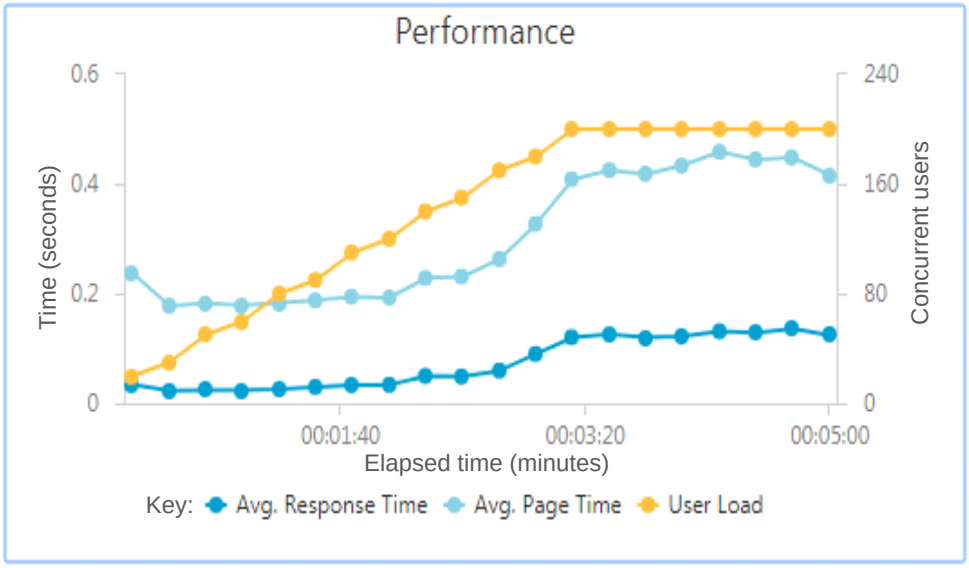

Figure 4.10: Graduated load test "Performance" data for WebApp_3

Throughput

From the throughput results graphed below, it was notable that the number of requests processed per second increased consistently with the increase in the number of concurrent users. It can be argued that that throughput increased with increase in user load.

On the other hand, the number of pages loaded per second increased marginally throughout the test period.

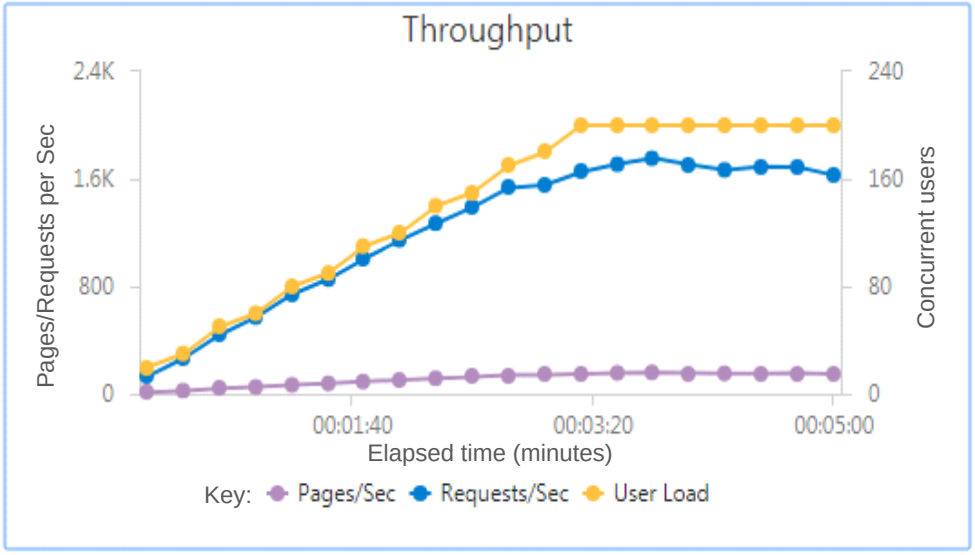

Figure 4.11: Graduated load test "Throughput" data for WebApp_3

Errors

From the errors graph below, it was notable that no failed requests were recorded throughout the test period.

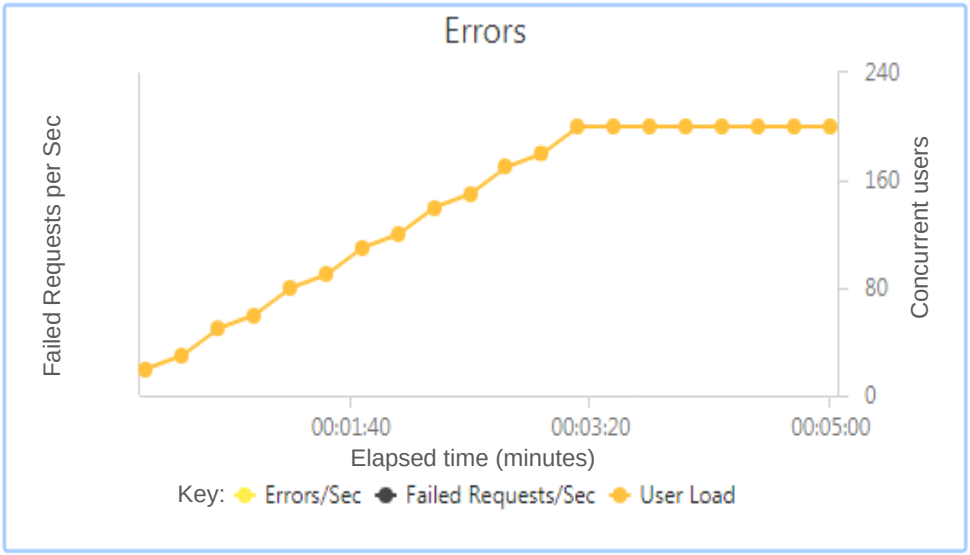

Figure 4.12: Graduated load test "Errors" data for WebApp_3

Tests

From the graph below, it was notable that the number of tests processed per second was consistent with the increasing user load throughout the test period. At the same time, the average test time remained consistently low throughout the test period.

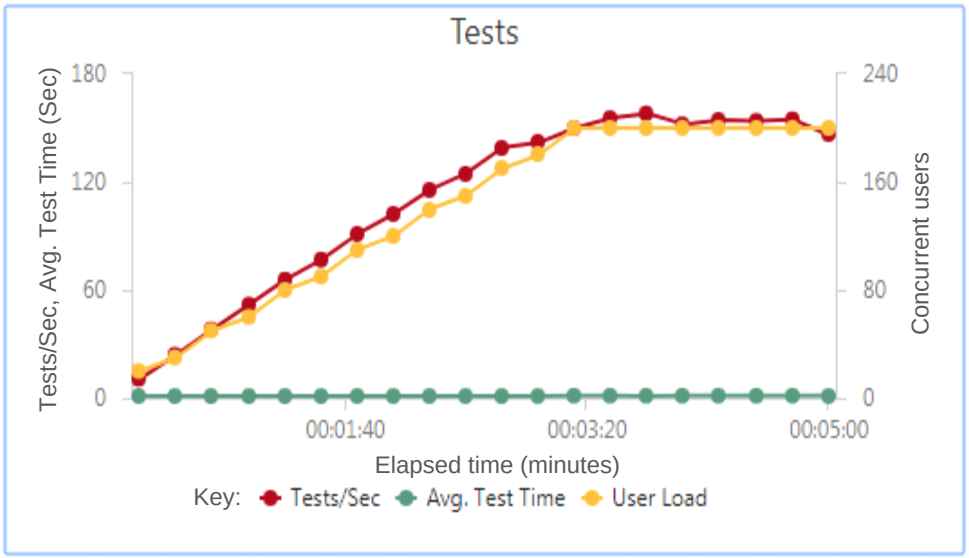

Figure 4.13: Graduated load test "Tests" data for WebApp_3

Discussion

The results show that the test application, WebApp_3 (http://www.practiceselenium.com), which is a static HTML website, has a good architectural design and was able to provide a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load and no page load errors were recorded during the test.

4.5.4 Test Application WebApp_4

The Free Online Bank Web site is published by Hewlett-Packard, Company for the sole purpose of demonstrating the functionality and effectiveness of Hewlett-Packard Fortify's WebInspect products in detecting and reporting Web application vulnerabilities.

The table below shows the test results for the graduated load test for this application:

Table 4.6: Graduated load test results for WebApp_4

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	12.200000	0.054645	-	9.270833	2,735
0:30	30	26.666670	0.017500	-	4.687500	2,728
0:45	50	43.133340	0.021638	-	6.562500	2,715
1:00	60	55.733330	0.017943	-	5.312500	2,689
1:15	80	71.800000	0.020427	-	8.020833	2,676
1:30	90	85.800000	0.018648	-	9.687500	2,678
1:45	110	98.933330	0.020889	-	13.437500	2,643
2:00	120	116.400000	0.020046	-	12.083330	2,644
2:15	140	128.733300	0.020715	-	17.395830	2,634
2:30	150	144.200000	0.019880	-	15.312500	2,598
2:45	170	156.333300	0.021748	-	17.291670	2,610
3:00	180	173.133300	0.020793	-	17.083330	2,607
3:15	200	187.800000	0.022009	-	23.125000	2,598
3:30	200	193.933300	0.019938	-	18.645830	2,597
3:45	200	193.066700	0.024517	-	19.583330	2,589
4:00	200	194.133300	0.018887	-	23.645830	2,592
4:15	200	193.866700	0.051238	-	16.770830	2,585
4:30	200	186.666700	0.022500	-	19.062500	2,586
4:45	200	196.733300	0.019993	-	15.937500	2,599
5:00	200	193.533300	0.018601	-	14.895830	2,601

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained consistently low throughout the test period, as depicted by the Y-Axis scale of between 0 and 0.06 seconds.

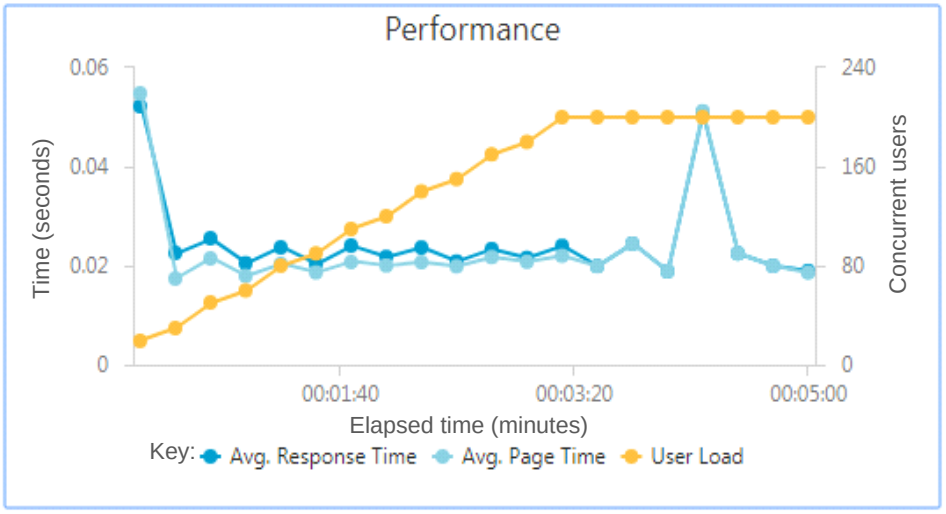

Figure 4.14: Graduated load test "Performance" data for WebApp_4

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that are successfully processed per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

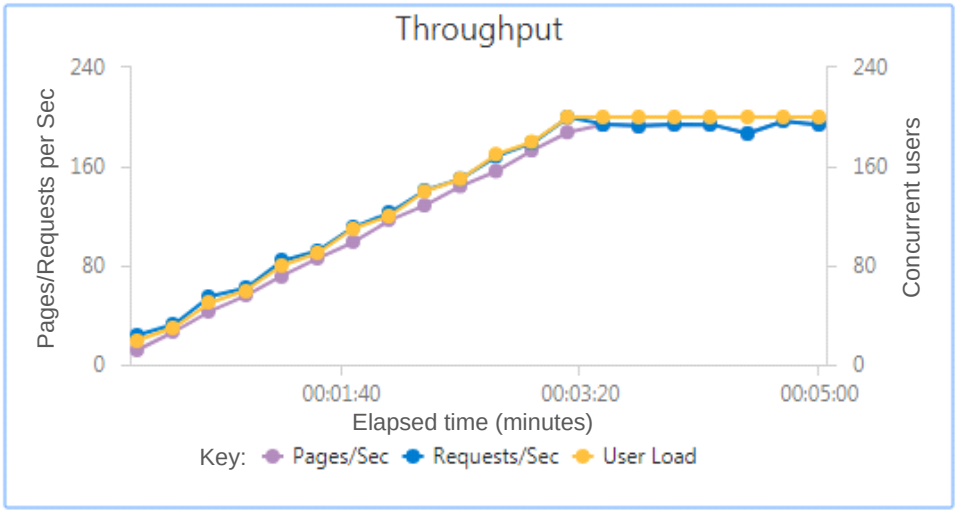

Figure 4.15: Graduated load test "Throughput" data for WebApp_4

Errors

From the errors graph below, was notable that no failed requests were recorded throughout the test period.

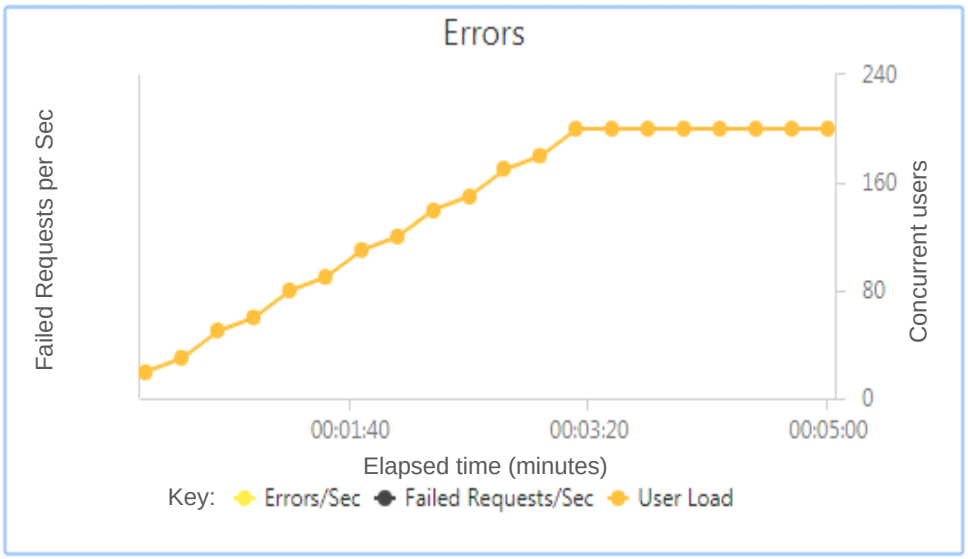

Figure 4.16: Graduated load test "Errors" data for WebApp_4

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained steady throughout the test period.

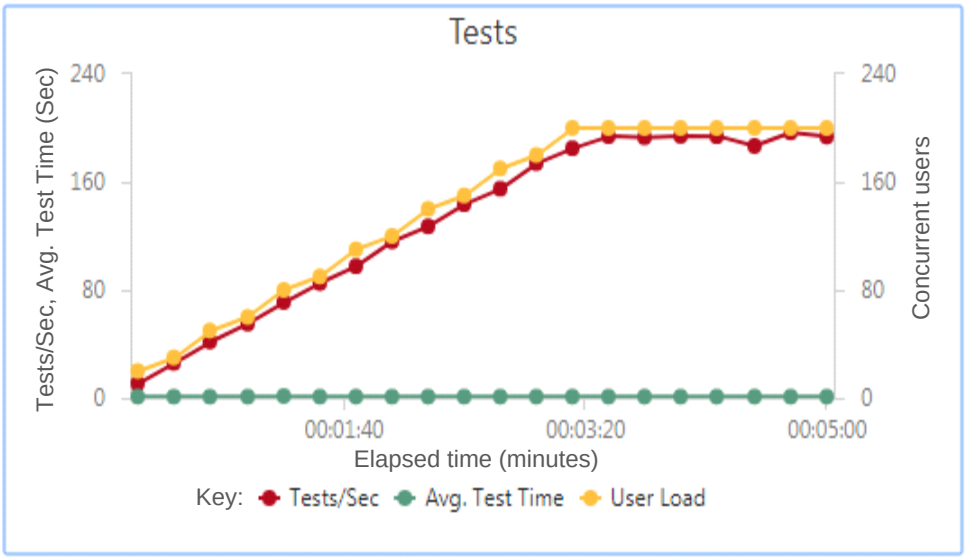

Figure 4.17: Graduated load test "Tests" data for WebApp_4

Discussion

The results showed that the test application, WebApp_4 (http://zero.webappsecurity.com), which is an online banking application, has a good architectural designed and was able to provide a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users. No failed page requests were experienced throughout the test period.

The application design also exhibited high scalability as throughput increased with increasing user load and no page load errors were recorded during the test.

4.5.5 Test Application WebApp_5

Fully functional e-Commerce site allowing performance testing for an e-Commerce application that supports interactions between client and server.

The table below shows the test results for the graduated load test for this application:

Table 4.7: Graduated load test for WebApp_5

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	12.200000	0.114754	5.333333	15.520830	2,900
0:30	30	25.933330	0.046272	10.666670	5.625000	2,906
0:45	50	41.600000	0.046474	17.200000	15.833330	2,904
1:00	60	54.000000	0.049383	22.666670	11.666670	2,866
1:15	80	66.933330	0.108566	29.333330	20.312500	2,842
1:30	90	76.266670	0.085664	34.600000	21.354170	2,778
1:45	110	89.800000	0.170007	41.133340	26.666670	2,755
2:00	120	100.266700	0.174202	46.000000	17.187500	2,746
2:15	140	118.466700	0.129994	53.266670	19.895830	2,745
2:30	150	132.466700	0.099648	57.666670	18.750000	2,740
2:45	170	139.533300	0.164835	64.933330	21.458330	2,739
3:00	180	134.866700	0.281265	69.733330	17.708330	2,727
3:15	200	137.200000	0.411565	76.200000	17.083330	2,713
3:30	200	145.600000	0.390568	80.000000	15.520830	2,718
3:45	200	141.866700	0.388158	79.933330	15.312500	2,718
4:00	200	139.800000	0.442537	78.333340	14.375000	2,718
4:15	200	147.666700	0.342212	79.800000	17.708330	2,725
4:30	200	153.000000	0.307625	79.066670	17.916670	2,722
4:45	200	154.266700	0.299481	79.066670	18.645830	2,720
5:00	200	143.000000	0.355711	78.866670	16.458330	2,723

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased in an irregular pattern as the user load increased. However, the values for the two metrics remained relatively low throughout the test period, as depicted by the y-axis scale of between 0 and 0.06 seconds.

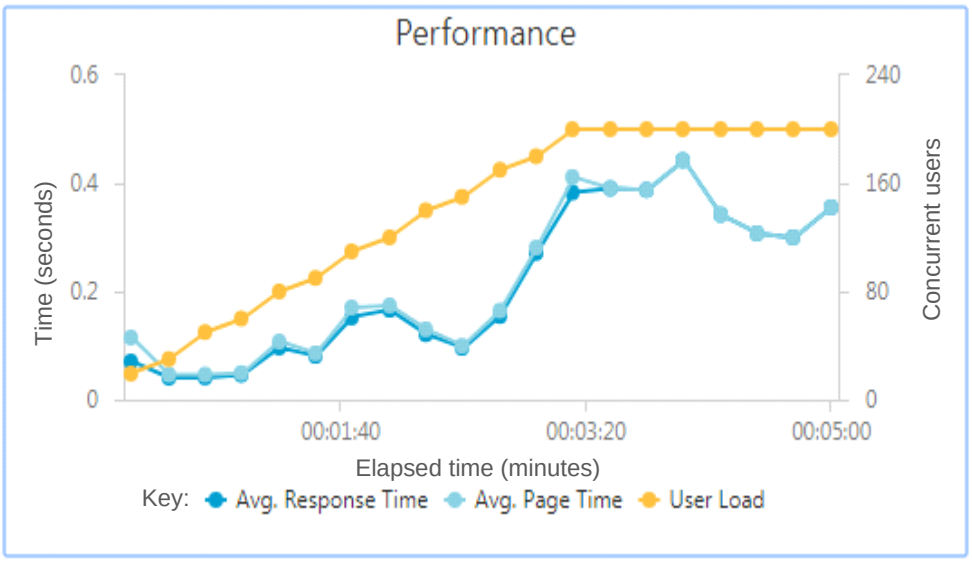

Figure 4.18: Graduated load test "Performance" data for WebApp_5

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

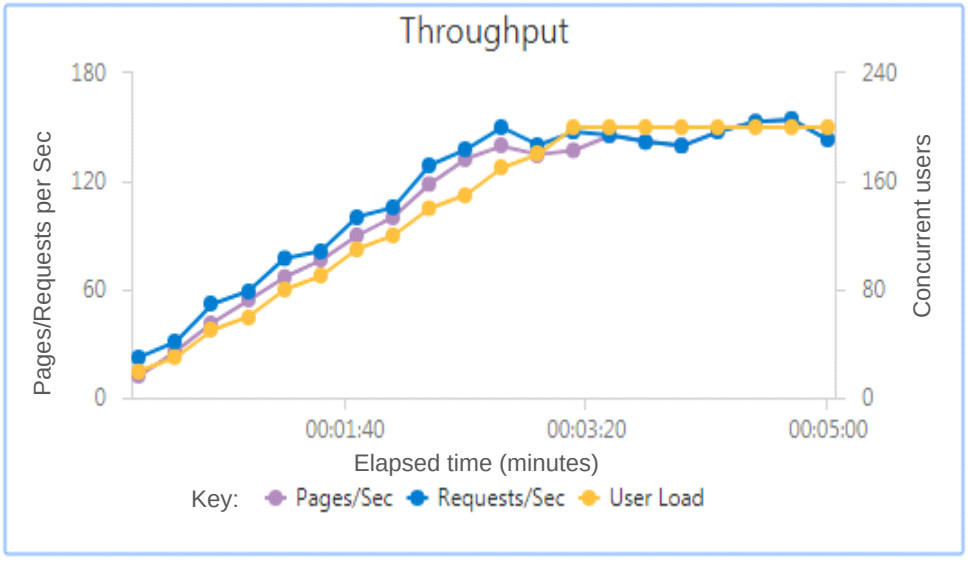

Figure 4.19: Graduated load test "Throughput" data for WebApp_5

Errors

From the errors graph below, was notable that rate of failed requests per second increased consistently with the increased in user load.

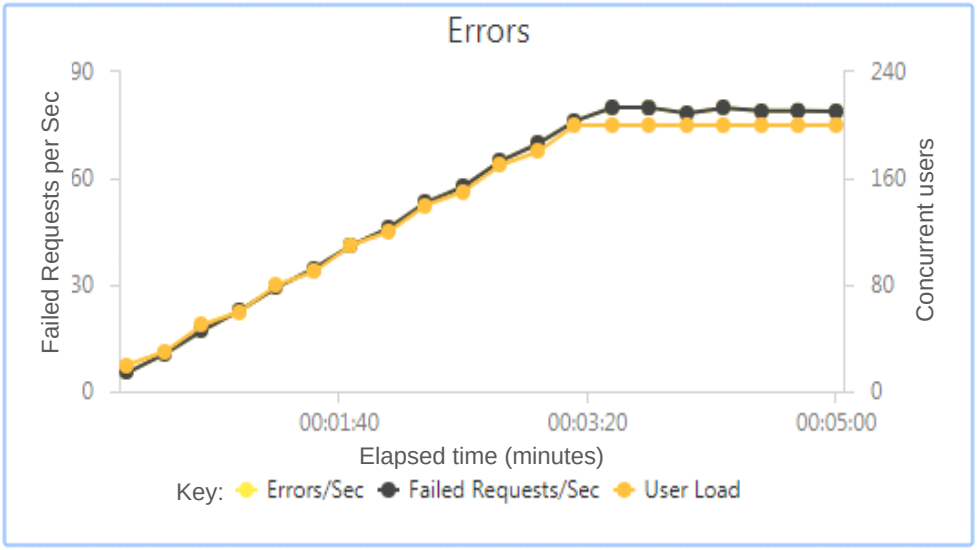

Figure 4.20: Graduated load test "Errors" data for WebApp_5

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained consistently low throughout the test period.

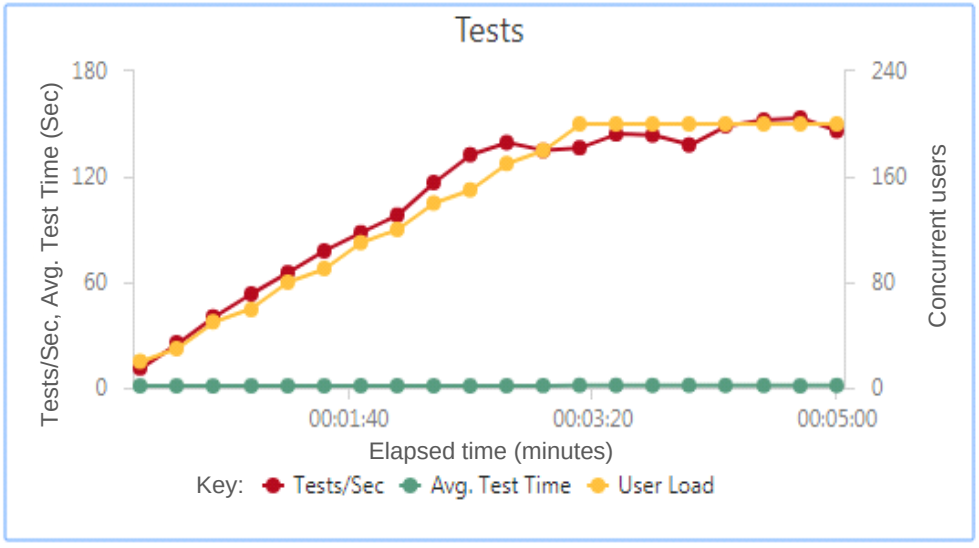

Figure 4.21: Graduated load test "Tests" data for WebApp_5

Discussion

The results showed that the test application, WebApp_5 (http://demo.nopcommerce.com), which is an end-to-end e-commerce application, has a good architectural designed and was able to provide a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load.

It was however worth noting that while page load errors were observed to increase consistently with the increase in user load, these did not seem to affect either the performance or the throughput of the application.

4.5.6 Test Application WebApp_6

This is a static website that allows for testing HTML website application modules that include dropdown, tabs, windows, date picker etc.

The table below shows the test results for the graduated load test for this application:

Table 4.8: Graduated load test results for WebApp_6

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	0.800000	6.916667	-	13.33333	2,861
0:30	30	4.000000	3.533333	3.200000	5.83333	2,867
0:45	50	0.933333	7.714286	0.466667	4.37500	2,868
1:00	60	1.466667	13.181820	-	8.95833	2,900
1:15	80	2.266667	30.205880	3.000000	7.18750	2,866
1:30	90	5.066667	14.460530	6.466667	5.93750	2,867
1:45	110	7.666667	11.182610	7.466667	10.62500	2,832
2:00	120	5.933333	10.741570	5.666667	13.95833	2,792
2:15	140	7.133333	17.009350	7.533333	13.12500	2,793
2:30	150	8.266666	12.782260	9.066667	4.37500	2,792
2:45	170	6.333333	20.652630	6.200000	2.29167	2,819
3:00	180	6.733333	23.445550	6.533333	4.58333	2,816
3:15	200	7.733333	18.025860	7.466667	4.79167	2,801
3:30	200	6.400000	32.312500	6.800000	0.62500	2,800
3:45	200	5.333333	33.325000	5.000000	1.56250	2,799
4:00	200	5.600000	34.488090	5.000000	1.04167	2,798
4:15	200	5.666667	37.835290	4.933333	1.35417	2,801
4:30	200	4.733333	38.309860	4.400000	2.39583	2,813
4:45	200	5.000000	34.973330	4.400000	0.93750	2,813
5:00	200	6.533333	37.142860	5.600000	1.66667	2,811

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that the average response time and the average page load time increased in a pattern consistent with the increase in user load. However, the values for the two metrics remained relatively low as depicted by the y-axis scale of between 0 and 0.36 seconds.

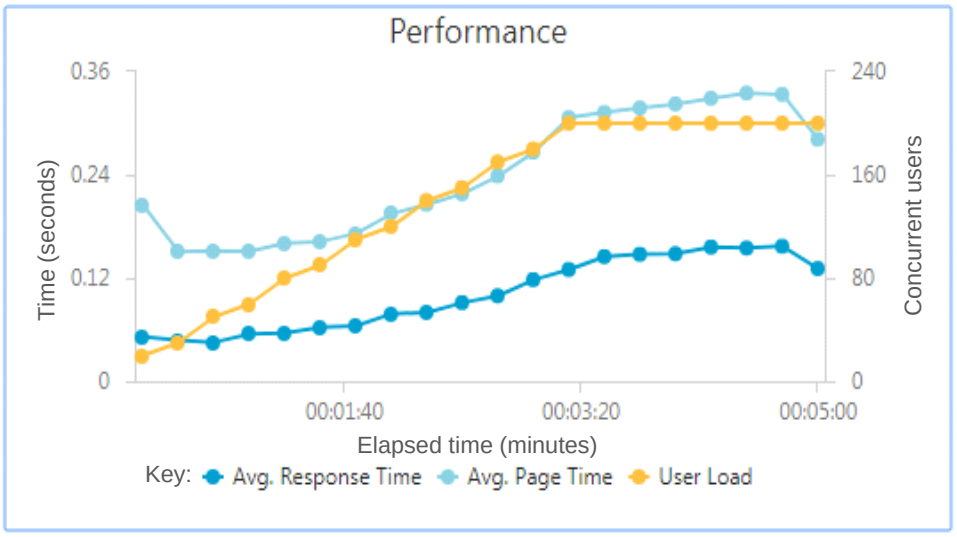

Figure 4.22: Graduated load test "Performance" data for WebApp_6

Throughput

From the throughput results graphed below, it was notable that the number of requests and pages per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

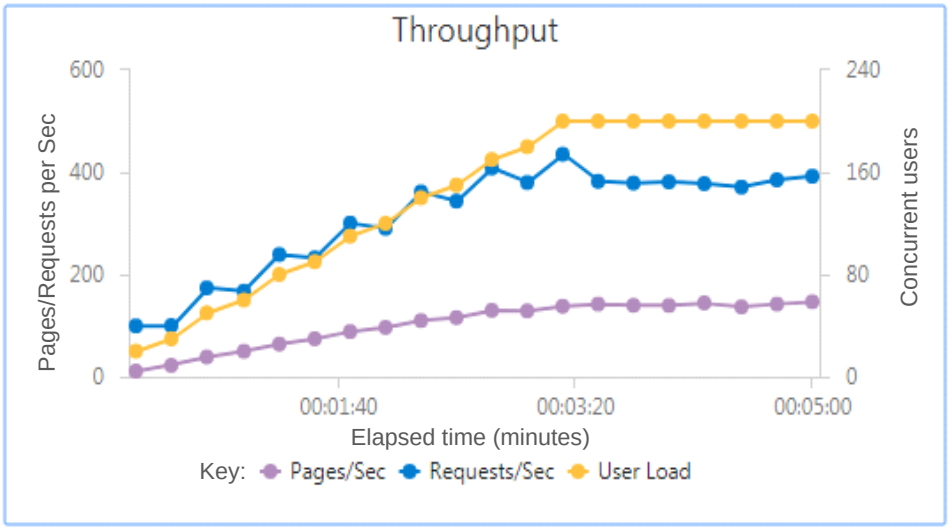

Figure 4.23: Graduated load test "Throughout" data for WebApp_6

Errors

From the errors graph below, was notable that rate of failed requests per second increased consistently with the increase in user load.

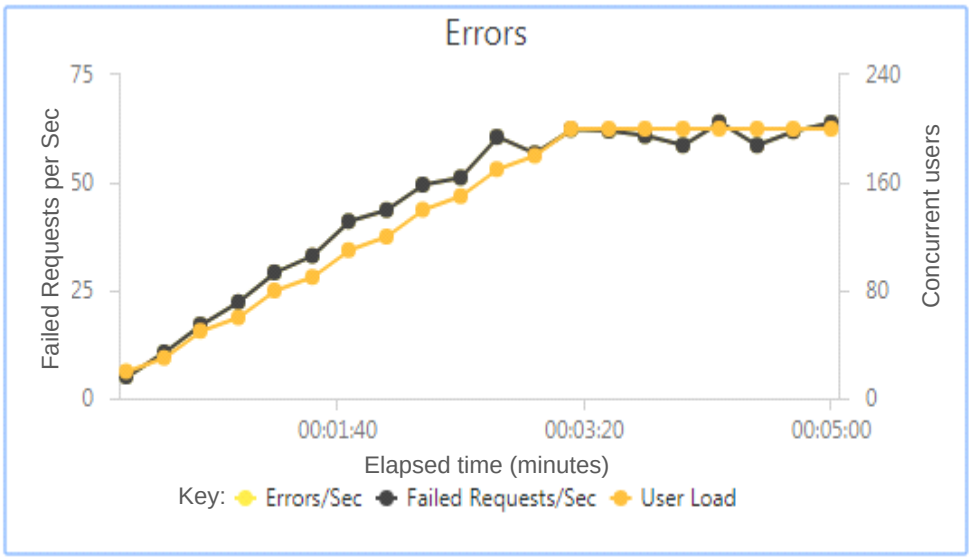

Figure 4.24: Graduated load test "Errors" data for WebApp_6

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. At the same time, the average test time remained consistently low throughout the test period.

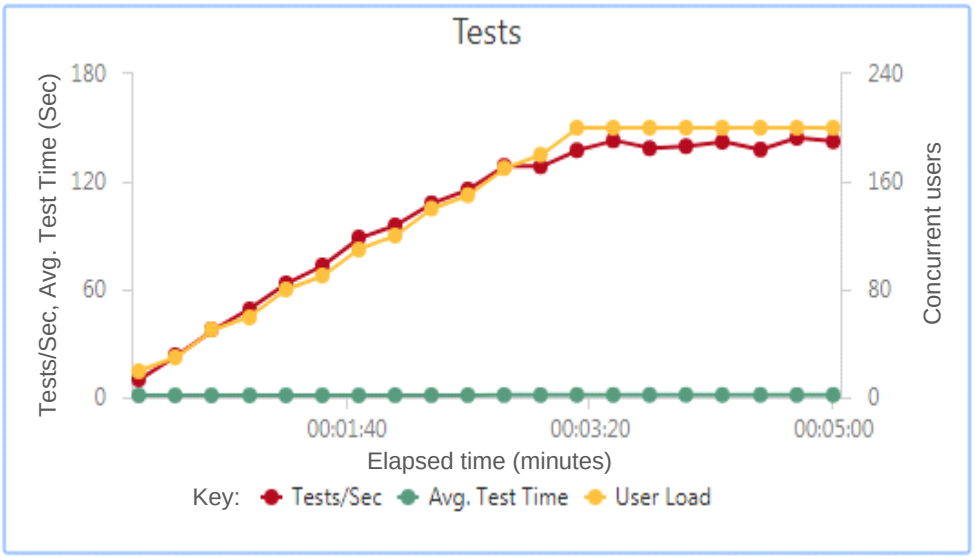

Figure 4.25: Graduated load test "Tests" data for WebApp_6

Discussion

The results showed that the test application, WebApp_6 (http://www.globalsqa.com), which is static HTML web site, has a good architectural designed and was able to provide a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load.

It was however worth noting that while page load errors were observed to increase consistently with the increase in user load, these did not seem to affect either the performance or the throughput of the application.

4.5.7 Test Application WebApp_7

This is a basic e-commerce site for testing e-commerce applications with interactions between client and server.

The table below shows the test results for the graduated load test for this application:

Table 4.9: Graduated load test results for WebApp_7

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	0.400000	9.500000	2.200000	13.750000	2,906
0:30	30	2.733333	3.243902	3.200000	3.125000	2,911
0:45	50	8.133333	2.647541	13.333330	8.229167	2,943
1:00	60	3.800000	2.894737	6.466667	9.583333	2,888
1:15	80	1.133333	13.529410	6.333333	6.666667	2,880
1:30	90	2.666667	31.400000	7.066667	12.083330	2,836
1:45	110	4.266667	23.500000	7.333333	17.812500	2,809
2:00	120	4.933333	15.513510	10.266670	9.791667	2,803
2:15	140	4.933333	24.635140	7.733333	2.291667	2,815
2:30	150	3.933333	29.745760	7.600000	1.145833	2,822
2:45	170	3.600000	34.333330	6.000000	4.791667	2,815
3:00	180	6.200000	30.107530	9.266666	5.000000	2,801
3:15	200	7.000000	23.866670	10.466670	1.041667	2,796
3:30	200	5.466667	30.268290	7.666667	1.458333	2,790
3:45	200	5.266667	35.506330	7.733333	1.250000	2,790
4:00	200	6.333333	34.484210	7.266667	0.833333	2,787
4:15	200	4.133333	40.967740	7.333333	0.729167	2,786
4:30	200	7.066667	29.537730	8.600000	1.354167	2,785
4:45	200	5.600000	36.988090	7.333333	1.666667	2,779
5:00	200	3.533333	42.264150	5.000000	0.833333	2,775

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased in an irregular pattern as the user load increased. At the same time, the values for the two metrics remained relatively high throughout the test period, as depicted by the y-axis scale of between 0 and 45 seconds.

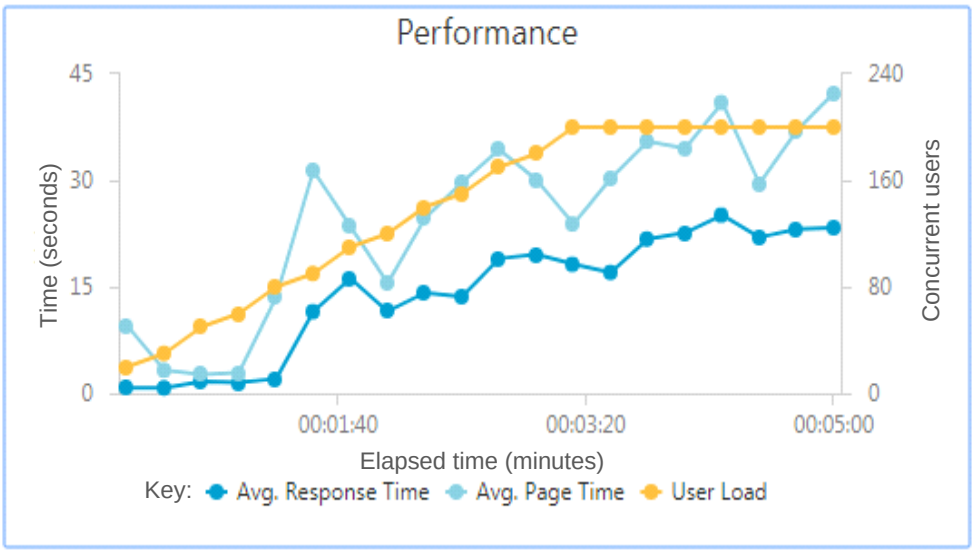

Figure 4.26: Graduated load test "Performance" data for WebApp_7

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second reduced consistently with the increase in the number of concurrent users. It can therefore be argued that throughput decreased with increase in user load.

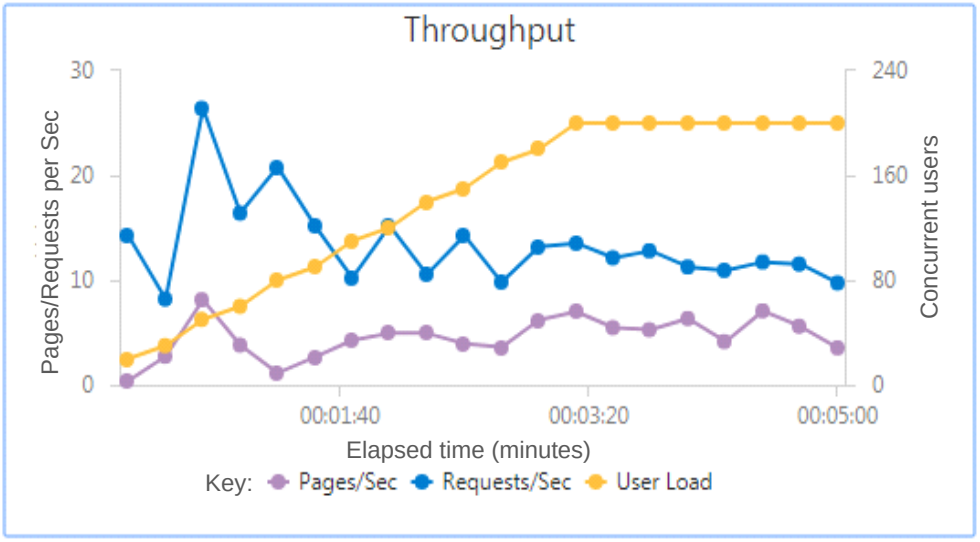

Figure 4.27: Graduated load test "Throughput" data for WebApp_7

Errors

From the errors graph below, was notable that rate of failed requests per second increased with the increased in user load and remained high throughout the test period.

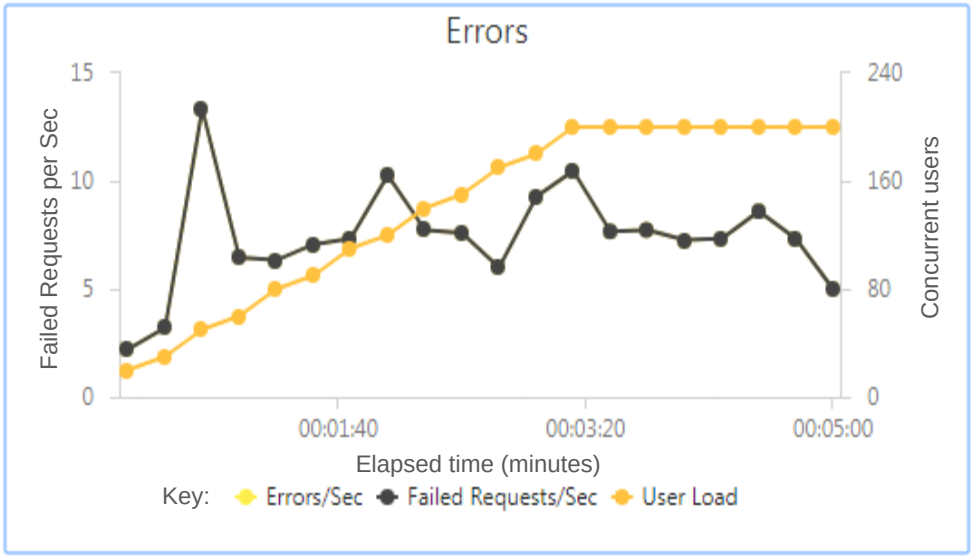

Figure 4.28: Graduated load test "Errors" data for WebApp_7

Tests

From the tests graph below, it was notable that the number of tests processed per second remained low throughout the test period in spite of the increase in user load. On the other hand, the average test time increased consistently with increase in user load.

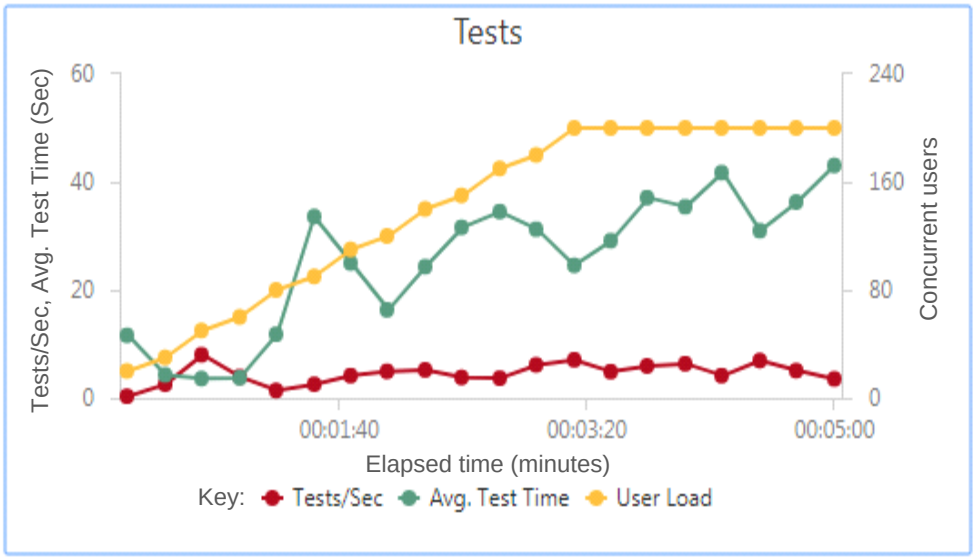

Figure 4.29: Graduated load test "Tests" data for WebApp_7

Discussion

The results show that the test application, WebApp_7 (http://store.demoqa.com), which is an e-commerce application, has a constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

With increasing user load, the number of failed requests increased as well as the average page load time, lending the application to provide an inconsistent user experience with changes in user load.

Compared to applications examined in previous tests, this application showed that throughput decreased with increase in user load as opposed to throughput increasing with increasing user load.

4.5.8 Test Application WebApp_8

This is a basic e-commerce site for testing e-commerce applications with interactions between client and server.

The table below shows the test results for the graduated load test for this application:

Table 4.10: Graduated load test results for WebApp_8

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	7.733333	1.146552	-	23.854170	2,866
0:30	30	14.800000	1.117117	-	20.000000	2,869
0:45	50	21.400000	1.411215	-	36.875000	2,890
1:00	60	27.200000	1.568627	2.733333	35.729170	2,838
1:15	80	34.733330	1.685221	4.533333	45.000000	2,804
1:30	90	34.333330	1.817476	54.733330	46.979170	2,760
1:45	110	17.733330	5.142857	93.466670	39.479170	2,732
2:00	120	21.800000	5.305810	55.400000	35.104170	2,655
2:15	140	28.066670	4.375297	64.600000	36.145830	2,672
2:30	150	22.600000	5.719764	18.400000	33.333330	2,626
2:45	170	36.333330	4.354128	22.733330	41.041670	2,588
3:00	180	23.866670	6.365922	22.266670	36.145830	2,576
3:15	200	34.600000	5.741811	6.533333	43.020830	2,591
3:30	200	19.466670	7.660959	35.066670	29.375000	2,583
3:45	200	20.000000	8.743333	58.333330	28.437500	2,560
4:00	200	42.733330	6.238689	40.266670	30.625000	2,559
4:15	200	23.333330	5.082857	47.200000	34.583330	2,585
4:30	200	26.400000	8.000000	27.266670	29.895830	2,580
4:45	200	38.600000	4.680484	22.600000	38.229170	2,581
5:00	200	54.066670	3.204686	114.600000	33.854170	2,583

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time increased marginally with the increase in user load. On the other hand, the average page load time increased steadily as the user load increased. The values for the two metrics remained high throughout the test period, as depicted by the y-axis scale of between 0 and 12 seconds.

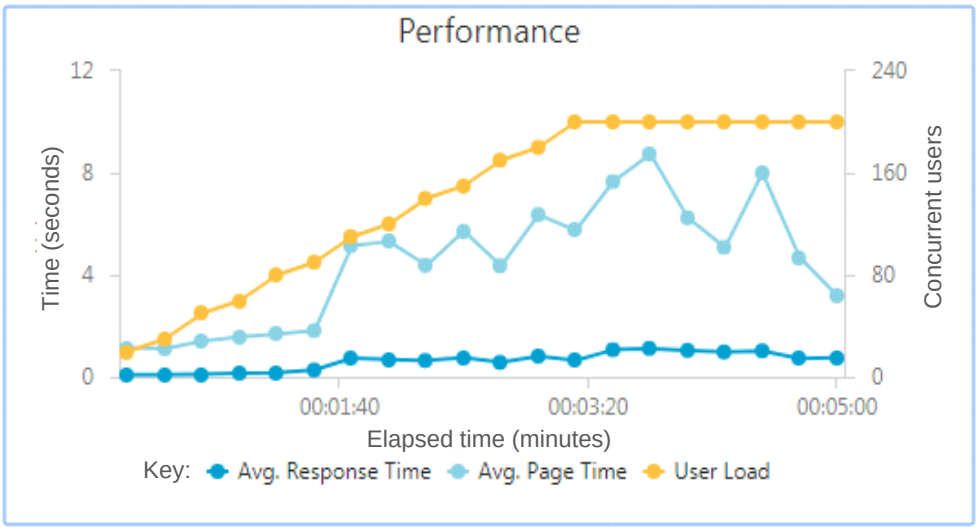

Figure 4.30: Graduated load test "Performance" data for WebApp_8

Throughput

From the throughput results graphed below, it was notable that the number of requests processed per second was not consistent with the increase in user load. The number of pages that were successfully processed per second increased marginally throughout the test period.

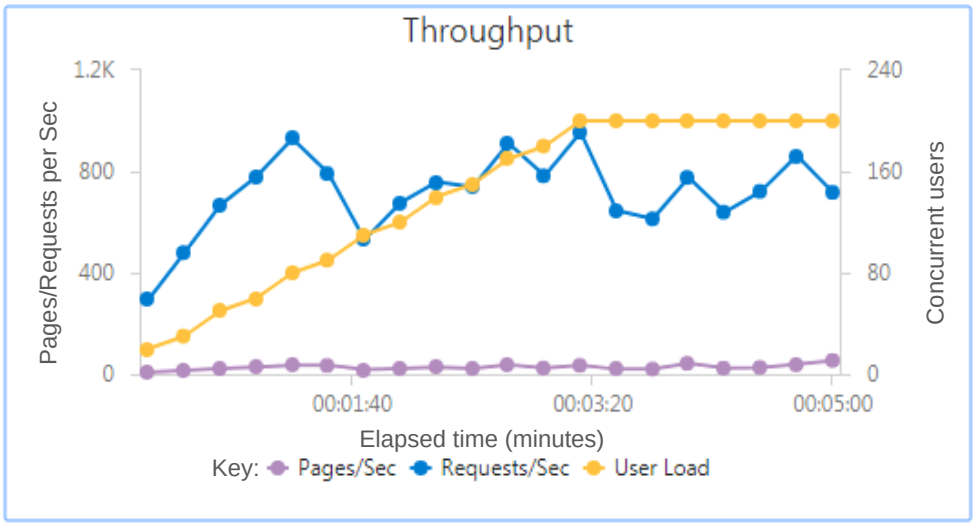

Figure 4.31: Graduated load test "Throughput" data for WebApp_8

Errors

From the errors graph below, an irregular of rate of failed requests per second was recorded throughout the test period.

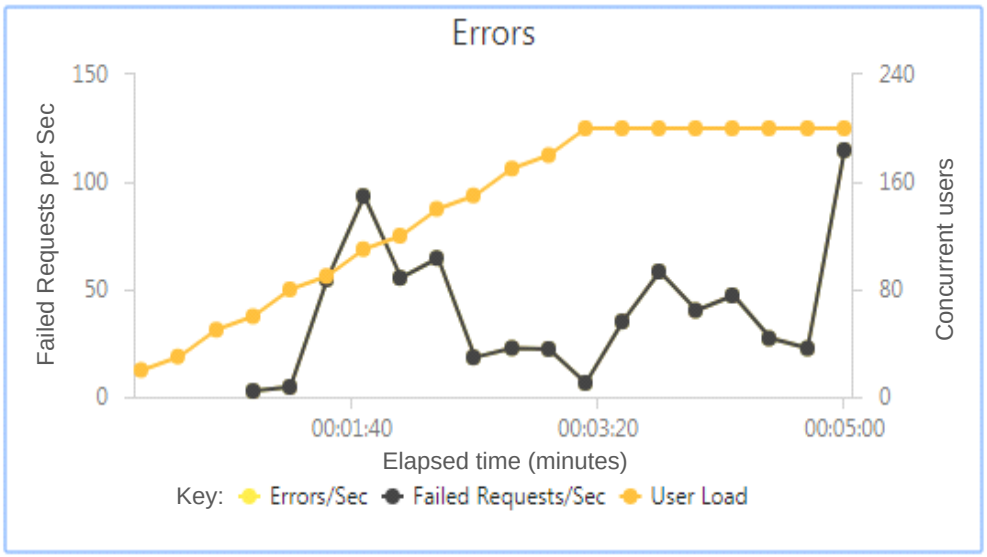

Figure 4.32: Graduated load test "Errors" data for WebApp_8

Tests

From the tests graph below, it was notable that the number of tests processed per second was irregular and not consistent with the number of concurrent users. On the other hand, the average test time increased marginally throughout the test period.

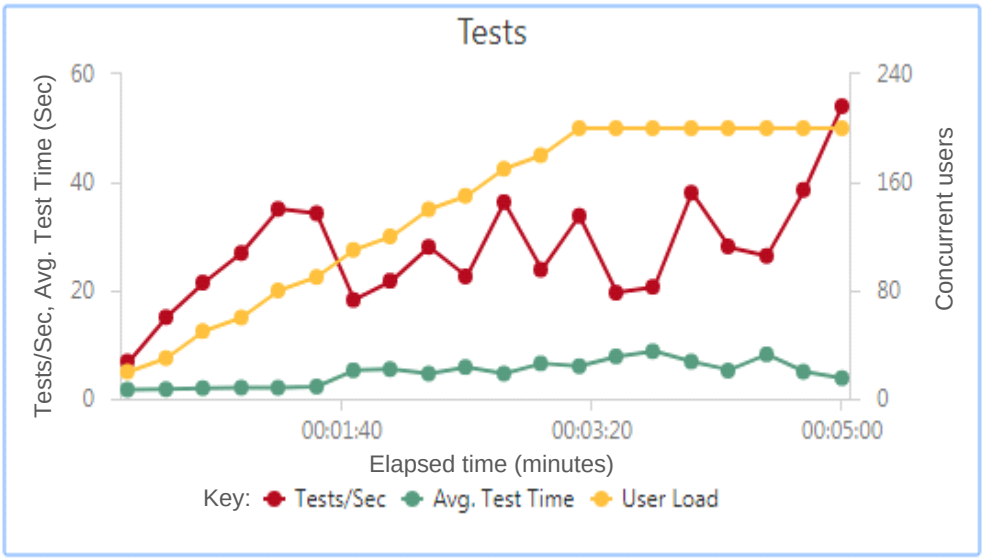

Figure 4.33: Graduated load test "Tests" data for WebApp_8

Discussion

The results show that the test application, WebApp_8 (http://awful-valentine.com), which is an e-commerce application with client-server interactions, has a constrained design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

Compared to other applications examined previously, this application exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load.

In this case, the association between throughput and user load was not clearly visualised.

4.5.9 Test Application WebApp_9

This is a sample insurance company web application.

The table below shows the test results for the graduated load test for this application:

Table 4.11: Graduated test results for WebApp_9

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	10.666670	0.662500	-	14.583330	2,882
0:30	30	22.666670	0.511765	-	9.895833	2,886
0:45	50	37.333330	0.526786	-	12.291670	2,884
1:00	60	48.000000	0.491667	-	20.312500	2,897
1:15	80	63.400000	0.502629	-	20.625000	2,868
1:30	90	73.000000	0.488585	-	32.187500	2,819
1:45	110	86.933330	0.519172	-	37.395830	2,797
2:00	120	99.000000	0.519865	-	36.354170	2,769
2:15	140	111.400000	0.539797	-	32.812500	2,762
2:30	150	125.200000	0.507455	-	34.375000	2,782
2:45	170	137.600000	0.520833	-	36.354170	2,751
3:00	180	141.133300	0.635805	-	39.166670	2,741
3:15	200	161.866700	0.556425	-	44.687500	2,735
3:30	200	167.533300	0.538400	-	43.854170	2,732
3:45	200	168.733300	0.512446	-	40.937500	2,728
4:00	200	170.333300	0.504501	-	41.041670	2,729
4:15	200	165.733300	0.549477	-	42.083330	2,742
4:30	200	170.466700	0.524052	-	41.354170	2,744
4:45	200	167.800000	0.522447	-	41.145830	2,743
5:00	200	168.200000	0.522394	-	40.833330	2,747

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained steady throughout the test period in spite of the increase in user load. The values for the two metrics remained relatively low throughout the test period, as depicted by the y-axis scale of between 0 and 0.75 seconds.

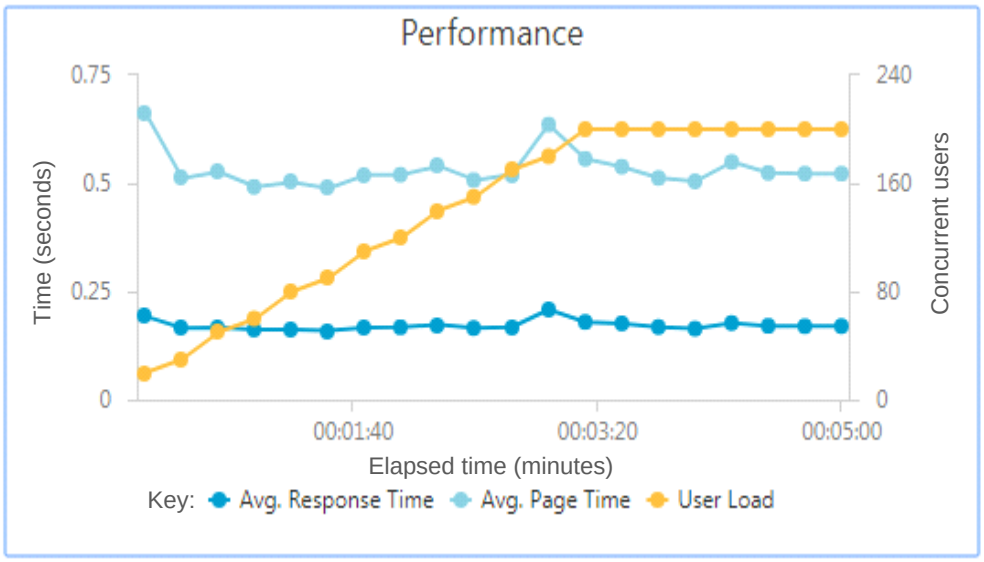

Figure 4.34: Graduated load test "Performance" data for WebApp_9

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

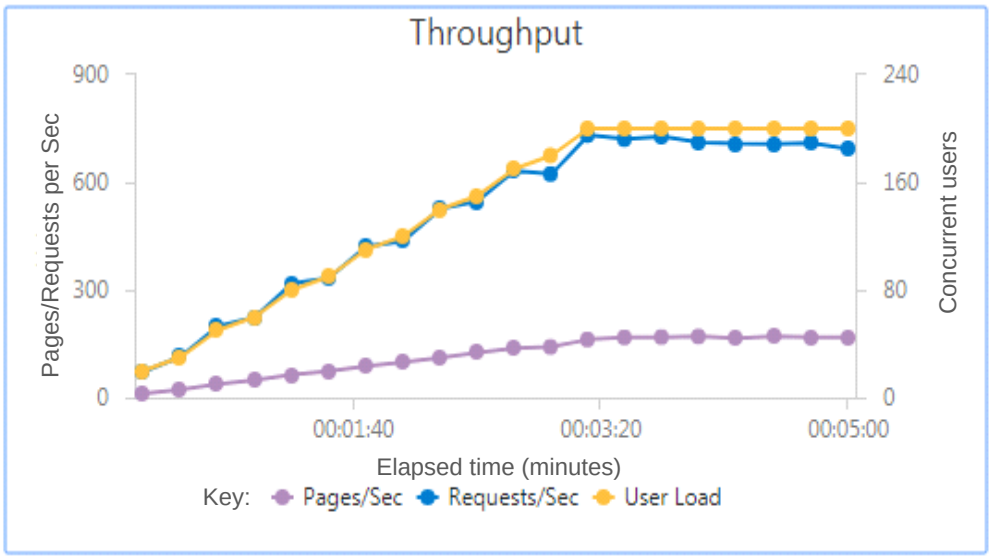

Figure 4.35: Graduated load test "Throughput" data for WebApp_9

Errors

From the errors graph below, was notable that no failed requests were recorded throughout the test period.

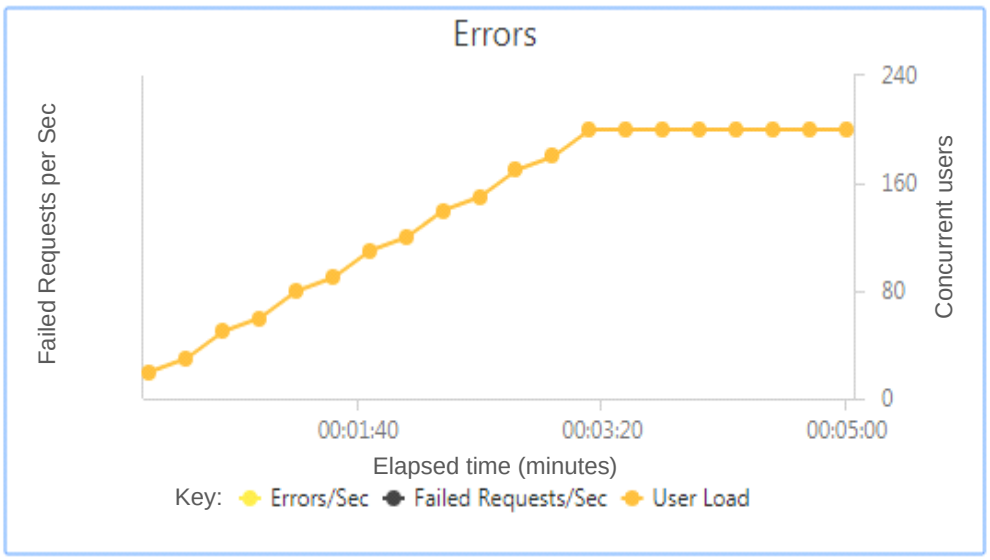

Figure 4.36: Graduated load test "Errors" data for WebApp_9

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained steady throughout the test period.

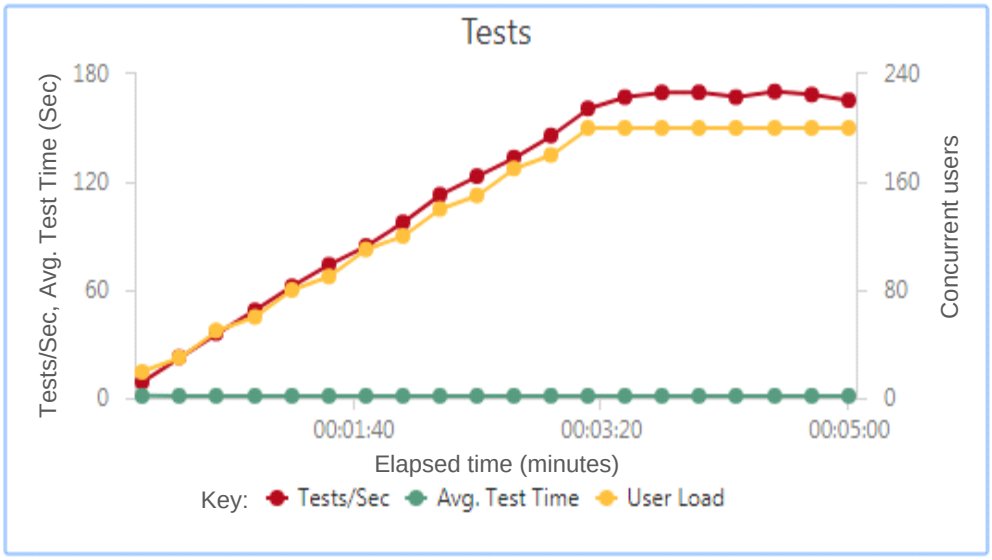

Figure 4.37: Graduated load test "Tests" data for WebApp_9

Discussion

The results show that the test application, WebApp_9 (http://demo.borland.com), which is an insurance web application, has a good architectural design which provided a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load and no page load errors were recorded during the test.

4.5.10 Test Application WebApp_10

This is a sample Tours & Travel booking web application.

The table below shows the test results for the graduated load test for this application:

Table 4.12: Graduated load test results for WebApp_10

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	12.666670	0.284211	-	21.666670	2,872
0:30	30	18.666670	0.228571	-	9.062500	2,877
0:45	50	18.666670	0.235714	-	15.625000	2,880
1:00	60	18.333330	0.225455	-	12.604170	2,876
1:15	80	19.066670	3.209790	1.333333	14.062500	2,852
1:30	90	18.733330	1.782918	0.733333	18.229170	2,817
1:45	110	19.000000	3.312281	1.666667	17.604170	2,805
2:00	120	8.466666	9.763780	1.933333	7.812500	2,807
2:15	140	2.266667	40.176470	2.266667	1.666667	2,829
2:30	150	3.066667	42.043480	3.266667	1.666667	2,803
2:45	170	3.333333	42.040000	3.333333	5.729167	2,812
3:00	180	3.666667	42.036370	3.800000	0.312500	2,812
3:15	200	4.333333	38.800000	4.333333	0.208333	2,811
3:30	200	4.666667	42.042860	4.666667	0.312500	2,811
3:45	200	3.666667	42.054550	3.733333	0.520833	2,811
4:00	200	5.666667	42.035290	5.666667	0.520833	2,811
4:15	200	4.800000	42.041670	4.800000	0.729167	2,816
4:30	200	4.000000	37.516670	3.600000	0.312500	2,822
4:45	200	6.666667	36.180000	5.733333	0.729167	2,823
5:00	200	5.200000	34.269230	4.266667	0.208333	2,827

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased significantly when the user load exceeds 100 users and remained high for the rest of the test period.

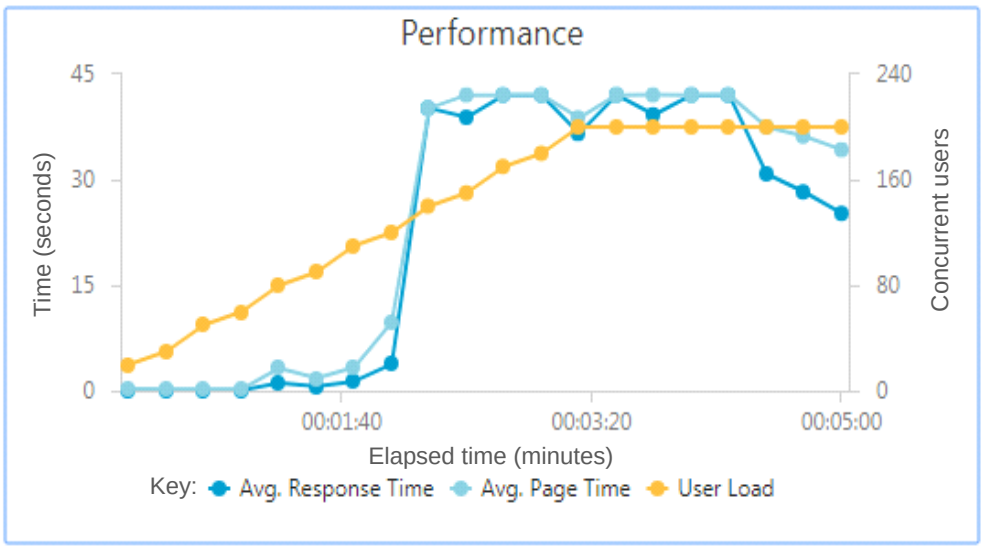

Figure 4.38: Graduated load test "Performance" data for WebApp_10

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second dropped to near zero when the user load exceeded 100 users and remained low for the rest of the test period.

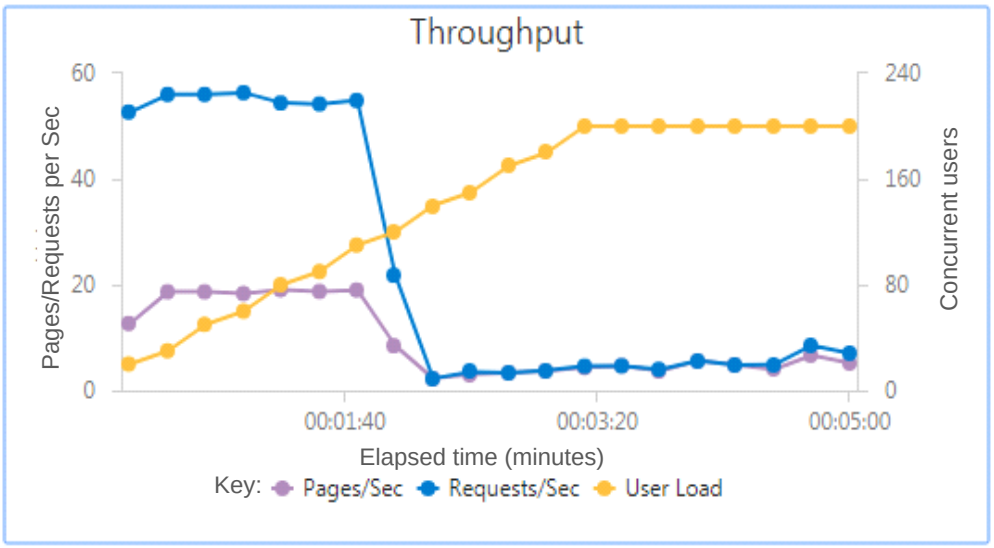

Figure 4.39: Graduated load test "Throughput" data for WebApp_10

Errors

From the errors graph below, it was notable that rate of failed requests per second increased consistently with the increased in user load.

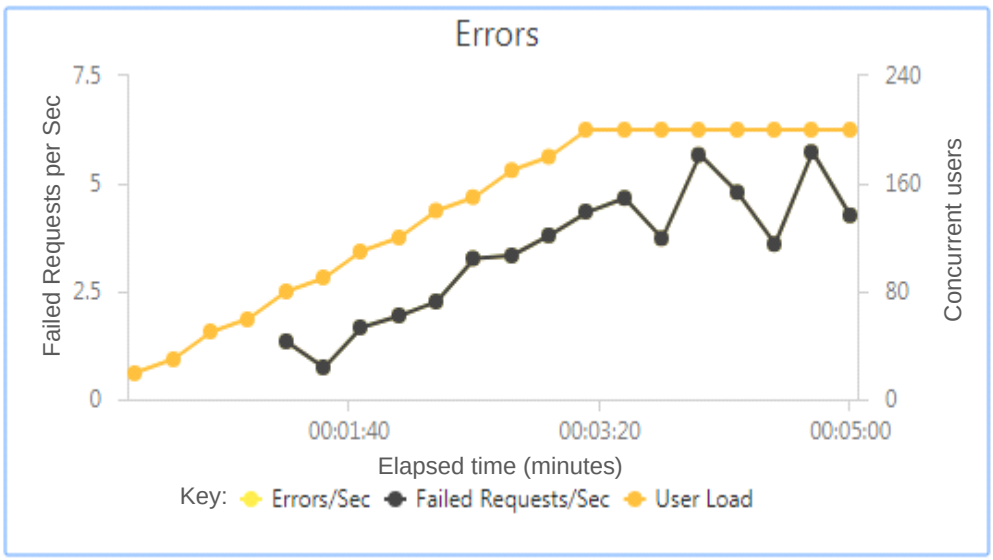

Figure 4.40: Graduated load test "Errors" data for WebApp_10

Tests

From the tests graph below, it was notable that the number of tests processed per second was dropped to near zero when the user load exceeded 100 users and remained low for the rest of the test period. On the other hand, the average test time increased significantly and remained high after the user load exceeded 100 users.

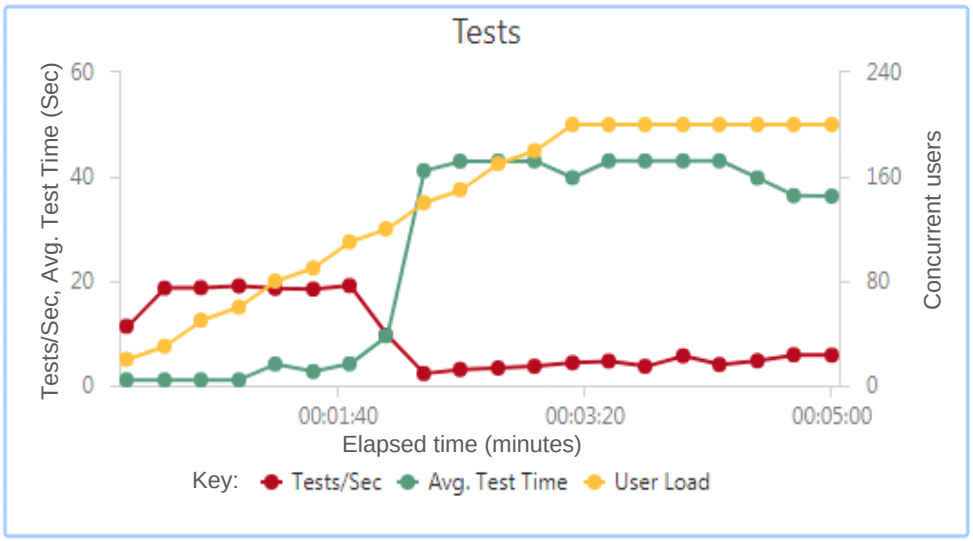

Figure 4.41: Graduated load test "Tests" data for WebApp_10

Discussion

The results show that the test application, WebApp_10 (http://phptravels.com), which is Tours and Travel booking application with client-server interactions, has constrained architectural design that the application stops responding to user requests when the user load exceeds relatively low number of concurrent users about 100.

Compared to other applications examined previously, this application exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load. In this case, the association between throughput and user load was not clearly visualised.

The application may be considered to have crashed at this user point of user load.

4.5.11 Test Application WebApp_11

A generic website with rich set of web UI functions specially designed to the needs of testing of web application of all types.

The table below shows the test results for the graduated load test for this application:

Table 4.13: Graduated load test results for WebApp_11

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	0.800000	6.916667	-	13.33333	2,861
0:30	30	4.000000	3.533333	3.200000	5.83333	2,867
0:45	50	0.933333	7.714286	0.466667	4.37500	2,868
1:00	60	1.466667	13.181820	-	8.95833	2,900
1:15	80	2.266667	30.205880	3.000000	7.18750	2,866
1:30	90	5.066667	14.460530	6.466667	5.93750	2,867
1:45	110	7.666667	11.182610	7.466667	10.62500	2,832
2:00	120	5.933333	10.741570	5.666667	13.95833	2,792
2:15	140	7.133333	17.009350	7.533333	13.12500	2,793
2:30	150	8.266666	12.782260	9.066667	4.37500	2,792
2:45	170	6.333333	20.652630	6.200000	2.29167	2,819
3:00	180	6.733333	23.445550	6.533333	4.58333	2,816
3:15	200	7.733333	18.025860	7.466667	4.79167	2,801
3:30	200	6.400000	32.312500	6.800000	0.62500	2,800
3:45	200	5.333333	33.325000	5.000000	1.56250	2,799
4:00	200	5.600000	34.488090	5.000000	1.04167	2,798
4:15	200	5.666667	37.835290	4.933333	1.35417	2,801
4:30	200	4.733333	38.309860	4.400000	2.39583	2,813
4:45	200	5.000000	34.973330	4.400000	0.93750	2,813
5:00	200	6.533333	37.142860	5.600000	1.66667	2,811

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased significantly after the user load reached 50 users and continued to increase in an irregular pattern for the rest of the test period.

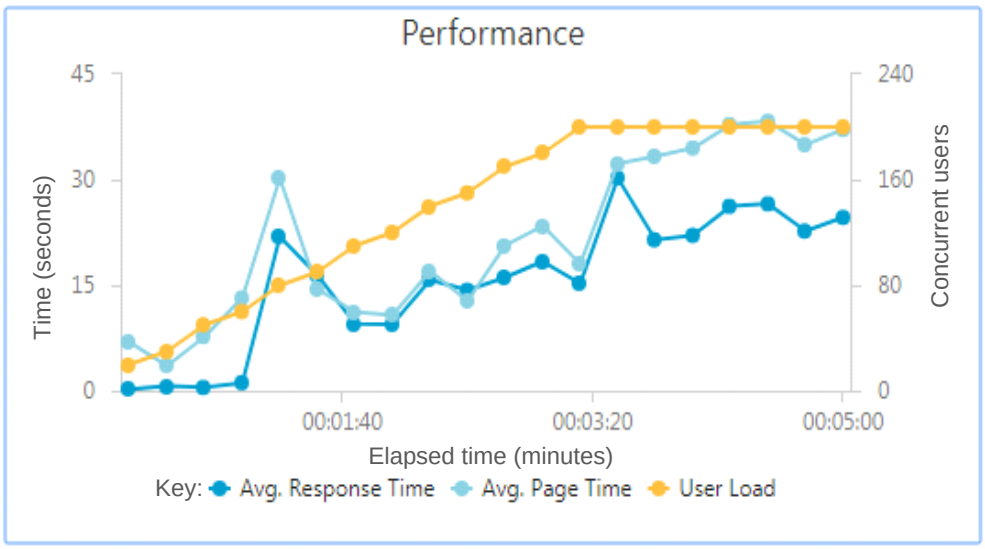

Figure 4.42: Graduated load test "Performance" data for WebApp_11

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second dropped to near zero when the user load exceeded 50 users and remained low for the rest of the test period.

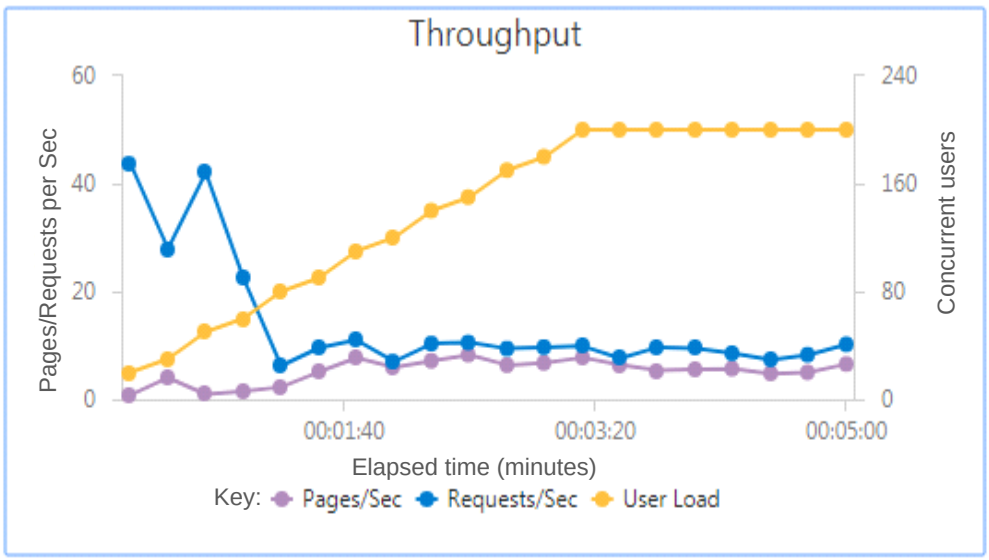

Figure 4.43: Graduated load test "Throughput" data for WebApp_11

Errors

From the errors graph below, it was notable that rate of failed requests per second increased in an irregular patter as the user load increased and remained high throughout the test period.

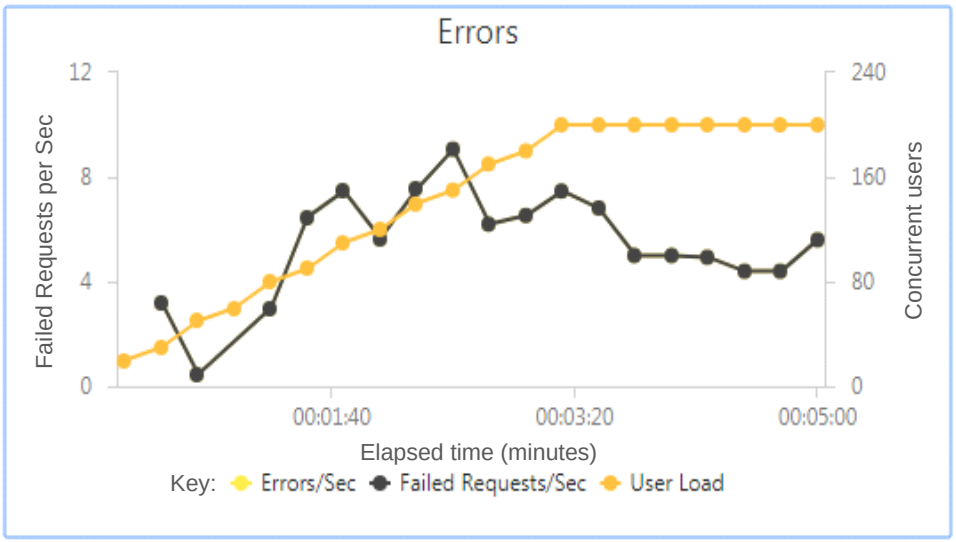

Figure 4.44: Graduated load test "Errors" data for WebApp_11

Tests

From the tests graph below, it was notable that the number of tests processed per second remained low throughout the test period. On the other hand, the average test time increased in an irregular pattern as the user load increased.

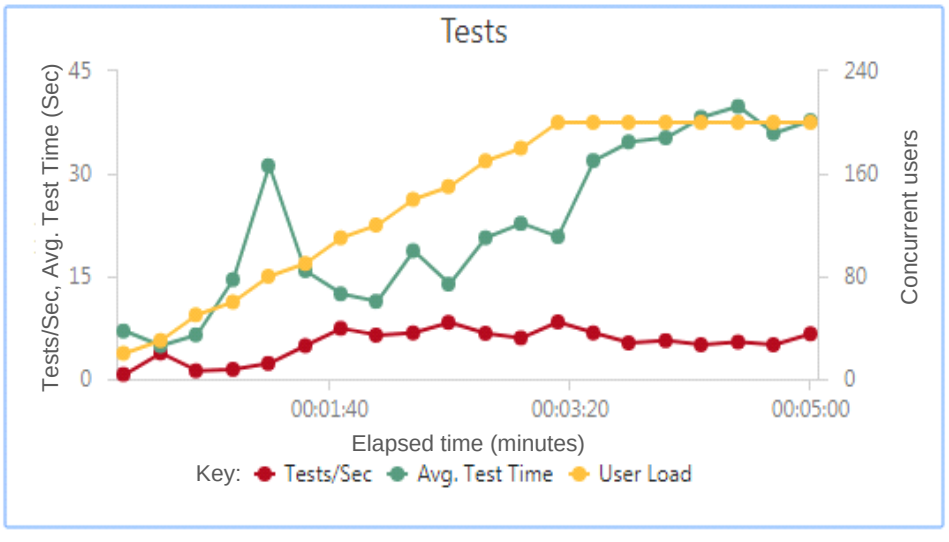

Figure 4.45: Graduated load test "Tests" data for WebApp_11

Discussion

The results show that the test application, WebApp_11 (http://demoqa.com), which is generic website with rich set of web UI functions, has constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

Compared to other applications examined previously, this application exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load.

In this case, the association between throughput and user load was not clearly visualised.

4.5.12 Test Application WebApp_12

This is a static website that allows for testing HTML website application modules that include dropdown, tabs, windows, date picker etc.

The table below shows the test results for the graduated load test for this application:

Table 4.14: Graduated test results for WebApp_12

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	8.733334	0.732824	9.733334	13.541670	2,890
0:30	30	18.400000	0.739130	22.000000	6.041667	2,890
0:45	50	27.800000	0.726619	30.266670	7.291667	2,926
1:00	60	36.266670	0.777574	40.333330	20.937500	2,911
1:15	80	47.800000	0.778243	55.466670	14.791670	2,892
1:30	90	56.733330	0.781434	68.066670	24.375000	2,836
1:45	110	67.666660	0.785222	82.666660	25.416670	2,790
2:00	120	77.133330	0.766638	90.466670	31.666670	2,748
2:15	140	86.800000	0.768049	103.066700	25.000000	2,743
2:30	150	96.933330	0.747593	112.733300	22.500000	2,750
2:45	170	107.533300	0.748915	124.333300	22.604170	2,778
3:00	180	115.666700	0.731412	131.466700	26.041670	2,775
3:15	200	125.600000	0.739915	141.533300	27.395830	2,735
3:30	200	130.933300	0.726069	143.266700	23.645830	2,752
3:45	200	131.200000	0.727134	142.866700	27.812500	2,740
4:00	200	134.333300	0.716129	145.933300	26.666670	2,745
4:15	200	132.333300	0.725441	147.400000	28.958330	2,724
4:30	200	134.533300	0.735877	150.200000	25.104170	2,766
4:45	200	133.466700	0.758741	151.133300	28.020830	2,684
5:00	200	136.933300	0.730769	153.400000	28.854170	2,756

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained steady throughout the test period in spite of the increase in user load. The values for the two metrics remained relatively low throughout the test period, as depicted by the y-axis scale of between 0 and 0.9 seconds.

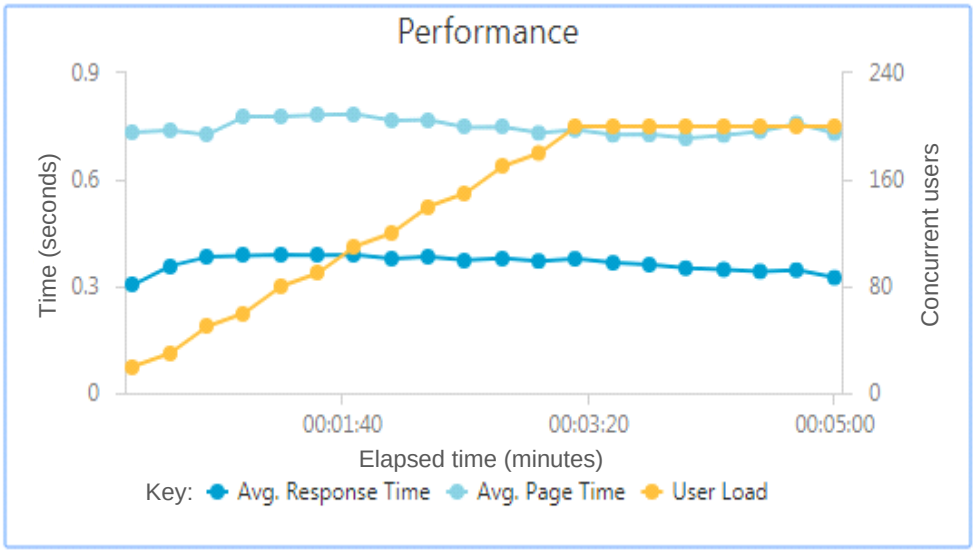

Figure 4.46: Graduated load test "Performance" data for WebApp_12

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

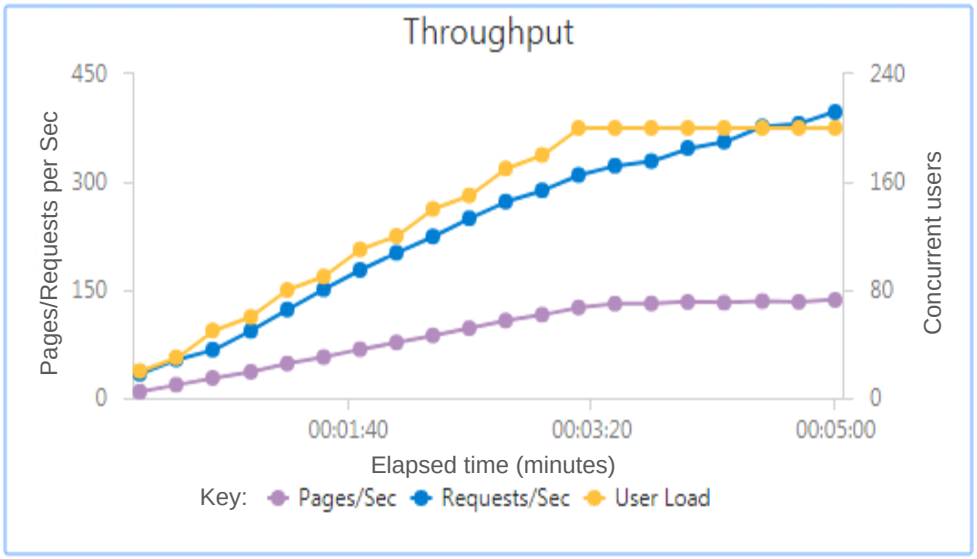

Figure 4.47: Graduated load test "Throughput" data for WebApp_12

Errors

From the errors graph below, was notable that rate of failed requests per second increased consistently with the increased in user load.

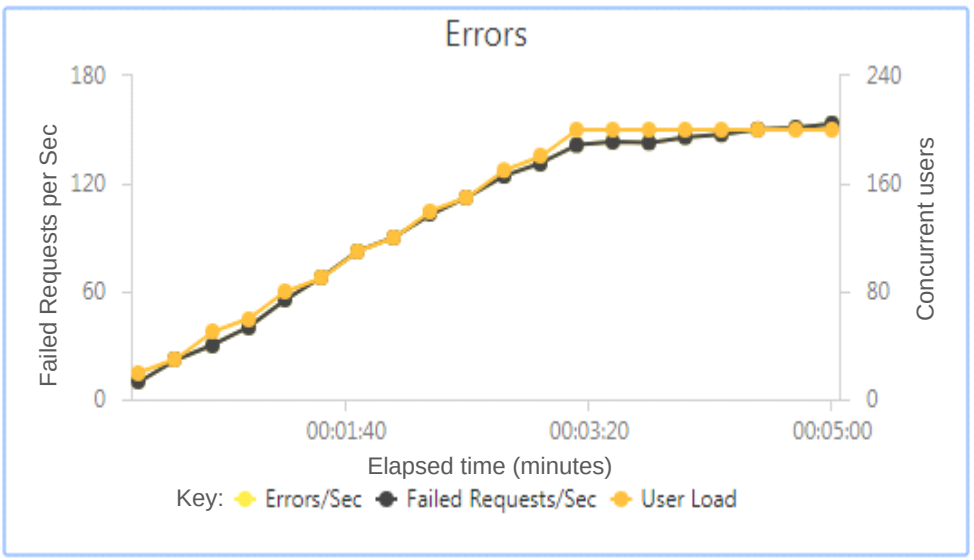

Figure 4.48: Graduated load test "Errors" data for WebApp_12

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained steady throughout the test period.

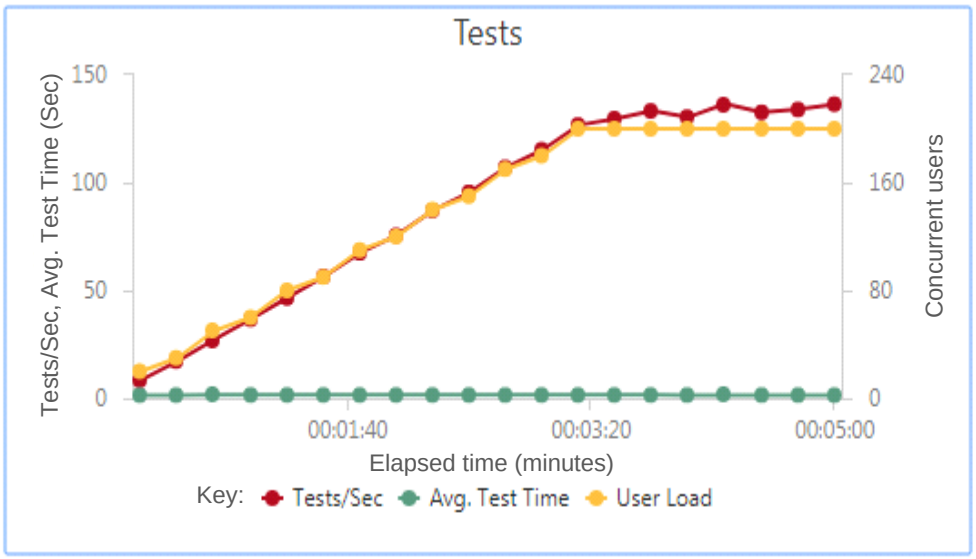

Figure 4.49: Graduated load test "Tests" data for WebApp_12

Discussion

The results show that the test application, WebApp_12 (http://thedemosite.co.uk), which is an online banking application, has a good architectural design that provided a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load.

It was however worth noting that while page load errors were observed to increase consistently with the increase in user load, these did not seem to affect either the performance or the throughput of the application.

4.5.13 Test Application WebApp_13

This is a generic web application for testing automated performance tests.

The table below shows the test results for the graduated load test for this application:

Table 4.15: Graduated test results for WebApp_13

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	4.866667	1.534247	-	14.687500	2,872
0:30	30	14.666670	0.981818	-	8.854167	2,864
0:45	50	23.000000	1.228986	0.133333	11.354170	2,894
1:00	60	34.066670	1.021526	0.066667	26.458330	2,859
1:15	80	41.600000	1.011218	0.133333	21.354170	2,824
1:30	90	51.466670	0.979275	0.133333	18.125000	2,818
1:45	110	59.066670	1.056433	0.266667	30.104170	2,778
2:00	120	72.733330	0.974336	0.200000	36.458330	2,706
2:15	140	79.466670	0.958893	0.200000	33.333330	2,697
2:30	150	90.866670	0.866471	0.066667	29.062500	2,695
2:45	170	101.333300	1.012500	0.466667	31.875000	2,714
3:00	180	111.333300	0.887425	0.133333	36.145830	2,696
3:15	200	119.000000	0.864426	0.066667	40.729170	2,682
3:30	200	126.666700	0.886842	0.200000	38.125000	2,670
3:45	200	130.733300	0.893422	0.266667	37.291670	2,678
4:00	200	128.733300	0.828586	0.066667	35.520830	2,707
4:15	200	130.466700	0.827798	0.066667	33.958330	2,716
4:30	200	132.466700	0.818319	0.066667	37.083330	2,725
4:45	200	129.733300	0.801644	-	36.875000	2,724
5:00	200	131.600000	0.814590	-	40.937500	2,724

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained steady throughout the test period in spite of the increase in user load. The values for the two metrics remained relatively low throughout the test period, as depicted by the y-axis scale of between 0 and 1.8 seconds.

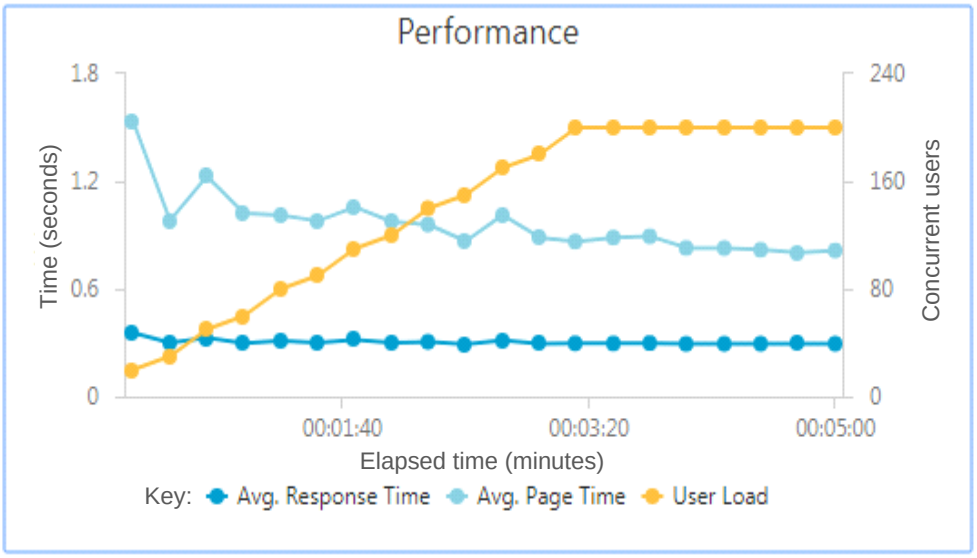

Figure 4.50: Graduated load test "Performance" data for WebApp_13

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second increased consistently with the increase in the number of concurrent users. The number of pages loaded per second increased marginally throughout the test period.

It can therefore be argued that throughput increased with increase in user load.

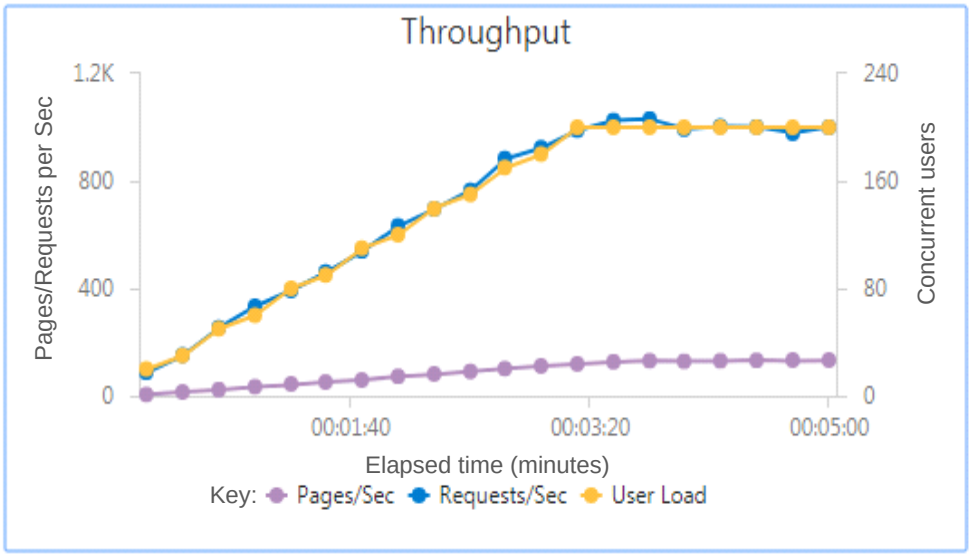

Figure 4.51: Graduated load test "Throughput" data for WebApp_13

Errors

From the errors graph below, an irregular rate of failed requests per second was recorded at certain points during the test period.

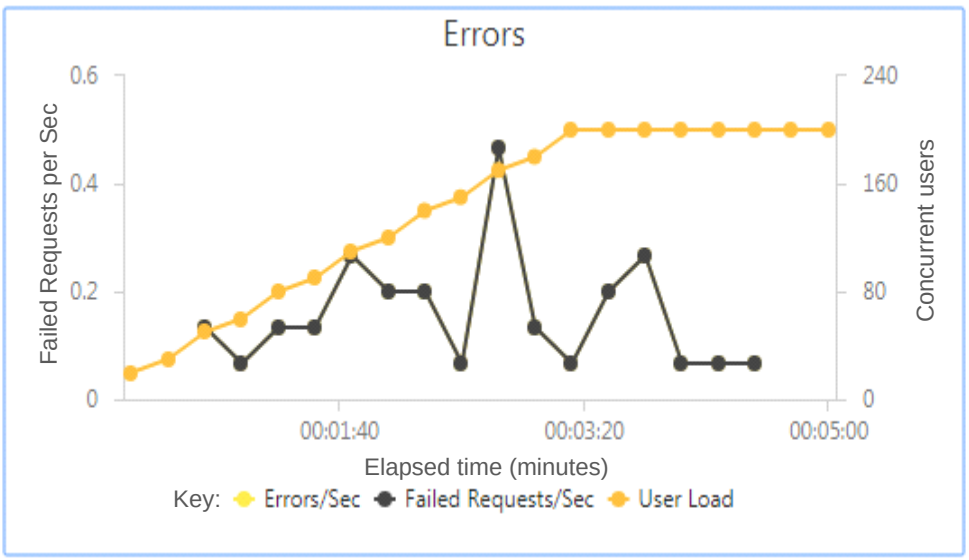

Figure 4.52: Graduated load test "Errors" data for WebApp_13

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained steady throughout the test period.

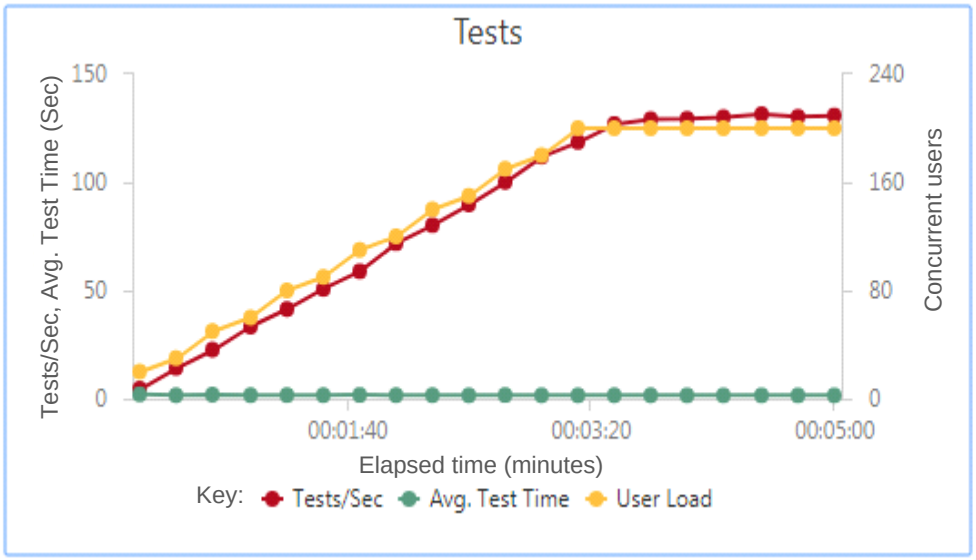

Figure 4.53: Graduated load test "Tests" data for WebApp_13

Discussion

The results show that the test application, WebApp_13 (http://www.way2automation.com), which is a generic web application for conducting automated performance tests, has a good architectural design that provided a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load.

It was however worth noting that while page load errors were observed a certain points during the test, these did not seem to affect either the performance or the throughput of the application.

4.5.14 Test Application WebApp_14

A generic website with rich set of web user interface (UI) functions specially designed to the needs of testing of web application of all types.

The table below shows the test results for the graduated load test for this application:

Table 4.16: Graduated load test results for WebApp_14

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	8.000000	0.766667	7.333333	17.187500	2,846
0:30	30	25.333330	0.071053	25.333330	3.020833	2,862
0:45	50	40.666670	0.075410	40.666670	3.020833	2,889
1:00	60	7.600000	0.561404	6.733333	14.479170	2,818
1:15	80	0.133333	25.000000	0.133333	4.791667	2,813
1:30	90	11.800000	6.050848	11.800000	9.479167	2,785
1:45	110	31.666670	1.957895	31.666670	14.166670	2,752
2:00	120	38.400000	2.059028	38.133340	11.770830	2,752
2:15	140	116.266700	1.264335	114.400000	19.270830	2,698
2:30	150	136.533300	0.077148	136.533300	11.770830	2,688
2:45	170	149.733300	0.077026	149.733300	16.145830	2,668
3:00	180	5.333333	0.075000	5.333333	3.645833	2,658
3:15	200	8.466666	18.677170	8.466666	0.416667	2,660
3:30	200	42.466670	4.411303	42.466670	1.250000	2,661
3:45	200	64.600000	1.728586	64.600000	3.020833	2,661
4:00	200	52.733330	2.102402	52.733330	2.187500	2,661
4:15	200	120.600000	2.571034	120.600000	4.062500	2,667
4:30	200	184.666700	0.069675	184.666700	6.562500	2,668
4:45	200	187.866700	0.059262	187.866700	5.729167	2,673
5:00	200	105.066700	0.372462	104.533300	7.291667	2,671

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased in an irregular pattern as the user load increased. However, the values for the two metrics varied significantly throughout the test period, oscillating between the values on the y-axis between 0 and 30 seconds.

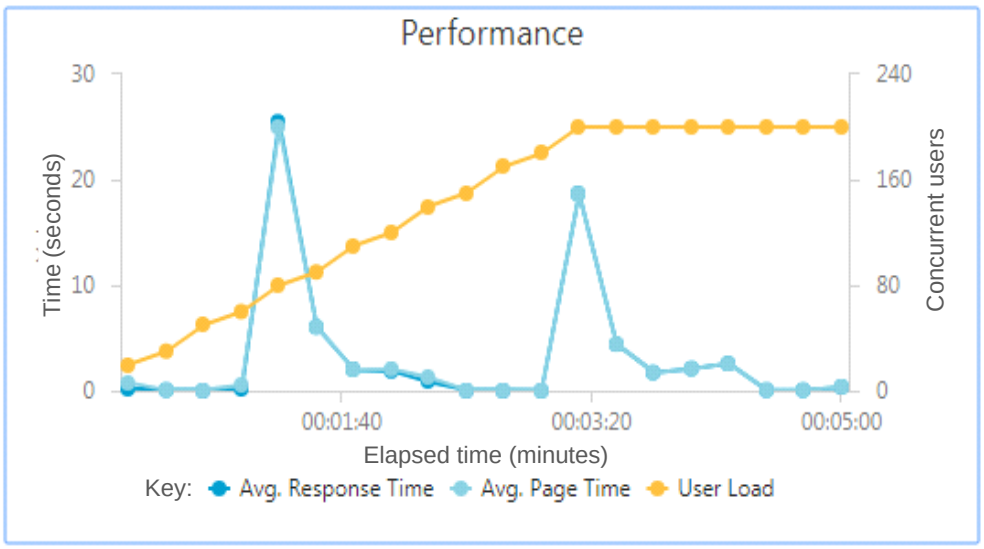

Figure 4.54: Graduated load test "Performance" data for WebApp_14

Throughput

From the throughput results graphed below, it was notable that the number of requests processed per second and the number of pages that were successfully processed per second was not consistent with the increase in user load. In this case, the association between the throughput and user load could not be visualised.

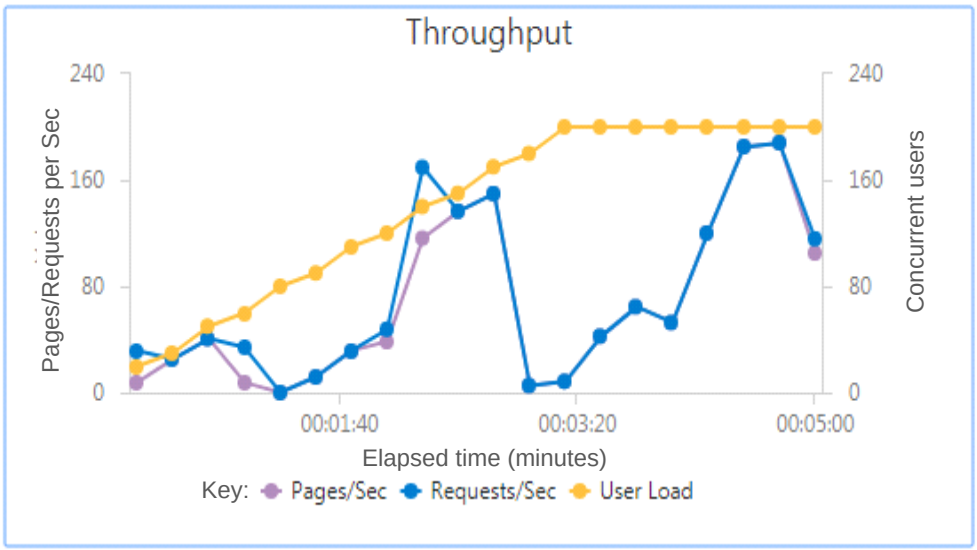

Figure 4.55: Graduated load test "Throughput" data for WebApp_14

Errors

From the errors graph below, an irregular of rate of failed requests per second was recorded throughout the test period.

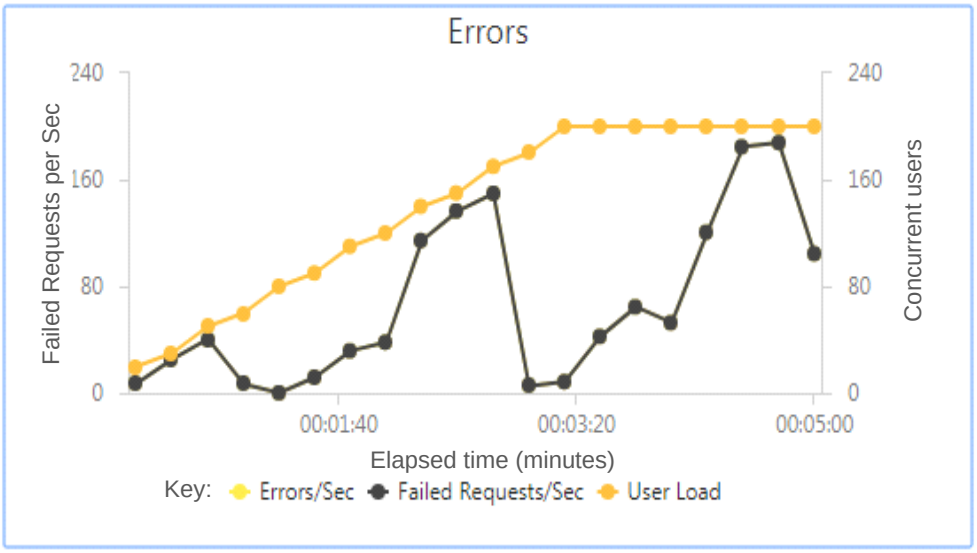

Figure 4.56: Graduated load test "Errors" data for WebApp_14

Tests

From the tests graph below, it was notable that the number of tests processed per second was irregular and not consistent with the number of concurrent users. On the other hand, the average test time increased marginally throughout the test period.

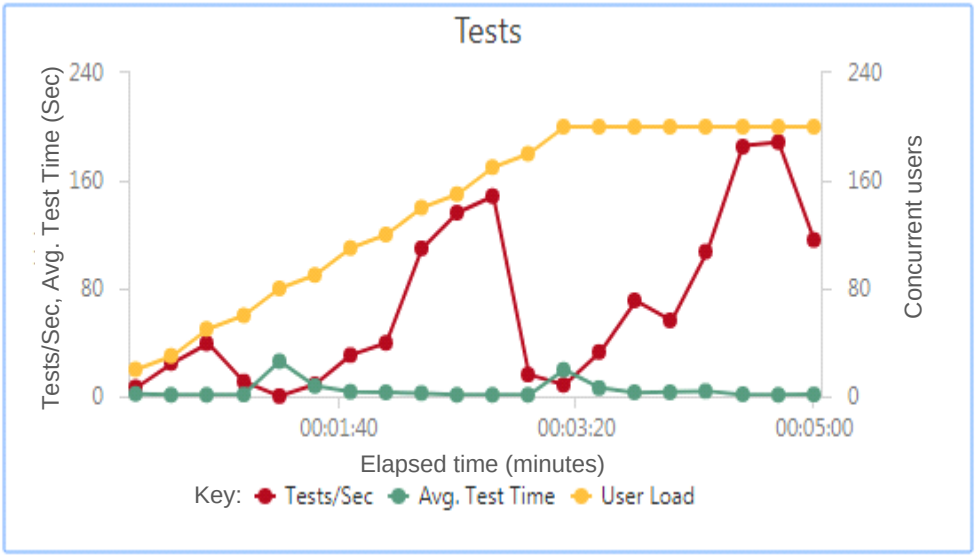

Figure 4.57: Graduated load test "Tests" data for WebApp_14

Discussion

The results show that the test application, WebApp_14 (https://www.ultimateqa.com), which is generic website with rich set of web UI functions, has constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

The application design exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load.

In this case, the association between throughput and user load was not clearly visualised.

4.5.15 Test Application WebApp_15

A generic website with rich set of web UI functions specially designed to the needs of testing of web application of all types.

The table below shows the test results for the graduated load test for this application:

Table 4.17: Graduated load test results for WebApp_15

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	-		-	16.354170	2,861
0:30	30	15.866670	1.617647	15.533330	2.708333	2,883
0:45	50	41.066670	0.064935	41.066670	1.875000	2,901
1:00	60	33.533330	0.063618	33.533330	13.854170	2,869
1:15	80	1.800000	9.259259	1.800000	5.416667	2,863
1:30	90	11.000000	4.272727	11.000000	10.625000	2,833
1:45	110	29.266670	2.154897	29.266670	14.479170	2,801
2:00	120	93.133330	1.532570	94.333340	20.208330	2,740
2:15	140	124.866700	0.071543	124.866700	9.270833	2,764
2:30	150	137.333300	0.080583	137.333300	10.312500	2,745
2:45	170	2.200000	1.606061	2.200000	5.833333	2,738
3:00	180	6.600000	18.414140	6.600000	4.583333	2,726
3:15	200	20.533330	8.961039	20.533330	0.833333	2,727
3:30	200	41.000000	3.617886	41.000000	1.979167	2,727
3:45	200	95.600000	3.262901	95.600000	2.500000	2,724
4:00	200	189.000000	0.065961	189.000000	10.937500	2,725
4:15	200	184.533300	0.065751	184.533300	14.062500	2,729
4:30	200	30.733330	0.088937	30.733330	1.979167	2,733
4:45	200	8.266666	15.629030	8.266666	0.625000	2,734
5:00	200	18.733330	11.946620	18.733330	0.729167	2,735

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time increased in an irregular pattern as the user load increased. However, the values for the two metrics varied significantly throughout the test period, oscillating between the values on the y-axis between 0 and 24 seconds.

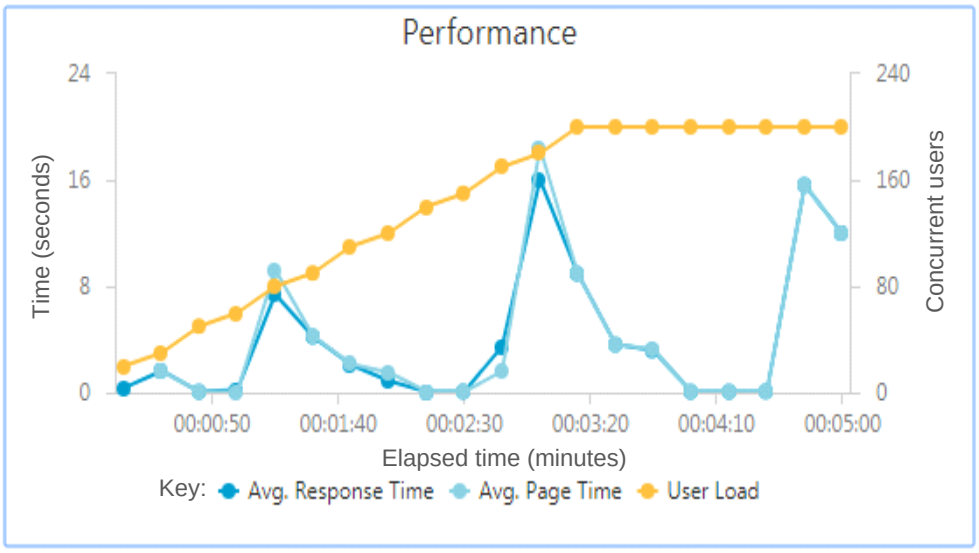

Figure 4.58: Graduated load test "Performance" data for WebApp_15

Throughput

From the throughput results graphed below, it was notable that the number of requests processed per second and the number of pages that were successfully processed per second was not consistent with the increase in user load. In this case, the association between the throughput and user load could not be visualised.

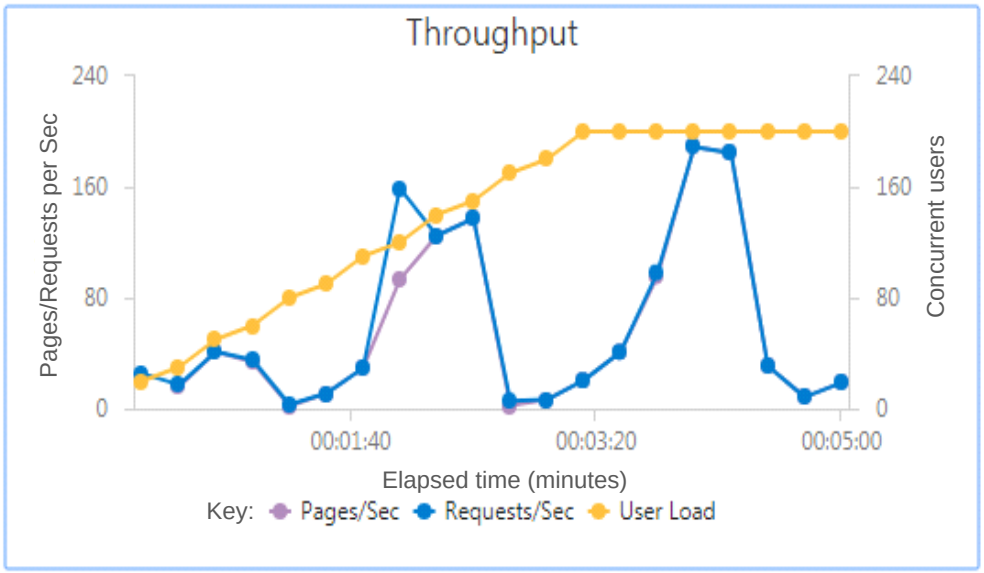

Figure 4.59: Graduated load test "Throughput" data for WebApp_15

Errors

From the errors graph below, an irregular of rate of failed requests per second was recorded throughout the test period.

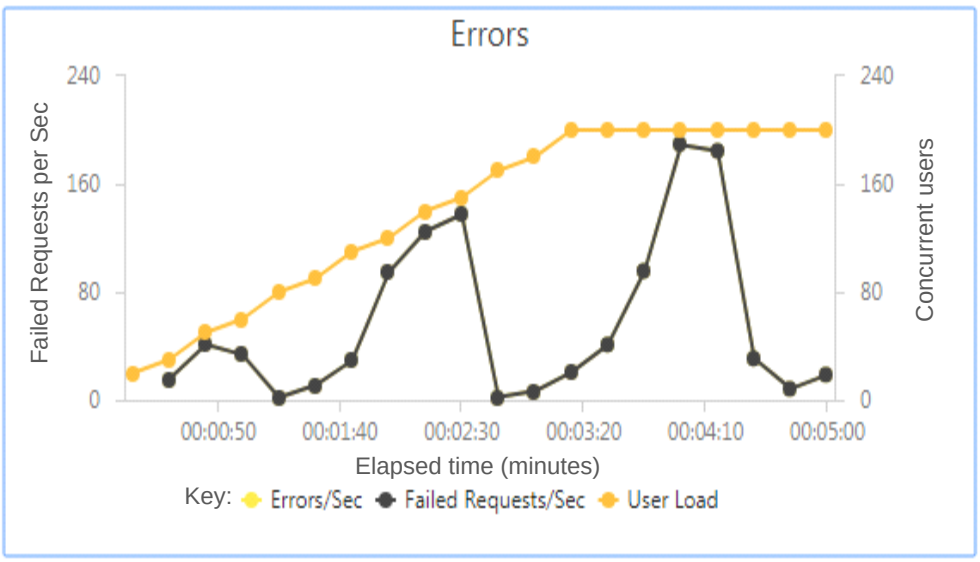

Figure 4.60: Graduated load test "Errors" data for WebApp_15

Tests

From the tests graph below, it was notable that the number of tests processed per second was irregular and not consistent with the number of concurrent users. On the other hand, the average test time increased marginally throughout the test period.

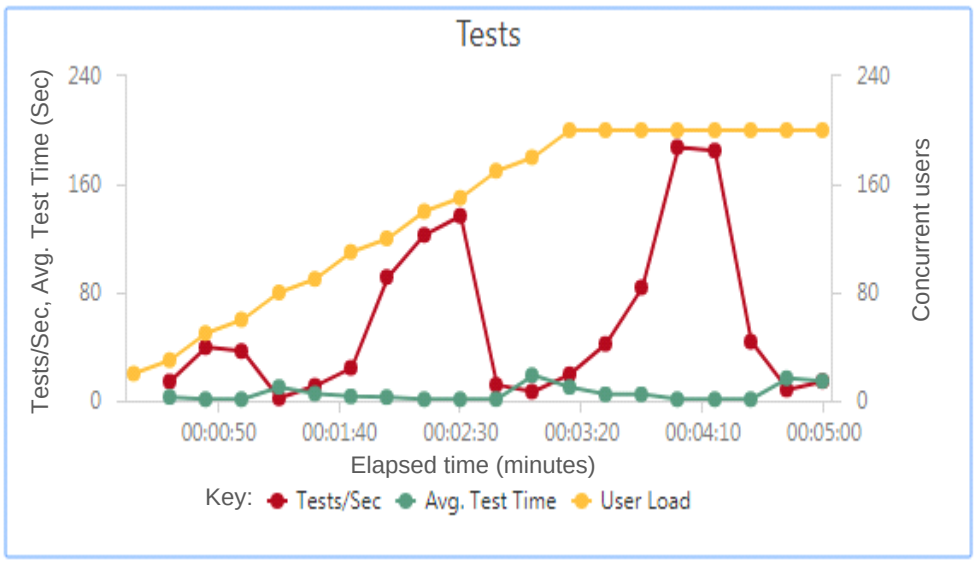

Figure 4.61: Graduated load test "Tests" data for WebApp_15

Discussion

The results show that the test application, WebApp_15 (https://www.qtptutorial.net), which is generic website with rich set of web UI functions, has constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

The application design exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load.

In this case, the association between throughput and user load was not clearly visualised.

4.5.16 Test Application WebApp_16

Portal for IBM open source at GitHub which his based on microservices.

The table below shows the test results for the graduated load test for this application:

Table 4.18: Graduated load test results for WebApp_16

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	13.333330	0.085000	-	22.604170	2,812
0:30	30	26.666670	0.017500	-	5.312500	2,769
0:45	50	42.666670	0.021875	-	15.000000	2,679
1:00	60	56.666670	0.012941	-	15.208330	2,606
1:15	80	72.400000	0.012891	-	14.062500	2,515
1:30	90	84.866670	0.012569	-	24.062500	2,426
1:45	110	101.400000	0.019066	-	33.125000	2,333
2:00	120	112.400000	0.013642	-	28.541670	2,231
2:15	140	130.800000	0.020387	-	23.958330	2,164
2:30	150	143.533300	0.012076	-	19.791670	2,165
2:45	170	157.266700	0.020772	-	30.833330	2,148
3:00	180	169.866700	0.014521	-	27.187500	2,143
3:15	200	183.600000	0.027596	-	33.750000	2,142
3:30	200	198.600000	0.009399	-	21.979170	2,145
3:45	200	192.133300	0.011450	-	24.583330	2,143
4:00	200	197.000000	0.013536	-	33.437500	2,146
4:15	200	195.533300	0.012274	-	29.270830	2,150
4:30	200	191.866700	0.011466	-	28.333330	2,157
4:45	200	197.533300	0.014175	-	31.354170	2,168
5:00	200	193.600000	0.011019	-	29.375000	2,203

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time and the average page load time remained steady throughout the test period in spite of the increase in user load. The values for the two metrics remained relatively low throughout the test period, as depicted by the y-axis scale of between 0 and 0.18 seconds.

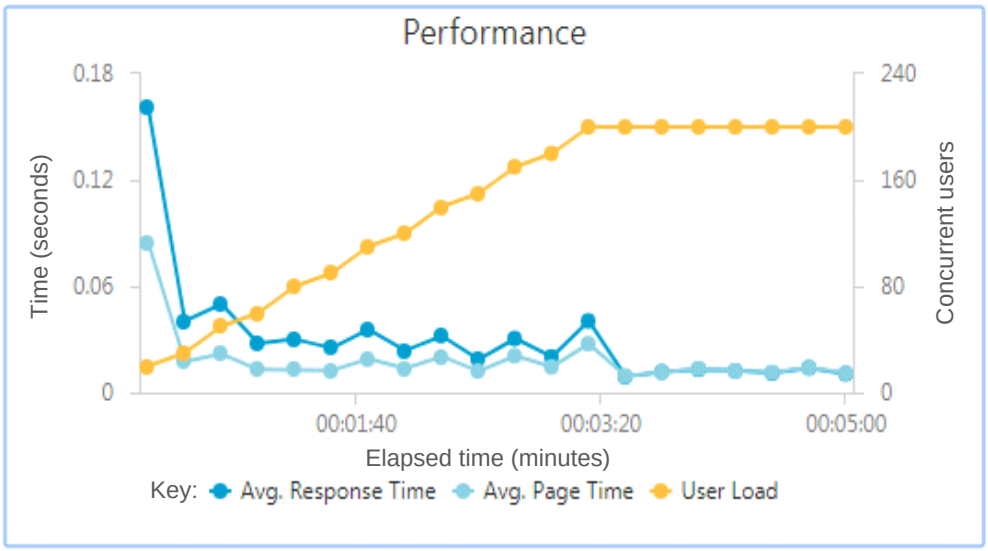

Figure 4.62: Graduated load test "Performance" data for WebApp_16

Throughput

From the throughput results graphed below, it was notable that the number of requests and number of pages that were successfully processed per second increased consistently with the increase in the number of concurrent users. It can therefore be argued that throughput increased with increase in user load.

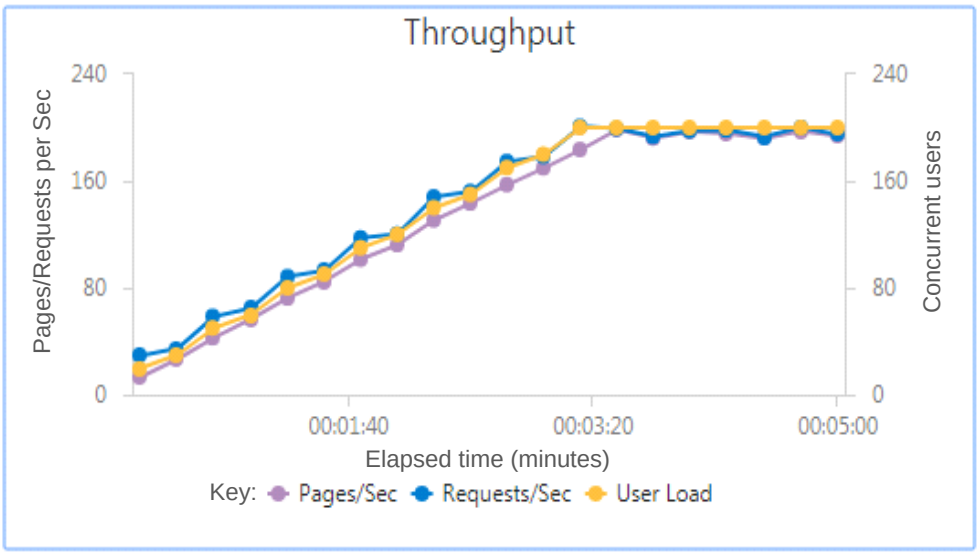

Figure 4.63: Graduated load test "Throughput" data for WebApp_16

Errors

From the errors graph below, was notable that no failed requests were recorded throughout the test period.

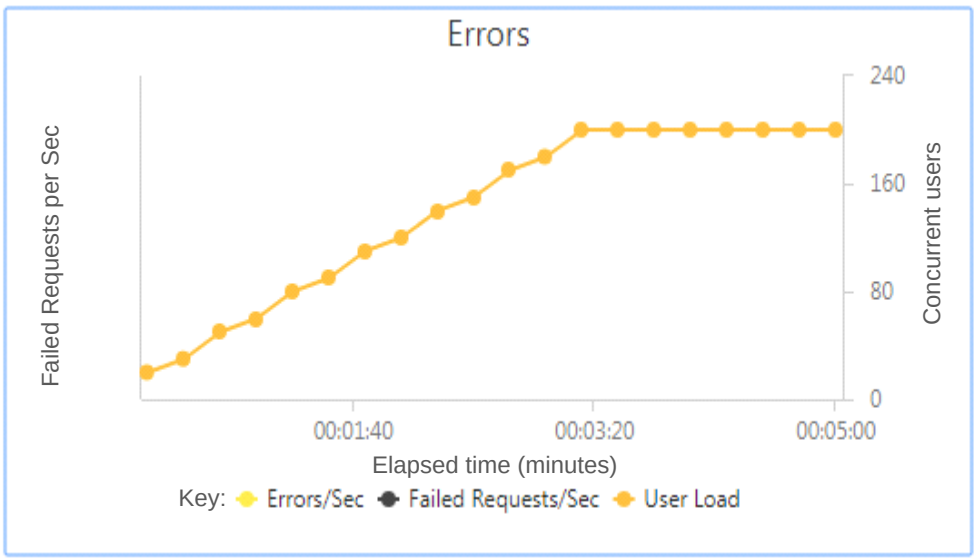

Figure 4.64: Graduated load test "Errors" data for WebApp_16

Tests

From the tests graph below, it was notable that the number of tests processed per second was consistent with the number of concurrent users. On the other hand, the average test time remained steady throughout the test period.

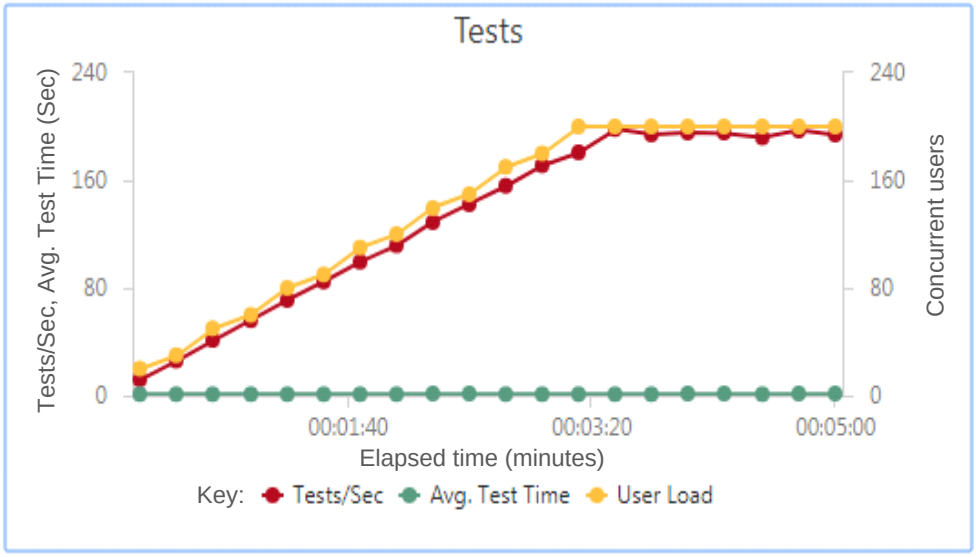

Figure 4.65: Graduated load test "Tests" data for WebApp_16

Discussion

The results show that the test application, WebApp_16 (http://ibm.github.io), which is an insurance web application, has a good architectural design that provided a consistent user experience at all levels of user load used during the test, up to the maximum level of 200 concurrent users.

The application design also exhibited high scalability as throughput increased with increasing user load and no page load errors were recorded during the test.

4.5.17 Test Application WebApp_17

This is a simple, static portal which is based on microservices.

The table below shows the test results for the graduated load test for this application:

Table 4.19: Graduated load test results for WebApp_17

Elapsed Time	No. of users	Pages/Sec	Avg. Page Time	Errors/Sec	% Processor Time	Available Mbytes
0:00						
0:15	20	13.333330	0.050000	-	26.145830	2,850
0:30	30	26.666670	0.025000	-	21.250000	2,867
0:45	50	43.333330	0.032308	-	36.250000	2,836
1:00	60	48.133340	0.292244	-	56.458330	2,869
1:15	80	72.000000	0.041667	-	68.229160	2,829
1:30	90	79.466670	0.103188	-	89.375000	2,808
1:45	110	77.266670	0.421053	-	100.000000	2,724
2:00	120	73.666660	0.914932	-	98.854160	2,668
2:15	140	68.200000	1.552297	-	100.000000	2,630
2:30	150	78.866670	1.622992	-	100.000000	2,606
2:45	170	80.333340	1.791701	-	100.000000	2,536
3:00	180	52.066670	2.354673	-	100.000000	2,480
3:15	200	62.600000	2.536741	-	95.625000	2,471
3:30	200	69.533330	2.619367	-	100.000000	2,395
3:45	200	77.666660	2.242060	-	100.000000	2,468
4:00	200	66.866670	2.658026	-	100.000000	2,437
4:15	200	72.533330	2.476103	-	100.000000	2,436
4:30	200	76.600000	2.408181	-	100.000000	2,444
4:45	200	80.733330	2.415359	-	100.000000	2,439
5:00	200	67.066670	2.692843	-	100.000000	2,441

These test results are further represented in the graphs below and discussed in the respective sections next to the graph.

Performance

From the performance graph below, it was notable that both the average response time increased marginally as the user load increased. The average page load time however, increased in an irregular pattern as the user load increased.

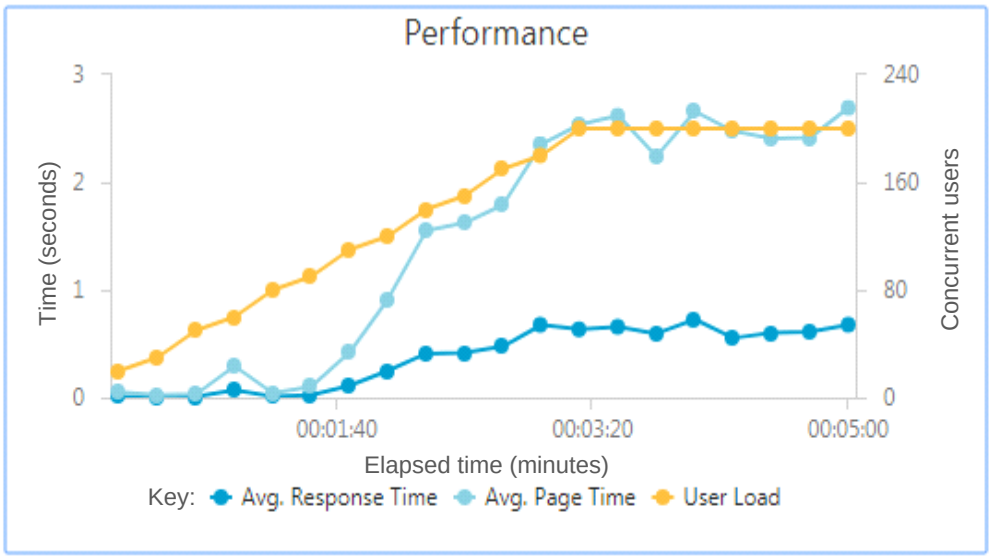

Figure 4.66: Graduated load test "Performance" data for WebApp_17

Throughput

From the throughput results graphed below, it was notable that the number of requests processed per second was not consistent with the increase in user load, increasing steadily to the peak at user load of about 50 users and remaining consistently high for the rest of the test period. The number of pages that were successfully processed per second increased marginally throughout the test period.

The association between throughput and user load could not be clearly visualised.

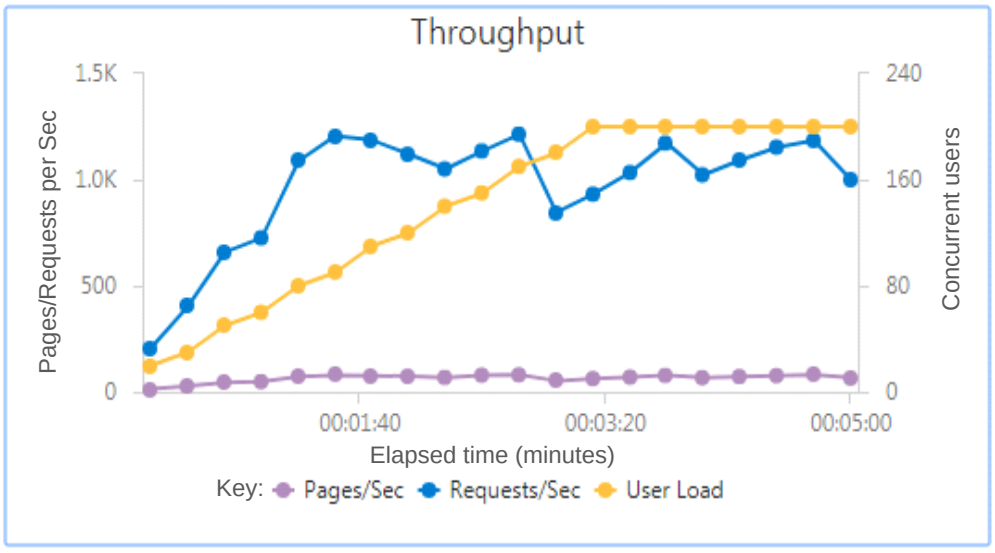

Figure 4.67: Graduated load test "Throughput" data for WebApp_17

Errors

From the errors graph below, was notable that no failed requests were recorded throughout the test period.

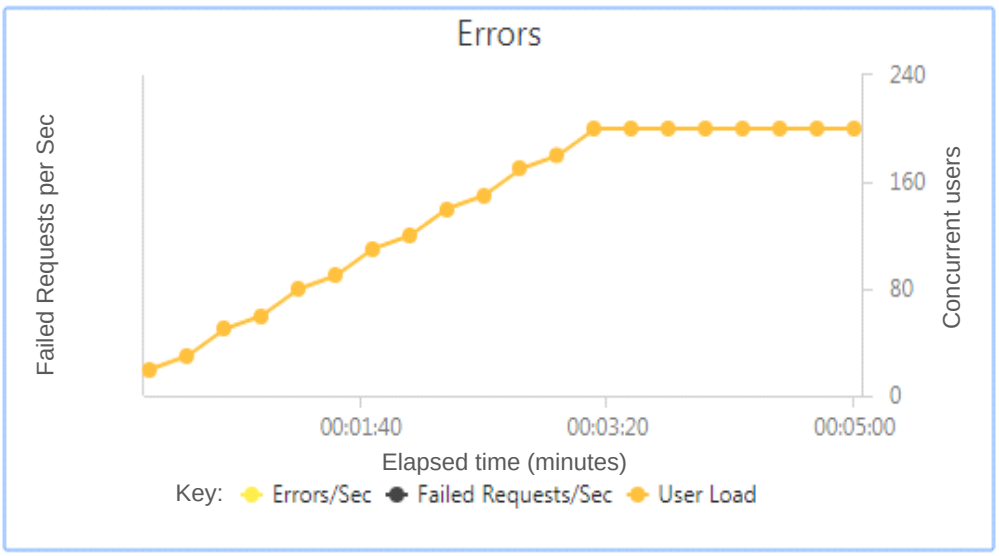

Figure 4.68: Graduated load test "Errors" data for WebApp_17

Tests

From the tests graph below, it was notable that the number of tests processed per second was irregular and not consistent with the number of concurrent users. On the other hand, the average test time increased marginally throughout the test period.

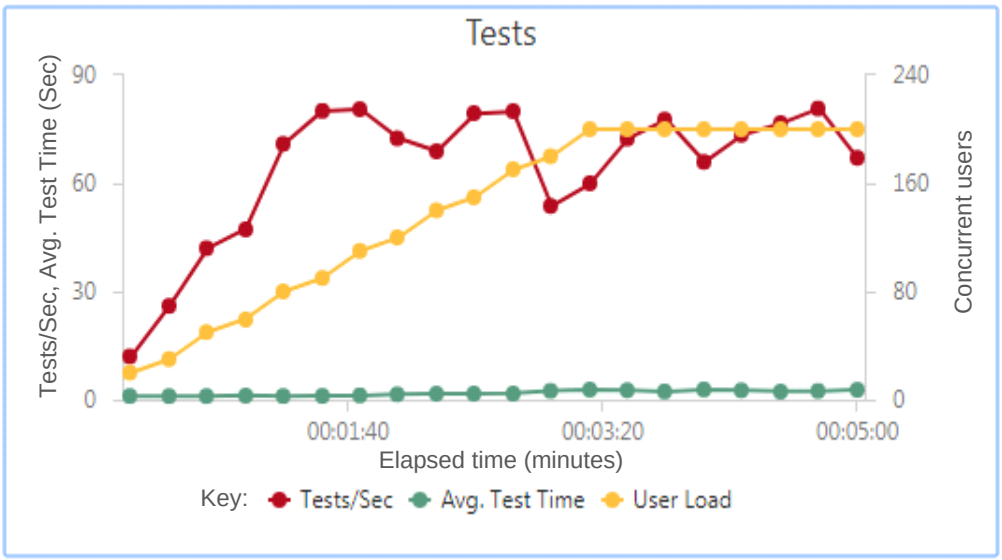

Figure 4.69: Graduated load test "Tests" data for WebApp_17

Discussion

The results show that the test application, WebApp_17 (http://square.github.io), which is a static portal that is based on microservices, has a constrained architectural design that affected user experience depending on the number of concurrent users used during the test, up to the maximum level of 200 concurrent users.

The application design exhibited an irregular pattern in performance, throughput, errors and tests graphs indicating an unpredictable application behaviour when subjected to different levels of user load.

In this case, the association between throughput and user load was not clearly visualised.

4.6 Summary of application test results

The graduated load test results have shown the difference in the behaviour of the test applications when subjected to the treatment of increasing user load.

After analysing the graphs, the applications were grouped into two categories - applications considered have a good architectural design and those with a constrained architectural design.

The characteristics of the two categories of applications are outlined below:

Table 4.20: Characteristics of observed application behavior

Good architectural design application characteristics	Constrained architectural design application characteristics
Performance was consistent throughout the test period in spite of increasing user load	Performance increased and varied in an irregular pattern as user load increased
Throughput increased consistently with increase in user load, illustrating a clear relationship between throughput and user load	Throughput increased in an irregular patter as user load increased such that the relationship between throughput and user load could not be visualised
No errors were recorded	Errors were recorded throughout the test period

Going by these characteristics, the test applications were classified into the two categories as shown in the table below:

Table 4.21: Classification of test applications by architectural design

Good architectural design applications	Constrained architectural design applications
WebApp_3 (http://www.practiceselenium.com)	WebApp_1 (http://automationpractice.com)
WebApp_4 (http://zero.webappsecurity.com)	WebApp_2 (http://newtours.demoaut.com)
WebApp_5 (http://demo.nopcommerce.com)	WebApp_7 (http://store.demoqa.com)
WebApp_6 (http://www.globalsqa.com)	WebApp_8 (http://awful-valentine.com)
WebApp_9 (http://demo.borland.com)	WebApp_10 (http://phptravels.com)
WebApp_12 (http://thedemosite.co.uk)	WebApp_11 (http://demoqa.com)
WebApp_13 (http://www.way2automation.com)	WebApp_14 (https://www.ultimateqa.com)
WebApp_16 (http://ibm.github.io)	WebApp_15 (https://www.qtptutorial.net)
	WebApp_17 (http://square.github.io)

The analysis of these graphical results provided a good basis for conducting a high level evaluation of the performance of the test applications. However, this level of analysis was not adequate for testing the research hypotheses. In order to test the hypotheses, inferential statistical analysis was required. The statistical analysis was conducted using Pearson Correlation Coefficient analysis and Moderation Multiple Regression analysis as discussed in the sections that follow.

4.7 Pearson Correlation Coefficient Analysis

Inferential statistics, is used to make inferences or to project characteristics from a sample to an entire population (Zikmund et al., 2013).

In this study, Pearson Correlation Coefficient (Statistics.laerd.com, 2015) was used to determine the correlation between throughput and scalability of applications.

From the conceptual framework, scalability for a given application A on a platform P was defined as:

$$\text{Scalability } S(A,P) = \frac{\text{Throughput } T(A,P)}{\text{Cost } C(A,P)}$$

The following thesis statement was postulated: *As throughput increases, scalability increases. Therefore, there is a positive relationship between throughput and scalability, such that high values of throughput are associated with high values of scalability.*

The null hypothesis for this test was:

H_0: $\rho = 0$; the correlation coefficient for the population is zero. There is no statistically significant relationship between throughput and scalability of applications.

The alternative hypothesis for this test was:

H_1: $\rho \neq 0$; the correlation coefficient for the population is not equal to zero. There is a statistically significant relationship between throughput and scalability of applications.

4.7.1 Pearson Correlation Coefficient Analysis for 1 test application

SPSS Statistics correlational analysis was used to test the hypothesis by determining the strength and direction of the relationship between throughput and scalability. Data for the test application WebApp_1 (http://automationpractice.com) was used to illustrate the computation process in the sections below.

The table below shows the values used for calculating *r* for throughput and scalability.

Table 4.22: X and Y values used for Correlation Coefficient analysis

Elapsed Time	No. of users	Pages/Sec	% Processor Time	Throughput (T) X-Values	Scalability (S) Y-Values
0:15	20	6.000000	23.437500	6.00	0.26
0:30	30	13.733330	26.875000	13.73	0.51
0:45	50	22.066670	38.125000	22.07	0.58
1:00	60	24.133330	29.895830	24.13	0.81
1:15	80	29.933330	42.812500	29.93	0.70
1:30	90	37.266670	36.666670	37.27	1.02
1:45	110	44.533330	37.604170	44.53	1.18
2:00	120	49.533330	37.500000	49.53	1.32
2:15	140	55.933330	35.520830	55.93	1.57
2:30	150	59.466670	22.187500	59.47	2.68
2:45	170	62.533330	19.479170	62.53	3.21
3:00	180	66.866670	23.020830	66.87	2.90
3:15	200	75.933330	18.020830	75.93	4.21

The step by step analysis process in explained in the sections below.

Step 1: Test for linear relationship between the variables

Pearson's correlation is only appropriate when there is a linear relationship between two variables. The scatter graph was used to visually determine if there was a linear relationship between throughput and scalability as shown below.

From visual inspection of the scatterplot below, it was determined that there was a linear relationship between throughput and scalability.

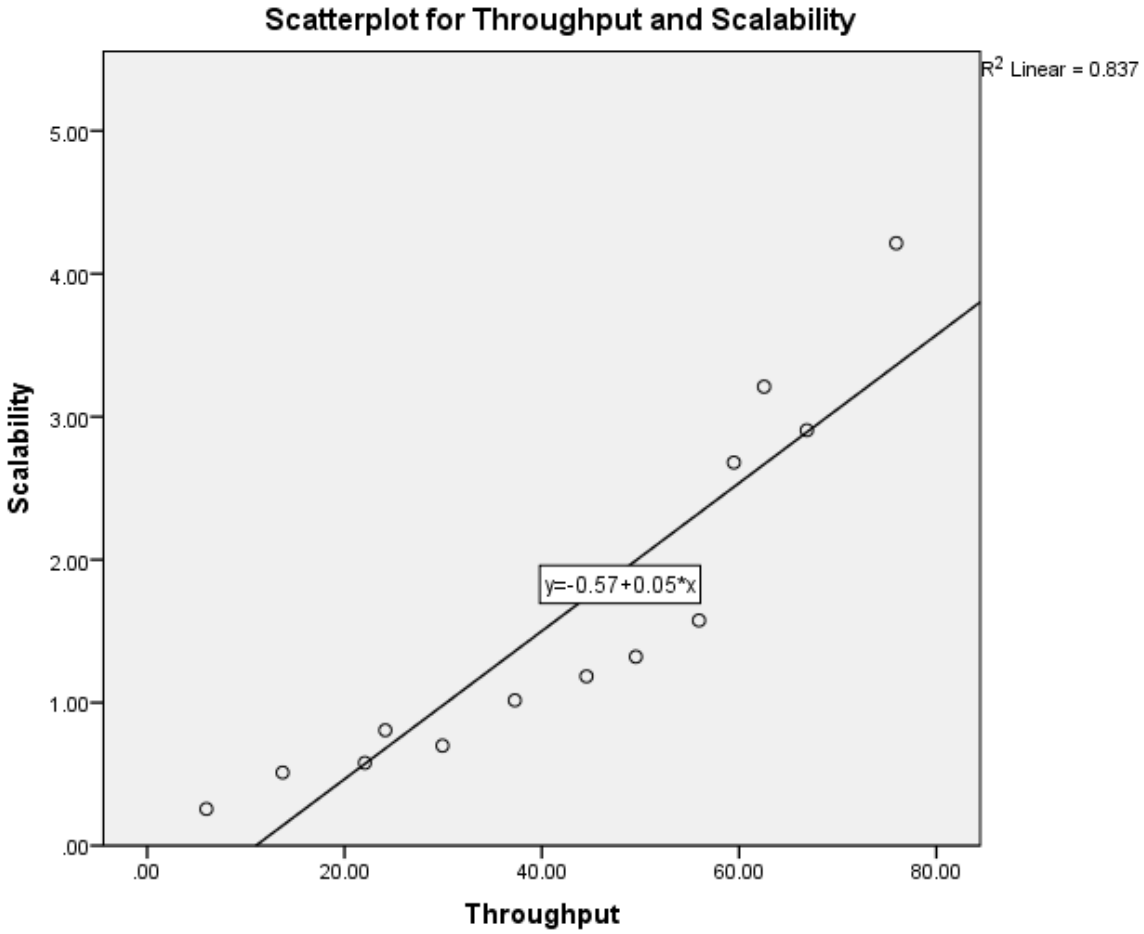

Figure 4.70: Scatterplot 1 showing relationship between Throughput and Scalability

Step 2: Test for outliers

When conducting a Pearson's correlation analysis, outliers are data points that do not fit the pattern of the rest of the data set (Statistics.laerd.com, 2015). These data points can often be identified from the scatterplot which was plotted when testing for linearity. By examining the scatterplot above, it was determined that there were no outliers in the data set.

Step 3: Test for normality

To assess the statistical significance of Pearson's correlation coefficient, you need to have bivariate normality, but this assumption is difficult to assess. Therefore, in practice, a property of bivariate normality is relied upon; that is, if bivariate normality exists, both variables will be normally distributed. However, this does not work in reverse; two normally distributed variables do not mean there is bivariate normality, but it is a level of assurance that can be lived with (Statistics.laerd.com, 2015).

SPSS Statistics was used to test both variables (throughput and scalability) for normality with the following results:

Table 4.23: Normality test results for Throughput and Scalability

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Throughput	.120	13	.200*	.967	13	.855
Scalability	.208	13	.127	.879	13	.070

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

From the results table above, it was observed that a Shapiro-Wilk test was ran for each variable. The significance value for this test for each variable, as highlighted above are: *throughput* = 0.855; *scalability* = 0.070.

If the assumption of normality has been violated, the "Sig." value will be less than .05 (i.e., the test is significant at the $p < .05$ level). If the assumption of normality has not been violated, the "Sig." value will be greater than .05 (i.e., $p > .05$). This is because the Shapiro-Wilk test is testing the null hypothesis that your data's distribution is equal to a normal distribution. Rejecting the null hypothesis means that your data's distribution is not equal to a normal distribution.

By this definition, both the “Sig.” values are greater than 0.05 (being 0.855 and 0.070), therefore the variables *throughput* and *scalability* are normally distributed, as assessed by Shapiro-Wilk's test ($p > .05$).

Step 4: Pearson correlation coefficient analysis

Since the data fulfilled the criteria for Pearson Correlation Coefficient analysis, the analysis was conducted, and the results obtained as shown in the output table below:

Table 4.24: Correlation analysis output showing coefficient values

		Throughput	Scalability
Throughput	Pearson Correlation	1	.915**
	Sig. (2-tailed)		.000
	N	13	13
Scalability	Pearson Correlation	.915**	1
	Sig. (2-tailed)	.000	
	N	13	13

**. Correlation is significant at the 0.01 level (2-tailed).

From the table above, it was determined that the Pearson correlation coefficient, *r*, is .951, which is positive, meaning, there is a positive correlation between throughput and scalability.

The magnitude of the Pearson correlation coefficient determines the strength of the correlation. (Cohen, 1988) provides general guidelines for assigning strength of association with different values for *r* as shown in the table below:

Table 4.25: Coefficient values and strength of association

Coefficient Value	Strength of Association
0.1 < | *r* | < .3	small correlation
0.3 < | *r* | < .5	medium/moderate correlation
| *r* | > .5	large/strong correlation

where | *r* | means the absolute value or r (e.g., | *r* | > .5 means *r* > .5 and *r* < -.5).

Therefore, the Pearson correlation coefficient for this test (*r* = .951) suggests a strong correlation. The Pearson correlation analysis therefore shows there was a strong positive correlation between throughput and scalability, *r* = .951.

Step 5: Coefficient of determination

The coefficient of determination is the proportion of variance in one variable that is "explained" by the other variable and is calculated as the square of the correlation coefficient (r^2). In this analysis, the coefficient of determination, r^2, is equal to $(0.951)^2 = 0.837225$. This can also be expressed as a percentage (i.e., 83.72%).

This means that throughput statistically explained 83.72% of the variability in scalability.

Step 6: Determining statistical significance

The Pearson correlation coefficient obtained from the analysis described the relationship between the two variables in the sample. A test for statistical significance is conducted to test the hypotheses about the linear relationship between the variables in the population the sample is from.

Table 4.26: Correlation analysis output showing statistical significance

		Throughput	Scalability
Throughput	Pearson Correlation	1	.915**
	Sig. (2-tailed)		.000
	N	13	13
Scalability	Pearson Correlation	.915**	1
	Sig. (2-tailed)	.000	
	N	13	13

**. Correlation is significant at the 0.01 level (2-tailed).

From the output table above, the statistical significance (*p*-value) of the correlation coefficient in this analysis would appear to be .000 (obtained from the "Sig. (2-tailed)" row). However, SPSS Statistics *p*-value of .000, indicates that p < .0005. Since $p < .05$ in this case (it is $p < .0005$), it can be concluded that the correlation coefficient is statistically significantly different from zero.

The Pearson correlation analysis shows there was a strong positive correlation between throughput and scalability, $r = .951$, $p < .0005$.

4.7.2 Pearson Correlation Coefficient analysis for all test applications

Pearson correlation coefficient analysis was conducted to determine *r* for the consolidated data for all the test applications.

The consolidated data in the table below was used for the analysis:

Table 4.27: Consolidated X and Y values used for r computation

Pearson Correlation Coefficient Analysis	
Throughput (X-Values)	**Scalability (Y-Values)**
8.4471	0.5299
19.1176	2.8232
30.6314	3.9319
34.1647	2.1752
39.9922	2.1439
47.3098	2.1238
55.1020	2.1032
64.3333	2.6065
77.1843	3.5742
84.8667	4.5394
84.4039	3.3079
76.9922	3.6803
85.0118	6.9434

The step by step analysis process is explained in the sections below.

Step 1: Test for linear relationship between the variables

Pearson's correlation is only appropriate when there is a linear relationship between two variables. The scatterplot was used to visually determine if there was a linear relationship between throughput and scalability as shown below:

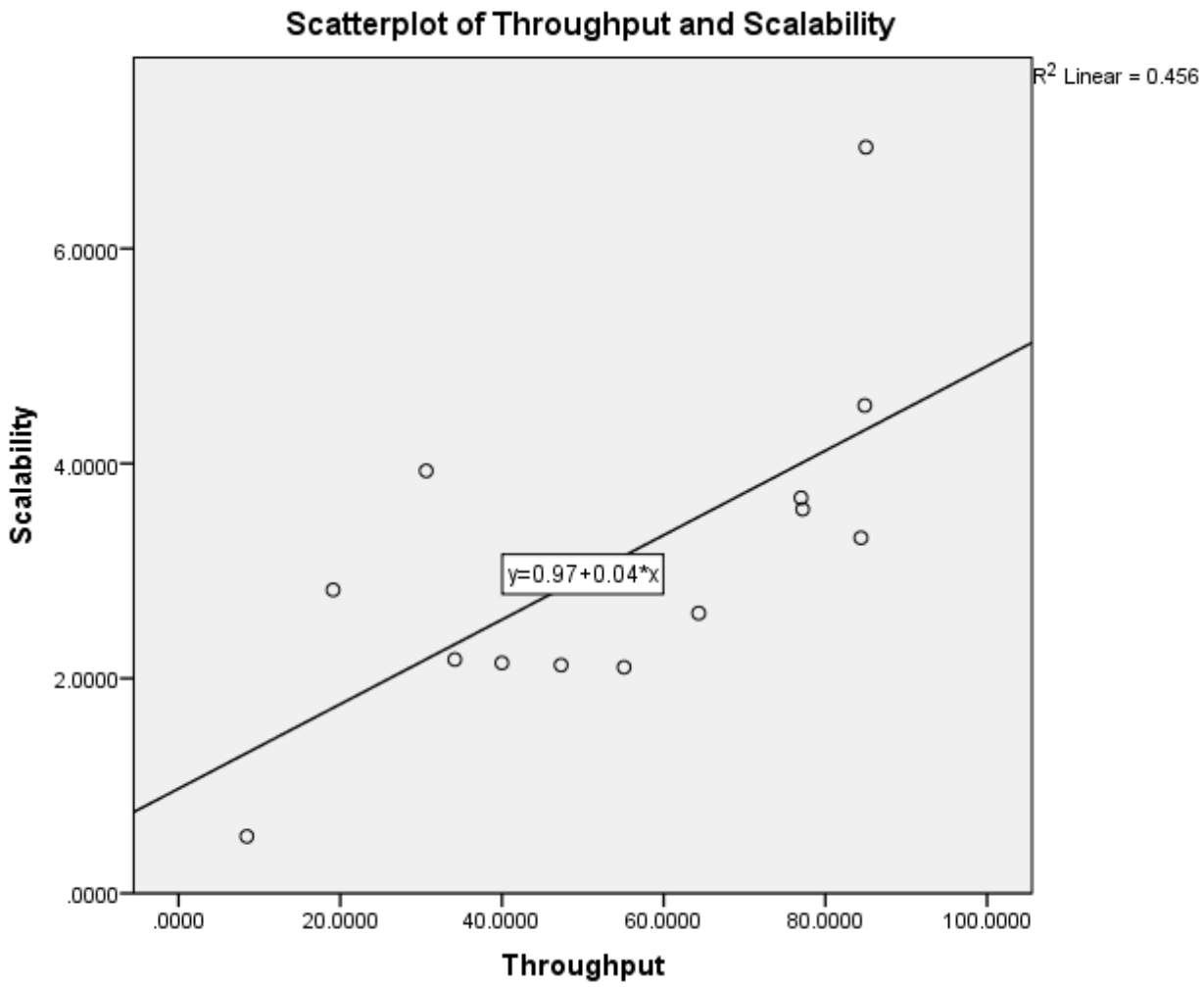

Figure 4.71: Scatterplot 2 showing relationship between Throughput and Scalability

From visual inspection of the scatterplot above, it was determined that there was a linear relationship between throughput and scalability.

Step 2: Test for outliers

When conducting a Pearson's correlation analysis, outliers are data points that do not fit the pattern of the rest of the data set (Statistics.laerd.com, 2015). These data points can often be identified from the scatterplot above, which was plotted when testing for linearity. By examining the scatterplot, it was determined that there were 3 outliers in this data set which are denoted by the 3 black dots on the scatterplot below.

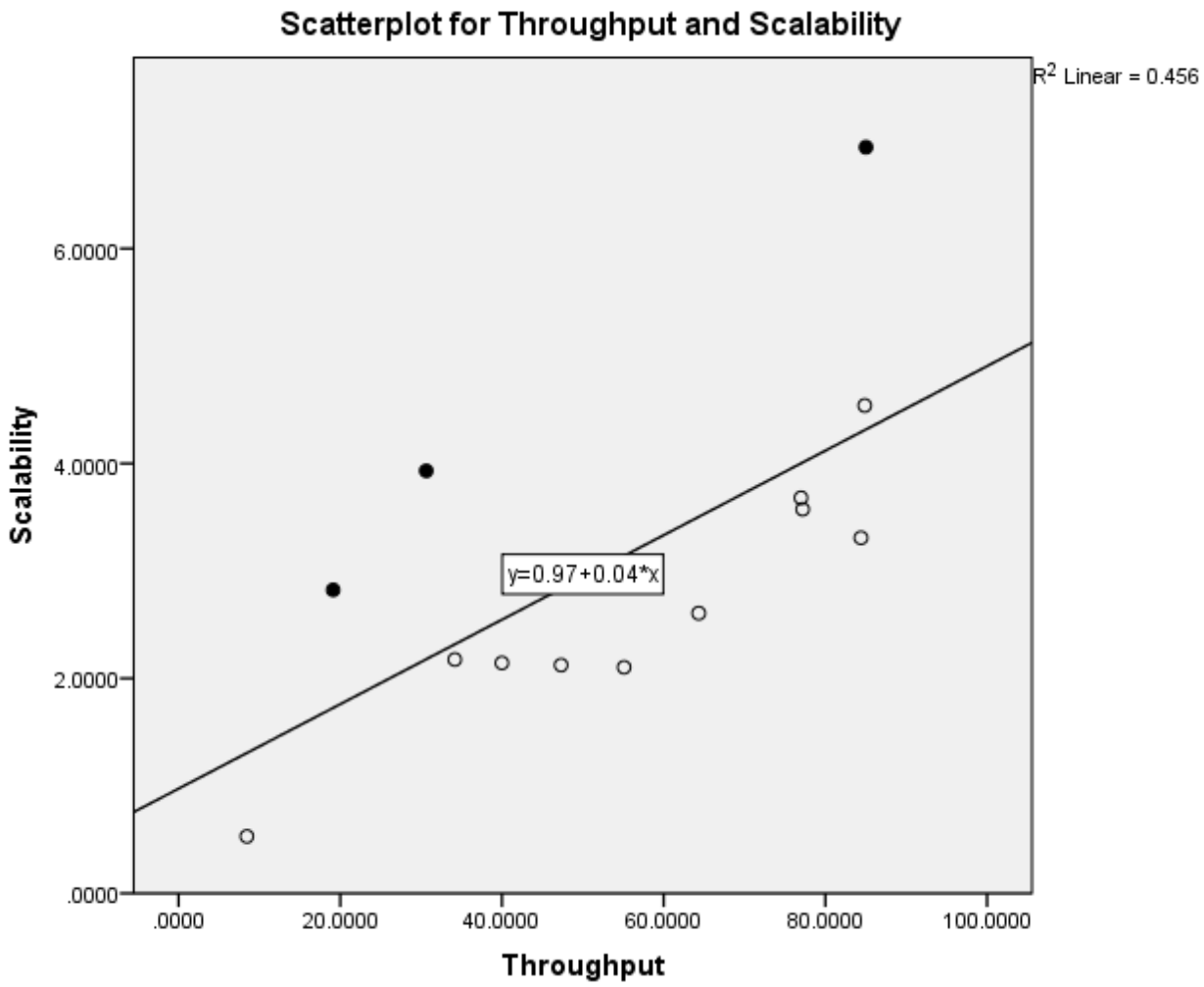

Figure 4.72: Scatterplot 3 showing outliers

A further examination of the data showed that the outliers were contributed by data for 3 test applications which had results that were not consistent with the results of other applications. These applications included:

- Test application WebApp_7 (http://store.demoqa.com)
- Test application WebApp_10 (http://phptravels.com)
- Test application WebApp_11 (http://demoqa.com)

The three applications had low values for throughput and high values of average page load time compared to the other test applications. Since these were valid test results, the outliers were retained in the Pearson correlation analysis.

Step 3: Test for normality

To assess the statistical significance of Pearson's correlation coefficient, you need to have bivariate normality, but this assumption is difficult to assess. Therefore, in practice, a property of bivariate normality is relied upon; that is, if bivariate normality exists, both variables will be normally distributed. However, this does not work in reverse; two normally distributed variables do not mean you have bivariate normality, but it is a level of assurance that can be lived with (Statistics.laerd.com, 2015).

SPSS Statistics was used to test both variables (throughput and scalability) for normality with the following results:

Table 4.28: Normality test results for Throughput and Scalability

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Throughput	.187	13	.200*	.917	13	.228
Scalability	.180	13	.200*	.917	13	.227

*. This is a lower bound of the true significance.
a. Lilliefors Significance Correction

From the results table above, it was observed that a Shapiro-Wilk test was ran for each variable. The significance value for each variable was: Throughput = 0.228; Scalability = 0.227. Since both the "Sig." values are greater than 0.05 (being 0.228 and 0.227), the variables were normally distributed, as assessed by Shapiro-Wilk's test ($p > .05$).

Step 4: Pearson correlation coefficient analysis

Since the data fulfilled the criteria for Pearson Correlation Coefficient analysis, the analysis was conducted, and the results obtained as shown in the output table below:

Table 4.29: Correlation analysis output showing coefficient values

		Throughput	Scalability
Throughput	Pearson Correlation	1	.675*
	Sig. (2-tailed)		.011
	N	13	13
Scalability	Pearson Correlation	.675*	1
	Sig. (2-tailed)	.011	
	N	13	13

*. Correlation is significant at the 0.05 level (2-tailed).

From the table above, it was determined that the Pearson correlation coefficient, r, was .675. Since the sign of the Pearson correlation coefficient is positive and $|r| > 0.5$, there was a strong positive correlation between throughput and scalability.

The Pearson correlation analysis showed there was a strong positive correlation between throughput and scalability, $r = .675$.

Step 5: Coefficient of determination

The coefficient of determination is the proportion of variance in one variable that is "explained" by the other variable and is calculated as the square of the correlation coefficient (r^2). In this analysis, the coefficient of determination, r^2, is equal to $(0.675)^2 = 0.455625$. This can also be expressed as a percentage (i.e., 45.56%). This means that throughput statistically explained 45.56% of the variability in scalability.

Step 6: Determining statistical significance

The Pearson correlation coefficient obtained from the analysis described the relationship between the two variables in the sample. A test for statistical significance was conducted to test the hypotheses about the linear relationship between the variables in the population the sample is from.

Table 4.30: Correlation analysis output showing statistical significance

		Throughput		Scalability
Throughput	Pearson Correlation	1		.675*
	Sig. (2-tailed)			.011
	N	13		13
Scalability	Pearson Correlation	.675*		1
	Sig. (2-tailed)	.011		
	N	13		13

*. Correlation is significant at the 0.05 level (2-tailed).

From the output table above, the statistical significance (p-value) of the correlation coefficient in this analysis was .011. Since $p < .05$, in this case ($p = .011$), it was determined that the correlation coefficient was statistically significantly different from zero.

The Pearson correlation analysis therefore showed there was a strong positive correlation between throughput and scalability, $r = .0.675$, $p = 0.011$.

4.7.3 Discussions

At first, a Pearson Correlation Coefficient analysis was run to assess the relationship between *throughput* and *scalability* for the graduated load test results for one test application.

Preliminary analyses showed the relationship to be linear, with both variables normally distributed, as assessed by Shapiro-Wilk's test ($p > .05$), and there were no outliers. There was a strong positive correlation between *throughput* and *scalability*, $r(11) = .915$, $p < .0005$, with *throughput* explaining 83.72% of the variation in *scalability*.

On the second instance, a Pearson Correlation Coefficient analysis was run to assess the relationship between *throughput* and *scalability* for the graduated load test results for all test applications used in the experiment.

Preliminary analyses showed the relationship to be linear with both variables normally distributed, as assessed by Shapiro-Wilk's test ($p > .05$), with 3 outliers. Correlation analysis results showed that there was a strong positive correlation between *throughput* and *scalability*, $r(11) = .675$, $p = 0.011$, with *throughput* explaining 45.56% of the variation in *scalability*.

Through the Pearson Correlation Coefficient analysis for the graduated load test results, it was determined that there was a strong positive correlation between *throughput* and *scalability*, which was statistically significant. Therefore, the null hypothesis is rejected, and the alternative hypothesis is accepted.

4.8 Moderation Multiple Regression Analysis

In the conceptual framework, the following thesis statement was postulated: *The architecture of a software application controls how the application utilizes computing resources and therefore impacts the performance of the application when processing load.*

The study sought to answer the question: *Does application architecture moderate the relationship between load and application performance?*

The following hypotheses were tested:

Null hypothesis:

H_0: Application architecture does not moderate the relationship between load and performance.

Alternative hypothesis:

H_1: Architecture does moderate the relationship between load and performance.

To examine the research question, moderation regression analysis by (Hayes, 2017) and discussed by (Cooper, 2015) was used to asses if the moderating variable (*architecture*) moderates the relationship between the independent variable (*load*) and the dependent variable (*performance*).

To test for moderation, a multiple linear regression was conducted using SPSS Statistics.

The independent variables of the regression were *load*, *architecture* and the *interaction between load and architecture*. The interaction was created by multiplying independent variable (*load*) and moderator (*architecture*) together. The dependent variable of the regression was *performance*.

To test the hypothesis, the interaction was evaluated for statistical significance, whereby moderation is supported when the interaction is statistically significant.

This moderation regression model is summarised in the conceptual and statistical diagrams below:

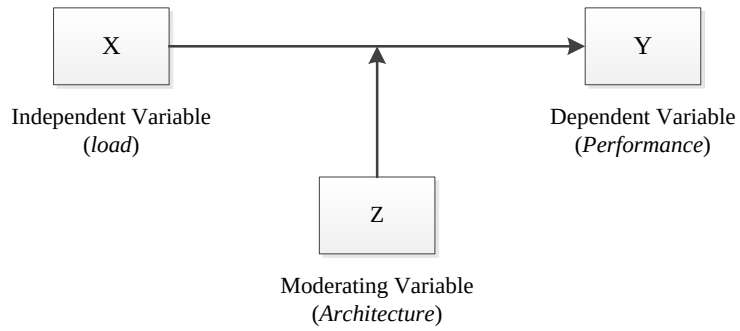

Figure 4.73: Conceptual diagram for moderation regression analysis

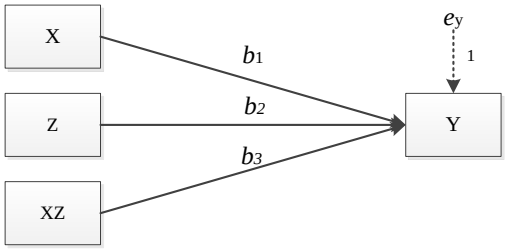

Figure 4.74: Statistical diagram for moderation regression analysis

The conditional effect of X on Y at a given value of Z is defined as: $Y = b_1 + b_3Z$

Moderated regression equation is defined as: $Y = b_0 + b_1X + b_2Z + b_3XZ + e$

4.8.1 Regression analysis for individual test applications

Test application WebApp_1 (http://automationpractice.com)

Data for the test application WebApp_1 (http://automationpractice.com) was used to illustrate the step by step computation process of the moderated multiple regression analysis in the sections below.

The table below shows the values used for calculating the conditional effect of X (*Load*) on Y (*Performance*) at a given value of Z (*Architecture*):

Table 4.31: X, Y and Z values used regression analysis

No. of users	Pages/Sec	Avg. Page Time	% Processor Time	Load (X-Value)	Performance (Y-Value)	Architecture (Z-Value)
20	6.000000	1.922222	23.437500	20	1.922	0.256
30	13.733330	1.796116	26.875000	30	1.796	0.511
50	22.066670	1.858006	38.125000	50	1.858	0.579
60	24.133330	1.872928	29.895830	60	1.873	0.807
80	29.933330	2.167038	42.812500	80	2.167	0.699
90	37.266670	1.919499	36.666670	90	1.919	1.016
110	44.533330	1.697605	37.604170	110	1.698	1.184
120	49.533330	1.776581	37.500000	120	1.777	1.321
140	55.933330	1.722288	35.520830	140	1.722	1.575
150	59.466670	1.791480	22.187500	150	1.791	2.680
170	62.533330	1.826226	19.479170	170	1.826	3.210
180	66.866670	1.794616	23.020830	180	1.795	2.905
200	75.933330	1.828797	18.020830	200	1.829	4.214

Using SPSS Statistics, moderation multiple regression analysis for this data yielded the following results:

Table 4.32: Moderated multiple regression variables

Model	Variables Entered	Variables Removed	Method
1	Arch, Load[b]	.	Enter
2	Load*Arch[b]	.	Enter

a. Dependent Variable: Performance

b. All requested variables entered.

The output table above shows that two regressions were ran.

The first regression, Model 1, contained the independent variable *Load* and *Arch*. The second regression, Model 2, is Model 1 plus the interaction term *Load*Arch* added to the model. Thus, Model 2 contains all three terms in the regression model – *load*, *Arch* and *Load*Arch* – and is the moderated multiple regression.

The table below shows the model summary:

Table 4.33: Moderated multiple regression model summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.495[a]	.245	.095	.1119296	.245	1.626	2	10	.245
2	.577[b]	.333	.111	.1109176	.088	1.183	1	9	.305

a. Predictors: (Constant), Arch, Load

b. Predictors: (Constant), Arch, Load, Load*Arch

c. Dependent Variable: Performance

Looking at results for Model 2, "R Square Change", shows the increase in variation explained by the addition of the interaction term (i.e., the change in R^2). The change in R^2 is reported as .088, which is a proportion. More usually, this measure is reported as a percentage (Laerd Statistics), therefore the change in R^2 was 8.8% (i.e., .088 x 100 = 8.8%). This represents the percentage increase in the variation explained by the addition of the interaction term.

It should be noted that a low R^2 value is a somewhat common feature of an interaction term. As such, one should always consider whether the interaction is practically important (Lund and Lund, 2015).

The next step in the process was to determine if the change in R^2 was statistically significant by examining the result in "Sig. F Change" column. If, the change in R^2 is not statistically significant (i.e., $p > .05$), there is no moderator effect (Lund and Lund, 2015). In this case, the value of "Sig. F Change" $p = 0.305$, which means that the moderator effect was not statistically significant and therefore there was no moderator effect.

In other words, for this specific application performance test, the moderator variable *architecture* does not statistically significantly moderate the relationship between *load* and *performance*.

Test application WebApp_2 (http://demo.nopcommerce.com)

Data for the test application WebApp_2 (http://demo.nopcommerce.com) was used as another example to illustrate the moderated multiple regression analysis in the sections below.

Table 4.34: X, Y and Z values used for regression analysis

No. of users	Pages/Sec	Avg. Page Time	% Processor Time	Load (X-Value)	Performance (Y-Value)	Architecture (Z-Value)
20	12.2000	0.1148	15.5208	20	0.1148	0.7860
30	25.9333	0.0463	5.6250	30	0.0463	4.6104
50	41.6000	0.0465	15.8333	50	0.0465	2.6274
60	54.0000	0.0494	11.6667	60	0.0494	4.6286
80	66.9333	0.1086	20.3125	80	0.1086	3.2952
90	76.2667	0.0857	21.3542	90	0.0857	3.5715
110	89.8000	0.1700	26.6667	110	0.1700	3.3675
120	100.2667	0.1742	17.1875	120	0.1742	5.8337
140	118.4667	0.1300	19.8958	140	0.1300	5.9543
150	132.4667	0.0996	18.7500	150	0.0996	7.0649
170	139.5333	0.1648	21.4583	170	0.1648	6.5025
180	134.8667	0.2813	17.7083	180	0.2813	7.6160
200	137.2000	0.4116	17.0833	200	0.4116	8.0312

Using SPSS, moderation multiple regression analysis for this data yielded the following results:

Table 4.35: Moderated multiple regression variables

Model	Variables Entered	Variables Removed	Method
1	Arch, Load[b]	.	Enter
2	Load*Arch[b]	.	Enter

a. Dependent Variable: Performance

b. All requested variables entered.

The first regression, Model 1, contained the independent variable *Load* and *Arch*. The second regression, Model 2 is Model 1 plus the interaction term *Load*Arch* added to the model. Thus, Model 2 contains all three terms in the regression model – *load*, *Arch* and *Load*Arch* – and is the moderated multiple regression.

The table below shows the model summary:

Table 4.36: Moderated multiple regression model summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.929[a]	.864	.836	.0417900	.864	31.686	2	10	.000
2	.981[b]	.963	.951	.0229265	.099	24.225	1	9	.001

a. Predictors: (Constant), Arch, Load

b. Predictors: (Constant), Arch, Load, Load*Arch

c. Dependent Variable: Performance

Looking at results for Model 2, "R Square Change", shows the increase in variation explained by the addition of the interaction term (i.e., the change in R^2). The change in R^2 is reported as .099, which is a proportion. More usually, this measure is reported as a percentage (Laerd Statistics), therefore the change in R^2 was 9.9% (i.e., .099 x 100 = 9.9%). This represents the percentage increase in the variation explained by the addition of the interaction term.

It should be noted that a low R^2 value is a somewhat common feature of an interaction term. As such, one should always consider whether the interaction is practically important (Lund and Lund, 2015).

The next step in the process was to determine if the change in R^2 was statistically significant by examining the result in "Sig. F Change" column. If, the change in R^2 is statistically significant (i.e., $p < .05$), there is moderation effect (Lund and Lund, 2015). In this case, the value of "Sig. F Change" $p = 0.001$, which means that the moderator effect was statistically significant and therefore there was moderation effect.

In other words, for this specific application performance test, the moderator variable *architecture* statistically significantly moderates the relationship between *load* and *performance*.

4.8.2 Regression analysis for all test applications

Moderation multiple regression analysis was conducted to examine the moderation effect using the consolidated data for all the test applications. The consolidated data in the table below was used for the analysis:

Table 4.37: Consolidated X, Y and Z values used for regression analysis

Moderation Regression Analysis		
Load (X-Value)	**Performance (Y-Value)**	**Architecture (Z-Value)**
20	1.4555	0.5299
30	0.8620	2.8232
50	1.0219	3.9319
60	1.4350	2.1752
80	5.3058	2.1439
90	3.9354	2.1238
110	3.2939	2.1032
120	3.2099	2.6065
140	5.7761	3.5742
150	6.0392	4.5394
170	6.7998	3.3079
180	7.8558	3.6803
200	7.6021	6.9434

Using SPSS Statistics, moderation multiple regression analysis for this data yielded the following results:

Table 4.38: Moderation multiple regression variables

Model	Variables Entered	Variables Removed	Method
1	Arch, Load[b]	.	Enter
2	Load*Arch[b]	.	Enter

a. Dependent Variable: Performance

b. All requested variables entered.

The first regression, Model 1, contained the independent variable *Load* and *Arch*. The second regression, Model 2, is Model 1 plus the interaction term *Load*Arch* added to the model. Thus, Model 2 contains all three terms in the regression model – *load*, *Arch* and *Load*Arch* – and is the moderated multiple regression.

The table below shows the model summary:

Table 4.39: Moderated multiple regression model summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.929[a]	.863	.835	1.0284217	.863	31.439	2	10	.000
2	.933[b]	.871	.828	1.0500131	.008	.593	1	9	.461

a. Predictors: (Constant), Arch, Load

b. Predictors: (Constant), Arch, Load, Load*Arch

c. Dependent Variable: Performance

Looking at results for Model 2, "R Square Change", shows the increase in variation explained by the addition of the interaction term (i.e., the change in R^2). The change in R^2 is reported as .008, which is a proportion. More usually, this measure is reported as a percentage (Laerd Statistics), therefore the change in R^2 was 0.8% (i.e., .008 x 100 = 0.8%). This represents the percentage increase in the variation explained by the addition of the interaction term.

It should be noted that a low R^2 value is a somewhat common feature of an interaction term. As such, one should always consider whether the interaction is practically important (Lund and Lund, 2015).

The next step in the process was to determine if the change in R^2 was statistically significant by examining the result in "Sig. F Change" column. If, the change in R^2 is statistically significant (i.e., $p < .05$), there is moderation effect (Lund and Lund, 2015). In this case, the value of "Sig. F Change" $p = 0.461$, which means that the moderator effect was not statistically significant and therefore there was no moderation effect.

In other words, running the moderation multiple regressing using the consolidated application performance test data, it was determined that moderator variable *architecture* does not statistically significantly moderate the relationship between *load* and *performance*.

4.9 Discussions

At first, moderation multiple regression analysis was ran using test results for one application to assess if the *architecture* moderates the relationship between the *load* and *performance*.

The results for the first test application showed that there was no statistical significance of the interaction term between *architecture* and *load* with change in R^2 reported as .088 and significance reported as $p = 0.305$ ($p > 0.05$).

The results for the second test application showed that there was statistical significance of the interaction term between *architecture* and *load* with change in R^2 reported as .099 and significance reported as $p = 0.001$ ($p < 0.05$).

Finally, moderation multiple regression analysis was ran using the consolidated test results for all applications to assess if the *architecture* moderates the relationship between the *load* and *performance*.

The results showed that there was no statistical significance of the interaction term between *architecture* and *load* with change in R^2 reported as .008 and significance reported as $p = 0.461$ ($p > 0.05$).

Through the moderation multiple regression analysis for the graduated load test results, it was determined that there no statistically significant moderator effect of architecture on the relationship between *load* and *performance*.

Therefore, the null hypothesis was accepted, and the alternative hypothesis was rejected.

5 CONCLUSION AND RECOMMENDATIONS

5.1 Research objectives

The objectives of this research had been outlined as follows:

- Identify factors driving the adoption of cloud computing as the new way of delivering computing services
- Discover the main application architectures used in the development of cloud based applications
- Conduct an experiment to measure and compare the performance of applications when subjected to different levels of load
- Analyse the data to determine the correlation between throughput and scalability
- Analyse the data to determine the moderating effect of architecture on the relationship between load and performance of the cloud-based applications.

5.2 Conclusions

Through a detailed literature search and review, development of a conceptual framework, using a quasi-experimental methodology for testing and data collection and using inferential statistical analysis tools, the research objectives were achieved.

Factors driving the adoption of cloud computing as the new way of delivering computing services were identified, primarily being the access to highly elastic computing resources that are costed based on usage.

The main application architectures used for cloud based applications were identified as Services Oriented Architecture and microservices architecture.

Using Microsoft Visual Studio Team Services on Microsoft Azure cloud platform, graduated load tests were conducted and performance data recorded for all applications in the sample population.

Two hypotheses were tested, whereby with the first hypothesis, the Alternative hypothesis (H_1) was accepted and in the second case, the Null hypothesis (H_0) was accepted.

5.2.1 Hypothesis 1

Through the Pearson Correlation Coefficient analysis for the graduated load test results, it was determined that there was a strong positive correlation between *throughput* and *scalability*, which was statistically significant.

Therefore, the null hypothesis was rejected, and the alternative hypothesis was accepted.

5.2.2 Hypothesis 2

Through the moderation multiple regression analysis for the graduated load test results, it was determined that there was no statistically significant moderator effect of architecture on the relationship between *load* and *performance*.

Therefore, the null hypothesis was accepted, and the alternative hypothesis was rejected.

5.3 Limitations of the investigation

There were several limitations in this study that are worth noting, as discussed in the sections below. Addressing these limitations in future investigations will increase the level of internal and external validity of the research findings and therefore the possibility to generalize the findings to the general population of cloud based applications.

5.3.1 True experimental design

A quasi-experimental methodology was used for this study, with convenience sampling and without a control group. This limited the internal validity of the experimental results. For a true experiment design, a random sample selected method should be used.

5.3.2 Direct study of application architecture factors

This study took an indirect approach to evaluate the relationship between architecture and performance by measuring the utilization of cloud computing resources during the graduated load test. A direct methodology of testing the actual application architecture factors should be considered. An ordinal scale may be considered to group various architecture factors to describe an architecture pattern as “good”, “fair”, or “bad”. This can then be used as a moderating variable for regression analysis.

5.3.3 Application load level

For the graduated load test, light load levels were used. The experiment therefore did not the test application performance at high load and the buckle zone. This limited the understanding of the performance characteristics of these applications at high load level and therefore how the performance characteristics related to the architecture of the applications.

5.3.4 Range of tests

On the other hand, the performance tests conducted were limited to web access and page response times. These tests can be diversified to include online business transactions such as e-commerce and financial transactions. The tests can also be diversified to test data base query processing transactions.

5.4 Recommendations for further research

This has been a very foundation study in the area of application architecture and performance management of cloud based applications, where there has been limited academic research, according to the literature search and review conducted.

Future studies in this area of cloud computing are therefore encouraged, particularly to improve on the conceptual framework and the experimental design so as to increase the internal and external validity of the research findings.

5.5 Implications to practitioners

The subject of cloud computing, application architecture and application performance are very important to cloud computing practitioners, who include cloud computing consultants, cloud services design professionals, solution architects, application performance management solution providers and professionals, business leaders among others.

This study has brought out very important factors that should be considered when businesses are developing a cloud computing strategy for business applications:

- Applications should be designed for high scalability, which translates to high throughput and therefore the ability to process more transactions for more users without impacting performance.

- It is important to check the performance of an application before it is launched or before deploying updates to production.
- Through such performance tests, key decisions regarding the application readiness to meet the performance expectations for the targeted user base can be determined, avoiding frustration from users and potential loss of business due to failure of services at peak loads.
- With test performance data available, businesses can make well informed decisions regarding whether to migrate existing applications to the cloud or to develop new cloud based applications, or even, to maintain their existing on premise applications.

BIBLIOGRAPHY

1 EECS Department, University of California, Berkeley (2009). Above the Clouds: A Berkeley View of Cloud Computing. [online] Available at: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html [Accessed 7 May 2017].

2 Pallis, G. (2010). Cloud Computing: The New Frontier of Internet Computing. *IEEE Internet Computing*, 14(5), pp.70-73.

3 Yu, W. and Chen, J. (2011). Semantic Service in Cloud Computing. In: *Advances in Information Technology and Education. Communications in Computer and Information Science, vol 201*. [online] Berlin, Heidelberg: Springer, Berlin, Heidelberg, p.156. Available at: https://link.springer.com/chapter/10.1007/978-3-642-22418-8_23 [Accessed 9 May 2017].

4 Research ICT Africa (2013). The cloud over Africa. [online] Cape Town: Research ICT Africa, pp.1-2. Available at: https://www.researchictafrica.net/publications/Evidence_for_ICT_Policy_Action/Policy_Paper_20_-_The_cloud_over_Africa.pdf [Accessed 12 May 2017].

5 Sriram, I. and Khajeh-Hosseini, A. (2010). Research Agenda in Cloud Technologies. In: 1st ACM Symposium on Cloud Computing, SOCC 2010. [online] pp.1-2. Available at: https://arxiv.org/ftp/arxiv/papers/1001/1001.3259.pdf [Accessed 12 May 2017].

6 Dai, C. (2017). Forrester Predictions: Ten Key Developments In Cloud Computing Shape The Industry In 2017. [Blog] Charlie Dai's Blog. Available at: http://blogs.forrester.com/charlie_dai/16-11-03-forrester_predictions_ten_key_developments_in_cloud_computing_shape_the_industry_in_2017 [Accessed 14 May 2017].

7 Linthcum, D. (2016). The Benefits of Cloud Computing for the Enterprise. [Blog] *CloudAcademy Blog*. Available at: http://cloudacademy.com/blog/the-benefits-of-cloud-computing-for-the-enterprise/ [Accessed 14 May 2017].

8 Turner, S. (2012). Journal of Technology Research. [online] Academic and Business Research Institute (ABRI), p.2. Available at: http://www.aabri.com/OC2012Manuscripts/OC12079.pdf [Accessed 14 May 2017].

9 Alkhalil, A., Sahandi, R. and John, D. (2016). A decision process model to support migration to cloud computing. International Journal of Business Information Systems,

[online] 24(1), pp.102-126. Available at: https://dl.acm.org/citation.cfm?id=3031057 [Accessed 22 Oct. 2017].

10 Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. NIST Special Publication 800-145. [online] Gaithersburg, MD: National Institute of Standards and Technology (NIST), pp.2-3. Available at: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf [Accessed 17 May 2017].

11 Danielson, K. (2008). Distinguishing Cloud Computing from Utility Computing. [Blog] SaaS Week. Available at: http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/ [Accessed 17 May 2017].

12 En.wikipedia.org. (2011). Cloud computing. [online] Available at: https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=411184904 [Accessed 18 May 2017].

13 Gartner. (2008). Gartner Says Cloud Computing Will Be As Influential As E-business. [online] Available at: http://www.gartner.com/newsroom/id/707508 [Accessed 18 May 2017].

14 Knorr, E. (2008). What cloud computing really means. [Blog] Infoworld. Available at: http://www.infoworld.com/article/2683784/cloud-computing/what-cloud-computing-really-means.html [Accessed 18 May 2017].

15 The Economist. (2009). Cloud computing: Clash of the clouds. [online] Available at: http://www.economist.com/node/14637206 [Accessed 18 May 2017].

16 Beal, V. (2017). What is Cloud Computing? Webopedia Definition. [online] Webopedia.com. Available at: http://www.webopedia.com/TERM/C/cloud_computing.html [Accessed 18 May 2017].

17 Cuttitta, A. (2013). *Talking about technology: A metaphoric analysis of Cloud computing and Web 2.0*. Masters Degree. Northern Arizona University.

18 En.wikipedia.org. (2017). Cloud computing. [online] Available at: https://en.wikipedia.org/wiki/Cloud_computing [Accessed 18 May 2017].

19 Johnston, S. (2017). *Cloud computing conceptual diagram*. [image] Available at: https://en.wikipedia.org/wiki/Cloud_computing#/media/File:Cloud_computing.svg [Accessed 18 May 2017].

20 Biswas, S. (2011). Cloud Computing vs. Utility Computing vs. Grid Computing: Sorting The Differences. [online] Cloudtweaks.com. Available at:

https://cloudtweaks.com/2011/02/cloud-computing-vs-utility-computing-vs-grid-computing-sorting-the-differences/ [Accessed 18 May 2017].

21 Walker, G. (2012). Cloud computing fundamentals - A different way to deliver computer resources. [online] Ibm.com. Available at: https://www.ibm.com/developerworks/cloud/library/cl-cloudintro [Accessed 18 May 2017].

22 Walker, G. (2012). Cloud computing layers embedded in the "as a Service" components. [image] Available at: https://www.ibm.com/developerworks/cloud/library/cl-cloudintro/figure2.gif [Accessed 18 May 2017].

23 Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., LEE, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010). A View of Cloud Computing. *Communications of the ACM*, [online] (Volume 53 Issue 4), pp.50-58. Available at: http://dl.acm.org/citation.cfm?id=1721672 [Accessed 19 May 2017].

24 En.wikipedia.org. (2015). *Data center*. [online] Available at: https://en.wikipedia.org/wiki/Data_center [Accessed 19 May 2017].

25 Arias, T. (2011). *The cloud computing infrastructure handbook*. 1st ed. Emereo Publishing, p.1.

26 Amazon Web Services (2015). *Inside Amazon's Cloud Computing Infrastructure*. [image] Available at: http://datacenterfrontier.com/wp-content/uploads/2015/09/amazon-dc-hamilton.jpg [Accessed 19 May 2017].

27 Somepalle, S. (2015). Five Essential Characteristics of Cloud Computing. [image] Available at: https://media.licdn.com/mpr/mpr/shrinknp_400_400/AAEAAQAAAAAAAAJuAAAAJDdlOTVkMmZhLTQ4MTUtNDFlNy1hOTcxLTM2ZDljYzNlZjNhMg.png [Accessed 27 May 2017].

28 MacVittie, L. (2008). *4 Things You Need in a Cloud Computing Infrastructure*. [online] Devcentral.f5.com. Available at: https://devcentral.f5.com/articles/4-things-you-need-in-a-cloud-computing-infrastructure [Accessed 27 May 2017].

29 MacVittie, L. (2013). Next Generation Application Architecture. [Blog] *F5 DevCentral*. Available at: https://devcentral.f5.com/articles/next-generation-application-architecture [Accessed 27 May 2017].

30 MacVittie, L. (2013). *Next Generation Application Architecture*. [image] Available at: http://file:///C:/Users/macvittie/AppData/Local/Temp/WindowsLiveWriter1286139640/supfiles1DF55BB3/evolvingapparchitecture6.png [Accessed 27 May 2017].

31 Garlan, D. and Shaw, M. (1994). An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, [online] Volume I. Available at: http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf [Accessed 27 May 2017].

32 Patterns & Practices Team, M. (2009). *Microsoft® Application Architecture Guide (Patterns & Practices) Second Edition*. 2nd ed. [ebook] Microsoft Press. Available at: https://msdn.microsoft.com/en-us/library/ff650706.aspx [Accessed 27 May 2017].

33 Rouse, M., Nolle, T. and Li, T. (2017). *What is application program interface (API)? - Definition from WhatIs.com*. [online] SearchMicroservices. Available at: http://searchmicroservices.techtarget.com/definition/application-program-interface-API [Accessed 27 May 2017].

34 Rouse, M. and Wigmore, I. (2016). *What is monolithic architecture? - Definition from WhatIs.com*. [online] WhatIs.com. Available at: http://whatis.techtarget.com/definition/monolithic-architecture [Accessed 27 May 2017].

35 Annett, R. (2014). *What is a Monolith? - Coding the Architecture*. [online] Codingthearchitecture.com. Available at: http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html [Accessed 27 May 2017].

36 Annette, R. (2014). *Module monolith*. [image] Available at: http://www.codingthearchitecture.com/images/monolith/module.png [Accessed 27 May 2017].

37 Annette, R. (2014). *Allocation monolith*. [image] Available at: http://www.codingthearchitecture.com/images/monolith/allocation.png [Accessed 27 May 2017].

38 Annette, R. (2014). *Runtime monolith*. [image] Available at: http://www.codingthearchitecture.com/images/monolith/runtime.png [Accessed 27 May 2017].

39 Richardson, C. (2017). *What are microservices?*. [online] Microservices.io. Available at: http://microservices.io/ [Accessed 1 Jun. 2017].

40 Huston, T. (2017). *What is Microservices Architecture?*. [online] Smartbear.com. Available at: https://smartbear.com/learn/api-design/what-are-microservices/ [Accessed 2 Jun. 2017].

41 Lewis, J. and Fowler, M. (2014). *Microservices*. [online] martinfowler.com. Available at: https://martinfowler.com/articles/microservices.html [Accessed 2 Jun. 2017].

42 Thomas, A. and Gupta, A. (2017). *Innovation Insight for Microservices*. [online] Gartner.com. Available at: https://www.gartner.com/doc/3579057?plc=ddc [Accessed 6 Jun. 2017].

43 Wetherill, J. (2014). *Microservices and PaaS (Part I) - DZone Integration*. [online] dzone.com. Available at: https://dzone.com/articles/microservices-and-paas-part-1 [Accessed 2 Jun. 2017].

44 Gooen, O. (2014). *Advantages and Disadvantages of a Monolith Application*. [online] Impact.hackpad.com. Available at: https://impact.hackpad.com/Advantages-and-Disadvantages-of-a-Monolith-Application-ZlrQRl3LHCg [Accessed 2 Jun. 2017].

45 APMdigest - Application Performance Management. (2016). *Top Factors That Impact Application Performance 2016 - Part 1*. [online] Available at: http://www.apmdigest.com/top-factors-that-impact-application-performance-2016-1 [Accessed 6 Jun. 2017].

46 APMdigest - Application Performance Management. (2016). *Top Factors That Impact Application Performance 2016 - Part 1*. [online] Available at: http://www.apmdigest.com/top-factors-that-impact-application-performance-2016-1 [Accessed 6 Jun. 2017].

47 En.wikipedia.org. (2017). *Application performance management*. [online] Available at: https://en.wikipedia.org/wiki/Application_performance_management [Accessed 6 Jun. 2017].

48 APMDigest (2016). *Top Factors That Impact Application Performance 2016 - Part 4*. Top Factors That Impact Application Performance 2016. [online] APMDigest, p.1. Available at: http://www.apmdigest.com/top-factors-that-impact-application-performance-2016-4 [Accessed 9 Jun. 2017].

49 Bordens, K. and Abbott, B. (2011). *Research design and methods*. 8th ed. New York: McGraw-Hill, p.125.

50 Statistics.laerd.com. (2013). *Pearson Product-Moment Correlation*. [online] Available at: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php [Accessed 12 Jun. 2017].

51 Richardson, C. (2017). Microservices Pattern: Microservice Architecture pattern. [online] microservices.io. Available at: http://microservices.io/patterns/microservices.html [Accessed 22 Jul. 2017].

52 Guiding Metrics. (2017). *Cloud Services Industry's 10 Most Critical Metrics - Guiding Metrics*. [online] Available at: http://guidingmetrics.com/content/cloud-services-industrys-10-most-critical-metrics/ [Accessed 16 Aug. 2017].

53 Sobel, W., Nguyen, J., Wong, H., Klepchukov, A., Fox, A., Patterson, D., Subramanyam, S., Sucharitakul, A. and Pati, S. (2017). *Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0*. [online] p.6. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.7403&rep=rep1&type=pdf [Accessed 16 Aug. 2017].

54 Cloudharmony.com. (2017). *Provider Directory | CloudHarmony*. [online] Available at: https://cloudharmony.com/cloudsquare/cloud-cdn-and-compute-and-dns-and-storage [Accessed 18 Aug. 2017].

55 Regoniel, P. (2016). *CONCEPTUAL FRAMEWORK DEVELOPMENT HANDBOOK. 1st ed*. [ebook] Simplyeducate.me. Available at: http://simplyeducate.me/2015/01/05/conceptual-framework-guide/ [Accessed 6 Nov. 2017].

56 McGaghie, W., Bordage, G. and Shea, J. (2001). *Problem Statement, Conceptual Framework, and Research Question. Academic Medicine*, [online] 76(9), pp.923-924. Available at: http://journals.lww.com/academicmedicine/Fulltext/2001/09000/Problem_Statement,_Conceptual_Framework,_and.21.aspx [Accessed 6 Nov. 2017].

57 Reitbauer, A. (2008). *Performance vs. Scalability*. [Blog] Dayntrace. Available at: https://www.dynatrace.com/blog/performance-vs-scalability/ [Accessed 6 Nov. 2017].

58 Haines, S. (2006). *Pro Java EE 5 performance management and optimization*. Berkeley, Calif: Apress, p.224.

59 Gartner (2010). *Keep the Five Functional Dimensions of APM Distinct*. [online] Gartner, pp.1-4. Available at: https://www.gartner.com/doc/1436734?ref=g_sitelink [Accessed 19 Nov. 2017].

60 Goldin, P. (2011). *Gartner's 5 Dimensions of APM*. [Blog] APM Digest. Available at: http://www.apmdigest.com/gartners-5-dimensions-of-apm [Accessed 19 Nov. 2017].

61 Xangati (2016). Gartner APM: *5 Key Aspects of Application Performance Monitoring*. [Blog] xangati. Available at: https://www.xangati.com/blog/gartner-apm-5-key-aspects-application-performance-monitoring/ [Accessed 19 Nov. 2017].

62 Kersey, M. (2000). *Scalability vs OOD*. [Blog] Google Groups. Available at: https://groups.google.com/forum/?hl=en#!msg/microsoft.public.inetserver.asp.components/c4G5Ehcr86c/FY9nj5BrhNkJ [Accessed 21 Nov. 2017].

63 Explorable (2009). *Non-Probability Sampling*. [Blog] Explorable. Available at: https://explorable.com/non-probability-sampling [Accessed 30 Nov. 2017].

64 Explorable (2008). *Experimental Research*. [Blog] Explorable. Available at: https://explorable.com/experimental-research [Accessed 30 Nov. 2017].

65 Bass, L., Clements, P. and Kazman, R. (2010). *Software architecture in practice*. Boston, Mass. [u.a.]: Addison-Wesley.

66 Cooper, B. (2015). *An Introduction to Moderated Mediation*.

67 Hayes, A. (2017). *Introduction to Mediation, Moderation, and Conditional Process Analysis, Second Edition*. New York: Guilford Publications.

68 Cirt.gcu.edu. (2017). *Benefits and Limitations of Experimental Research - Center for Innovation in Research and Teaching*. [online] Available at: https://cirt.gcu.edu/research/developmentresources/research_ready/experimental/benefits_limits [Accessed 5 Dec. 2017].

69 Zikmund, W., Babin, B., Carr, J. and Griffin, M. (2013). *Business research methods*. 9th ed. Mason, OH: South-Western, p.486.

70 Stangroom, J. (2017). *Pearson Correlation Coefficient Calculator*. [online] Socscistatistics.com. Available at: http://www.socscistatistics.com/tests/pearson/Default2.aspx [Accessed 21 Dec. 2017].

71 Vogels, W. (2006). A Word on Scalability. [Blog] *All Things Distributed*. Available at: http://www.allthingsdistributed.com/2006/03/a_word_on_scalability.html [Accessed 22 Nov. 2017].