THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

PERFORMANCE EVALUATIONS OF CLOUD COMPUTING PLATFORMS

Master Thesis

GÜLTEKİN ATAŞ

İSTANBUL, 2013

THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING

PERFORMANCE EVALUATIONS OF CLOUD COMPUTING PLATFORMS

Master Thesis

GÜLTEKİN ATAŞ

Supervisor: ASST. PROF. DR. VEHBİ ÇAĞRI GÜNGÖR

İSTANBUL, 2013

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Name of the thesis: Performance evaluations of cloud computing platforms
Name/Last Name of the Student: Gültekin ATAŞ
Date of the Defense of Thesis: June 13, 2013

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. Dr. Tunç BOZBURA
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Tarkan AYDIN
Program Coordinator

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Comittee Members	Signature
Thesis Supervisor Assist. Prof. Dr. Vehbi Çağrı GÜNGÖR	---
Member Assist. Prof. Dr. Mehmet Alper TUNGA	---
Member Assist. Prof. Dr. Tevfik AYTEKİN	---

ABSTRACT

PERFORMANCE EVALUATIONS OF CLOUD COMPUTING PLATFORMS

Gültekin Ataş

Computer Engineering

Thesis Supervisor: Asst. Prof. Dr. Vehbi Çağrı Güngör

June 2013, 75 pages

There are many different types of cloud computing offers in the information technologies market. This study deals with the performance evaluations of three public "*platform as a service*" solutions offered by different companies. Cloud Foundry, Heroku, and OpenShift are the selected platforms for comparison. The market researches show that the cloud computing services will be one of the most demanded services that companies will look for in near future. Thus, companies will want to know which solution is good at which function. Similar to the other solutions, there is no silver bullet in cloud computing. Hence, the comparison must be based on the common things and those things should be the most used or needed resources. Therefore, computing power, database operations, and the main memory bandwidth are tested and compared in this study.

The test results show that each platform performs differently in each of the area tested. A provider may be the best in one of the functions or sub functions of the performance criteria; however, it may be worse or the worst in another. This thesis proposes the most critical functions and sub functions for a PaaS solution. It also proposes a suitable set of benchmarking algorithms and suggests the best provider based on the test results by using two different decision making methods, and by taking the needs of the customer into account. Since the evaluation is based on the performance, the fees are not compared and not considered as a criterion. Similarly, other factors, like user friendliness, supported add-ons and services are nice to have; nevertheless, they are not the basic and common drivers of company needs in general.

The performance results of the three platforms are compared to each other by using Analytic Hierarchy Process (AHP) and Logic Scoring of Preference (LSP). Four different scenarios are presented and the evaluation results and rankings of AHP and LSP are provided for each.

Keywords: Cloud Computing, Platform as a Service, Performance Evaluation, Analytic Hierarchy Process, Logic Scoring of Preference

ÖZET

BULUT BİLİŞİM PLATFORMLARININ PERFORMANS DEĞERLENDİRMESİ

Gültekin Ataş

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Vehbi Çağrı Güngör

Haziran 2013, 75 sayfa

Bilgi teknolojileri piyasasında çok çeşitli bulut bilişim çözümleri sunulmaktadır. Bu çalışma, üç farklı firmanın sunduğu genel kullanıma açık üç farklı "*Hizmet olarak Platform*" çözümünün performans değerlendirmesi üzerinedir. Bu performans değerlendirmesi için seçilen platformlar, Cloud Foundry, Heroku ve OpenShift platformlarıdır. Piyasa araştırmaları göstermektedir ki, bulut bilişim hizmetleri çok yakın bir gelecekte, firmaların en çok talep edeceği bilişim hizmetlerinden birisi olacaktır. Bu sebeple bu hizmeti almak isteyen müşteriler, hangi çözümün hangi konuda daha iyi olduğunu bilmek isteyeceklerdir. Diğer çözümlerde olduğu gibi, bulut bilişimde de her konuda en iyi olan bir çözüm bulunmamaktadır. Bu sebeple karşılaştırma, bazı ortak işlevler üzerinde yapılmalı ve bu ortak işlevler de en çok ihtiyaç duyulan ya da kullanılan işlevler olmalıdır. Bundan dolayı bu çalışmada, işlem gücü, veri tabanı işlemleri ve ana hafıza bant genişliği işlevleri test edilmiş ve birbirleriyle kıyaslanmıştır.

Test sonuçları göstermiştir ki her platform, test edilen her işlevde aynı performansı gösterememektedir. Bir platform, ana işlevlerden veya alt işlevlerden birinde en iyi olabilirken başka bir işlevde en iyi olamamakta hatta en kötü performansı gösterebilmektedir. Bu tez çalışması, "*Hizmet olarak Platform*" çözümleri çerçevesinde belirli işlevleri ve bunların alt işlevlerini en kritik özellikler olarak sunmaktadır. Ek olarak bu özelliklerin ölçülebilmesi için uygun kıyaslama algoritmalarından oluşan bir set ile birlikte, farklı ihtiyaçları da göz önünde tutabilen iki farklı karar verme metodu kullanmak suretiyle en iyi platformu önermektedir. Bu çalışmadaki temel değerlendirme performans odaklı olduğu için firmaların hizmet ücretleri değerlendirmeye dahil edilmemiştir. Benzer şekilde, kullanıcı dostu olup olmadığı, sahip olduğu eklenti veya desteklediği servisler de, genel anlamda bu servislerden faydalanacak firmaların temel ve ortak ihtiyaçları olmasından ziyade, "*olsa iyi olur*" özellikler olarak görülmüş ve değerlendirmeye dahil edilmemiştir.

Üç servis için de performans sonuçları, Analitik Hiyerarşi Süreci (AHP) ve Tercihlerin Mantıksal Puanlaması (LSP) metotları kullanılarak karşılaştırılmıştır. Dört farklı

senaryo önerilmiş ve her biri için AHP ve LSP ile elde edilen sonuç ve sıralamalar sunulmuştur.

Anahtar Kelimeler: Bulut Bilişim, Hizmet Olarak Platform, Performans Değerlendirme, Analitik Hiyerarşi Süreci, Tercihlerin Mantıksal Puanlaması.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AHP	:	Analytic Hierarchy Process
CPU	:	Central Processing Unit
CSMIC	:	Cloud Services Measurement Initiative Consortium
DB	:	Database
EC2	:	Elastic Compute Cloud
ECU	:	Elastic Compute Cloud Computing Unit
GCD	:	Generalized Conjunction / Disjunction
GHz	:	Gigahertz
HTTP	:	Hypertext Transfer Protocol
IaaS	:	Infrastructure as a Service
IT	:	Information Technologies
I/O	:	Input output
JSP	:	Java Server Pages
KB	:	Kilo Bytes
KPI	:	Key Performance Indicators
LSP	:	Logic Scoring of Preference
MB	:	Mega Bytes
MCDM	:	Multiple Criteria Decision Making
MFLOPS	:	Millions of Floating Point Instructions per Second
MOPS	:	Millions of Operations per Second
ms	:	millisecond
MWIPS	:	Millions of Whetstone Instructions per Second
PaaS	:	Platform as a Service
QC	:	Quasi Conjunction

QD	:	Quasi Disconjunction
RSRM	:	Relative Service Ranking Matrix
RSRV	:	Relative Service Ranking Vector
SaaS	:	Software as a Service
SLA	:	Service Level Agreement
TPC-W	:	Transaction Processing Performance Council Web Benchmark
VM	:	Virtual Machine
XaaS	:	Anything as a Service

LIST OF SYMBOLS

Disjunction degree	:	d
Elementary criterion function	:	G_i
Elementary preference	:	E_i
Global preference	:	E
Input preferences of the aggregation block	:	e_i
Microsecond	:	µs
Output preference	:	e_0
Performance variable	:	X_i
System's total aggregation logic function	:	L
Weighted power	:	r
Weights	:	W_i

1. INTRODUCTION

Cloud computing has been one of the most popular concept of the last decade in information technologies. Although the original concept goes back to 1960s, widely used real world applications and commercial usage has begun in last decade.

Cloud computing services are offered in various models and service categories, although the general idea is to provide the infrastructure and/or the services to the customers over the internet. Another important aspect is that the customer must be able to manage it through an online interface without a need of a technical operator on the provider company side. Therefore, the customers can hire necessary resources from the providers and pay fees for the services they used based on the contract type selected. It can be thought as the computing services as a utility. As in the public electricity utility case, the cost of having and maintaining an electric generator to supply the home devices is very high and not feasible or viable for most of the public consumers in economic and operational sense. Instead, the customer makes an agreement with the electricity provider based on the power and the usage.

As a specific type of cloud computing, the Platform as a Service (PaaS) is offered as the software and the hardware infrastructure by the providers. As it is the case in the Infrastructure as a Service (IaaS), the virtual servers and other infrastructure components are also provided in PaaS, even though they cannot be controlled by the customer. Additionally, the operating systems, application servers, development frameworks, necessary libraries and the deployment infrastructure are also offered to the consumers by the PaaS service providers. The PaaS concept is to give the all necessary hardware and software environment necessary to develop and deploy the software for the consumers.

Since the cloud computing is thought as the future's information technologies (IT) solution, companies need tools and methods on how to choose the best and the right solution for themselves. The aim of this thesis is to propose functions and sub functions

for general computing needs of consumers in PaaS and then propose the best provider for the consumers according to the needs specified by them using adjustable weights. For the comparison and ranking of the providers, Analytic Hierarchy Process (AHP) and Logic Scoring of Preference (LSP) methods are used. To show the ranking and score changes according to the needs, 4 different scenarios are provided. AHP and LSP are applied to all four scenarios for the mentioned three providers.

1.1 TYPES OF CLOUD COMPUTING

There is a wide range of cloud computing services. Therefore, several categorizations have been done in the IT sector. The most used and referred classifications are based on the deployment model and the service model. The categorization based on the deployment model is as follows:

i Public Cloud: The cloud is open for general public use. Management of the cloud is done by the provider company.

ii Private Cloud: The cloud is private in terms of its use by single organization with its customers. It may be owned by the consumer or by the provider company. Management is generally done by the company which consumes the cloud services.

iii Community Cloud: The cloud is open for specific consumers of organizations like government, universities, and developers. It stands between public and private cloud in terms of ownership and beneficiary. Management can be distributed among the organizations.

iv Hybrid Cloud: It is a combination of two or more cloud deployment types described above. Management of the cloud is divided between the provider company (public side) and the customer company (private side).

Another categorization is done based on the offered service type. There are many of types in this category. Hence, sometimes this categorization is called anything as a service (XaaS) in general. The offered services in this categorization correspond to the main elements of a general IT system. A typical IT system consists of physical infrastructure, operating systems or server systems working on the infrastructure, and

the business applications working on top of them. Therefore, each of these layers corresponds to the following types of cloud computing below:

i Infrastructure as a service (IaaS): Virtual resources which correspond to hardware basics are given to the consumers. Virtual machines, storage facilities, firewall, network resources and some basic network software like protocols and others.

ii Platform as a service (PaaS): The infrastructure in IaaS is in place already. The consumer gets running operating system, application servers, deployment systems, source code versioning systems, as well as database storage, programming frameworks and libraries.

iii Software as a service (SaaS): The physical infrastructure, running operating systems, necessary databases, and frameworks are in place already. Consumers have also applications already built and running on this structure. They benefit the applications provided to users without programming and maintaining them.

1.2 PAAS

Platform as a service (PaaS) is a cloud service type which provides users the computing platform, necessary frameworks, libraries and servers. Therefore, the consumers do not need to spend time on any job related to install, manage and configure the application and/or database servers, deployment processes, source code control system, storage system and underlying network resources. Providers handle all these administrative and operational burdens. Some providers also offer development, debugging, and testing infrastructure on the cloud platform additionally.

Consumers have control over the configuration and resources of the application although they have no control over the infrastructure as opposed to the IaaS. Hence, the infrastructure is by default given like in IaaS in order to run everything needed over that; however, consumers have no control over the virtual machine resources, network resources, configuration of infrastructure, servers, operating systems running on them.

They can increase the number of worker processes or number of instances of their application and add or remove additional services for their applications.

The security of resources is the responsibility of the providers; nevertheless, consumers should take precautions in order to ensure the programming security. Therefore, consumers must follow the security design rules and test it in development life cycle as they should do in regular development projects.

1.3 SELECTED PAAS PLATFORMS

There are a lot of PaaS options in cloud computing sector. To name some; Amazon Elastic Compute Cloud (Amazon EC2), OpenShift by RedHat, Heroku, Windows Azure, Cloud Foundry, Google App Engine, Force.com, and Mendix. Amazon and Azure are the most studied platforms among these.

The main service of PaaS solutions is providing the frameworks and libraries to developers, to develop and deploy their programs as stated earlier. The database service is an essential part of almost all programs. Therefore, every PaaS solution either includes their database services or uses a third party database service for their customers' application. Most of the PaaS providers give at least MySQL or Postgre database services. Besides, some of them provide database backup utilities and more add-ons to ease the data management.

The main reason for choosing Heroku, OpenShift and Cloud Foundry in this thesis is that they are the PaaS solutions offered by well-known world-wide companies in IT sector. Since they are big players of the IT sector and still investing to the cloud computing, they will probably be among the most popular platforms in future.

1.3.1 Heroku

Heroku has been founded in 2007. The founders were Orion Henry, Adam Wiggins, and James Lindenbaum.[1] Their goal was to create a PaaS which is easy to use and deploy software. They build it on a multi-tenant architecture. Therefore, they would not struggle with the administration of virtual machines and the administration of the platform stacks, and their installations would be easier. On December 2010, it is acquired by Salesfoce.com.

Applications are hosted under their root domains like Heroku.com or herokuapp.com like the other providers. They are run inside of a dyno which is a name given for the basic container unit in Heroku platform. Each dyno is isolated from each other. It has 512 MB memory resource. Hence, to use more memory or to process more requests or to provide redundancy, the number of dynos can be increased. They are hosted in an execution environment which is named as dyno manifold. It is distributed, scalable, and fault tolerant. Therefore, it is the duty of this environment to restart the dynos when a new version of the application is deployed to the cloud, or when configuration changes or when the application crashes. Also, when hardware issues occur, it moves the dynos to other physical servers. The most popular languages or frameworks those supported by Heroku are Ruby, Python, Java, Spring, Play, Django, Rails, Scala, Clojure, and Node.js. It is mainly based on Ruby and it not an open-source as the other two platforms.

Git[2] is used for revision control and source code management which is initially designed and developed by Linus Torvalds. Plug-ins like eGit can be added to Eclipse to automate all deployment processes. If eGit is used with Eclipse IDE, after making the changes to the application, committing the files for change and then pushing them to the

[1] Salesforce.com signs definitive agreement to acquire Heroku. 2010. http://news.heroku.com/news_releases/salesforcecom-signs-definitive-agreement-to-acquire-heroku [accessed 13 January 2013]

[2] Getting Started - A short history of Git. 2013. http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git [accessed 13 January 2013].

upstream is enough to release it to the cloud. The application will be deployed and restarted automatically.

Database services are offered as add-ons in Heroku. Heroku Postgres was selected as the database-as-a-service add-on for the tests. Several visual software clients can be used to connect database. PgAdmin III client was used to connect and manage the test database on Heroku in this thesis work. There are many other database services they offer, like Cloudant Data Layer as a Service, Redis Cloud Beta, MongoHQ, ClearDB MySQL, Amazon RDS, etc. There are many types of add-ons offered; for example, SendGrid e-mailing, Cloudinary image management, openredis hosting add-ons and so on.

1.3.2 OpenShift

OpenShift is the PaaS solution owned by RedHat. They also offer OpenShift Enterprise which can be installed on company's premises as private, public or hybrid cloud. OpenShift platform is an open-sourced platform. OpenShift is first released as a free service on May 2011. They are the first PaaS which supports Java 6 EE.[3]

Git is used for revision control and source code management like in Heroku. The most known languages or frameworks supported by OpenShift are Java, PHP, Ruby, Node.js, Python, and Perl.

The services and application frameworks are offered as cartridges. Consumers can add or remove cartridges easily with small configuration settings. The basic unit of an application in runtime is named as gear. A gear is the name for the resource container for cartridges and application's resources like Central Processing Unit (CPU) power, memory and like.

[3] Issac Roth, 2011. Announcing OpenShift: The Platform-as-a-Service for developers who love open source and CDI. http://cloudcomputing.info/en/news/2011/04/vmware-announces-its-paas-solution-called-cloud-foundry.html [accessed 13 January 2013]

The database services offered by OpenShift are MongoDB NoSQL, MySQL, and PostgreSQL. Management of the database can be handled via the web interfaces (after adding cartridges like RockMongo or phpMyAdmin to the application) or via the client software after enabling tunneling which requires some additional complicated installation and configuration.

1.3.3 Cloud Foundry

Cloud Foundry is the PaaS solution of VMware, Inc. Cloud Foundry is also an open-source cloud platform like OpenShift. It is still offered as beta. Micro Cloud Foundry is another version of their PaaS software which can be run on desktop computers or notebooks and it is the first PaaS in this sense. Cloud Foundry's announcement to public is done on April 2011.[4]

The languages and frameworks supported by Cloud Foundry are Java, Spring for Java, Grails, Node.js, Ruby for Rails and Sinatra, Python, PHP, Scala. The database systems they offer are MySQL, MongoDB, Redis, vFabric PostgreSQL. Besides, they offer RabbitMQ as messaging service for applications. There is no support for managing database via database management client software yet. Users must use console commands to manage and view database contents or use coding in their applications.

1.4 MOTIVATION AND GOALS

There are many companies offering PaaS service today and the number goes up every year due to the demand and trends shown in market researches. According to an article of Olga Kharif (2012) on Bloomberg.com, Market Research Media Ltd states that the

[4] Surksum, K. v., 2011. VMware announces its PaaS solution called Cloud Foundry. http://cloudcomputing.info/en/news/2011/04/vmware-announces-its-paas-solution-called-cloud-foundry.html. [accessed 13 January 2013].

cloud computing market is expected to grow 30 percent every year and in 2020, it will have a share of $270 billion.[5]

Louis Columbus (2012) from Forbes gives numbers of Gartner report. He says that PaaS market was $900M in 2011 and estimated to reach $2.9B in 2016 which means an average of $360M growth each year.[6]

Vivek Kundra (2010, pp. 6-7), the U.S. Chief Information Officer, declares the official "Cloud First" policy. He says that since the cloud is economical, flexible and fast, they decided to take the advantage of cloud. He tells about the case of Car Allowance and Rebate System of Federal Government. 250,000 transactions were expected although the demand went much beyond the estimations and hence they lived several system outages. Therefore, he states that if they implement a cloud solution, they will easily respond to such unexpected fluctuations. Also he says that the flexibility of cloud will give them the ability of adding or removing the hardware and software without much effort. Then, he declares their plan of moving to the cloud. They will require their chief information officers to determine three services in their agencies to migrate to the cloud and prepare their project plan. Their schedule for migrating to cloud should be 18 months at most.

The forecasts and trends are different in numbers; however, it is clear that high rates of growth and demand are expected in near future. Hence, as it is the case for other types of cloud services, the PaaS solution will also become an indispensable option for business of every size. They will have to decide to select the best and the most suitable PaaS for their needs. Nevertheless, it is also very clear for the researchers of the area that there is no PaaS company that is the best at everything. This is because the different needs of the companies those looking for the PaaS services. Therefore, the problem or

[5] Kharif, O., 2012. Kleiner Perkins considering new fund for cloud-computing services startups. *Bloomberg*, http://www.bloomberg.com/news/2012-02-10/kleiner-perkins-considering-new-fund-for-cloud-computing-services-startups.html

[6] Columbus, L., 2012. Cloud computing and enterprise software forecast update. *Forbes.com*, http://www.forbes.com/sites/louiscolumbus/2012/11/08/cloud-computing-and-enterprise-software-forecast-update-2012/

the need is to have a system that helps to evaluate the provided solutions and helps to decide between them. Since the needs are different for each customer and even for each project, the system must break down the functionalities and the performance items and then measure them separately. Finally it must suggest the best solution according to the priorities set by the customer.

The studies about cloud computing are mostly concentrated on IaaS and SaaS. PaaS is not researched in detail as other two cloud types. Key performance items for cloud computing are studied in several works. CPU power and memory bandwidth are important key performance items for PaaS and IaaS; however, they are not studied in decomposed functions or areas. The approach that is observed in the researches is that the CPU power is tested roughly with benchmarks based on floating point calculations. In some of them, response times are tested by using some scientific applications. These applications may be good for IaaS; although, they are not suitable for PaaS service. It is not easy to use and configure them and they are not good candidates for widely adoption in cloud performance researches. Besides, database services are not included in PaaS studies. They are studied separately in the cloud field, mostly as Database as a Service. Since they are not a part of IaaS they are not studied in those studies also. Therefore, it is obvious that an appropriate and comprehensive set of performance items for PaaS is needed. Also suitable benchmarks to measure these performance items must be provided. Several researches have used evaluation methods to compare the performance of cloud offers. However, more studies must employ statistical comparison and ranking methods to get better understanding for reasonable comparison of alternatives.

To address these needs, a framework of necessary PaaS features is proposed. It consists of 19 different functions which are decompositions of 3 main functions. In order to provide a common base for performance evaluation of PaaS solutions, the basic and inevitable functions of applications are selected. These are the computing power, database operations and the memory bandwidth, which are the 3 main functions mentioned above. Every main function is decomposed into sub functions for consumers to project their needs more precisely. For example, computing power is subdivided to 8

sub performance items, like floating-point operations, conditional structures, integer arithmetic, and mathematical functions. This will help companies to adjust the weights according to these sub functions and reflect their application's structure more accurately. Therefore, the model will be more close to the real world application needs. This decomposition has ended up in 19 sub functions. This framework of functions constitutes a good basis for testing and comparison of the PaaS solutions.

Cloud Foundry, Heroku, and OpenShift are the selected commercial PaaS platforms for this study. They have been tested in line with the specified functions. Since there are many performance items in the proposed framework, a statistical method is needed to evaluate the results correctly. The platforms have been compared by two different evaluation methods to select the best. These methods are AHP and LSP. Two different methods are used to reinforce the decision making. The evaluation of each method is performed under four different scenarios. They represent different types of application which have different computing resource needs. They also help to develop the understanding of the performance items' effect on the results and ranking scores. Consequently, the main contributions of this thesis can be summarized as follows:

i A framework of necessary PaaS features is proposed which consists of 3 main functions and 19 sub functions. It contains the most essential features of PaaS solutions and therefore, it provides a strong basis for performance evaluations of PaaS platforms.

ii A suitable benchmark set is provided in parallel to the feature framework. They are necessary to get the performance items' test results. Whetstone algorithm, STREAM algorithm, and a specific set of basic queries are the parts of the suggested benchmarking set.

iii To make the decision making more reliable, two different comparison and ranking methods are applied to the results. The test results are combined under four different scenarios and then, AHP and LSP methods are applied to these test result sets. The scenarios are employed to simulate the different resource needs of applications. To the best of knowledge of this thesis'

writer, the LSP technique is first applied to the selection of PaaS platform problem in this work.

iv Stability of the platforms according to the test results are also discussed and compared to each other.

1.5 CONSTRAINTS

Most of the other functionalities or the add-ons offered by the PaaS providers change and may not be provided by all of them in general. Also the programming language support differs in each provider. Hence, Java is selected as the common language which is supported not only by these providers but by most of the PaaS providers.

Another constraint is the database service offered. Most of them provide MySQL or PostgreSQL service in common; however, the implementations are of course different. Therefore, the database service which is the most encouraged by the provider is selected for each of them while considering it to be a widely used database solution. For example the PostgreSQL implementation of Heroku is named as Heroku Postgres. They also offer MySQL service as ClearDB MySQL. Since PostgreSQL is the most encouraged database service by Heroku and it is a widely used and supported database system, Heroku Postgre is selected for Heroku applications. For OpenShift, the most encouraged database solution was MongoDB NoSQL database; however, since it is not offered widely by other PaaS providers, the second database solution has been chosen which is MySQL 5.1. Following the same principle, the most encouraged database service of the Cloud Foundry was MySQL 5.1. Since it is a widely used database, it is selected for the applications in Cloud Foundry.

Heroku is tested with the basic free evaluation subscription. Heroku names the processing unit as dyno. It is an isolated container having computing resources in its own virtualized environment. Basic subscription has one dyno. It has 512 MB memory and 1x CPU share. The CPU's computing power is not declared to the subscribers or public. There are some claims that it is approximately one micro Amazon EC2 which is at maximum 2 ECUs of Amazon (1 ECU is equivalent to 2007 Xeon processor having

1.0 – 1.2 GHz CPU). Here, ECU is the measurement unit defined by Amazon and it is short for EC2 (Elastic Compute Cloud) Computing Unit. Nevertheless, Heroku does not confirm it.

OpenShift is also tested with the basic free evaluation subscription. OpenShift has named its computing unit as gear. Their basic and free subscription is named small gear and a small gear has 512 MB memory, 1 GB of disk size, 250 threads per small gear. They also do not declare the CPU equivalent processing power.

Cloud Foundry is tested with the basic free developer subscription. Cloud Foundry does not give any detail about the computing resources it uses, since it is still in beta and did not constitute a resource versus cost plan.

The absence of exact and full detail of the resource given to the consumer is a limitation on the comparison researches. Even though they mostly declare the memory resource amount, they do not specify CPU power to compare directly with other platform offers. Also they do not declare any performance figures or metrics about their database service performance. Therefore, they cannot be verified with the performance tests and that makes the benchmark algorithms as the only way to compare the performance figures.

1.6 THESIS ORGANIZATION

In this thesis, three public PaaS platforms are tested and compared by using two different evaluation and ranking methods. A detailed feature framework is proposed and benchmarking algorithms for these features are provided. The thesis is organized as follows:

In section 2, literature research is presented. A background for the cloud computing works, cloud definitions and characteristics are provided here. The studies on performance and comparison of cloud computing alternatives are also discussed in this section.

In section 3, information about the benchmarking algorithms and evaluation methods are presented. Test data is presented in tabular and chart forms. The feature framework is also presented in this section.

In section 4, the evaluation results of the test results are provided. Evaluation methods are applied to test results by using four different scenarios those simulate different resource needs of different types of applications.

In last section, the discussion about the results is provided. The best platform of the three is offered here and other research findings are presented. The thesis work ends with the conclusion sub section.

2. LITERATURE REVIEW

2.1 BACKGROUND

Although the cloud computing has become popular in last decade, the concept is older than the expected, going back to early 1960s. In one of his speech held at MIT in 1961, John McCarthy had foresaw that if the computer technology of 1960s continues to be the mainstream in future, then it is highly probable that it can be offered as a public utility just like the public telephone service. He claims that if that happens, then such an organization of computing will change the industry significantly (Garfinkel and Abelson 1999).

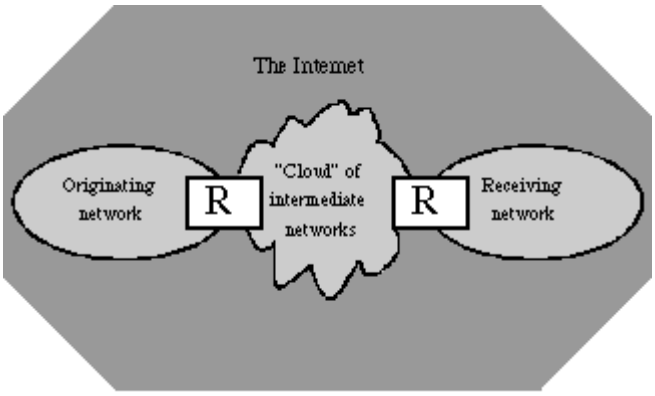
Douglas F. Parkhill (1966) has given several characteristics of cloud computing, similar to the ones provided today. He also was one of the persons who foresaw that the computer industry will evolve such that it becomes a public utility servicing in future.

J.C.R. Licklider (1968), who is very famous in computer history, also described a future which is very much like the internet and cloud computing as we know today. He had admitted that the idea of interconnected computers around the world is not a new idea even in 1960s. He had discussed about a multi-access system and the load that the user put onto such system for the service he requires. He had given even a brief cost details of such a system for an ordinary user and compares it with its alternatives.

There are ongoing discussions about who used the cloud and the cloud computing terms first. The general consensus is that the cloud term is first used by the telecommunication companies. The cloud symbol has been used in network charts to simplify and encapsulate the complex network structures in a single figure for a couple of decades. Many say that the cloud term is started to be used at the beginning of 1990's in telecommunication and ATM networks.

There is a claim that the cloud term is first used publicly in a figure of the research of Gillett and Kapor (1996) from Harvard University. They used a cloud figure and named it as "'Cloud' of intermediate networks" to depict the networks taking place between the destination and the target networks. The figure is shown below in Figure 2.1, where R means the router or gateway between the end points of the networks.

Figure 2.1: The Internet's confederation approach



Source: Sharon Eisner Gillett and Mitchell Kapor, (1996) The Self-governing Internet: Coordination by Design, Harvard University

One of the first usages of the term in academia belongs to Ramnath Chellappa (1997). He claims that the cloud computing is a new paradigm in computing field. He also states that its limits will be determined not by the technical advancements but by the economic reasons.

Another well-known and recent usage of the cloud computing term publicly came from an important figure of the IT sector, Eric Schmidt (2006).[7] The CEO of the Google had used the term with an emphasis on its business side. He also emphasized the close relation of advertising and the cloud computing. He said that the relationship forms a new business model in which consumer can choose the platform, client and database

[7] Schmidt, E., Conversation with Eric Schmidt hosted by Danny Sullivan, *Search Engine Strategies Conference* [online], August 9, 2006, http://www.google.com/press/podium/ses2006.html [accessed 20 January 2013]

solutions. He also had stated that the advertising is pushing advancements in cloud computing.

“Utility computing” and “hosted computing” terms also have been used and are still being used for cloud computing. They emphasize the business model of the computing services; however, they highly imply and also rely on the cloud infrastructure. Armbrust *et al.* (2009, p. 1) distinguish the Public Cloud, which is the cloud computing open for public to consume in pay-as-you-go economic model, and the Utility Computing, which is the name for the consumed service itself. They used the cloud computing term for any services which may be an application, system software or hardware that provided over the Internet.

Cloud computing has evolved from the advancements made in grid computing or distributed computing. They are not substitutes for cloud computing term today; however, they were used in a similar meaning in 1990s. Foster *et al.* (2008) state that the grid computing was the substitute term for the cloud computing in 1990s. It meant to describe the computing service on demand. They also state that even both grid and cloud computing terms are used to express different concepts today; both of them try to switch the paradigm from proprietary computing systems into a system hosted, maintained, and provisioned by third parties to the consumer companies. To do that, they both try to decrease the cost, improve the flexibility and they must be more reliable to meet policies and the standards of the consumers.

Distributed computing is a special type of parallel computing. They are connected nodes of computer networks over private or public connections. They are still being used in scientific and academic world. They are task-oriented and have minimum interaction so that a batch of jobs is submitted and results are obtained and combined to form the final output. The computers are located at different sites which are remarkably away from each other and loosely coupled. In fact, this is one of the architectures which are employed by the cloud computing in the background.

Therefore, the terms described above have been used interchangeably in time but oriented according to different objectives. Wang *et al.* (2010, pp. 141-142) emphasize the differences of cloud computing from other types of computing, like Global computing, Grid computing, and Internet Computing. They state that the cloud computing term implies a system which is managed through user-centric interfaces, lightweight local client software; also it provides on-demand service and offers guaranteed QoS. The system is autonomous (automatically configured, coordinated, and composed), scalable and flexible as well.

The first example of widely used commercial cloud computing service came from Salesforce.com in 1999. They were providing applications over web pages.[8] Amazon Web Service in 2002 followed as the next important example in commercial cloud computing history. With the emergence of Amazon Web Service, consumers could search and display Amazon.com products in their applications. In 2006, Amazon launched S3, their storage service. By using S3 service, consumers can create data objects as big as 5 GB. There was no limit on the number of data objects. Its commercial goal is to provide customers, a simple, fast and inexpensive storage over the web. Google Docs has followed the trend in 2006. It has offered both web-based office applications and storage which is an example of Software as a Service (SaaS).

Microsoft released its public cloud service of Windows Azure in 2008, which was commercially available on 2012. Azure provides both SaaS and IaaS which means that the customer can develop and deploy their applications on the Azure and also they can manage their computing infrastructure according to their needs. Then, many companies entered the cloud sector offering diverse services in the last decade.

2.2 DEFINITION AND CHARACTERISTICS OF CLOUD COMPUTING

There are many definitions of the cloud computing term. Therefore, for brevity, only a few of them will be mentioned here. Mell and Grance (2011, p. 2) define it as a

[8] A complete history of cloud computing, 2013. http://www.salesforce.com/uk/socialsuccess/cloud-computing/the-complete-history-of-cloud-computing.jsp [accessed 13 January 2013]

computing model which allows consumers to get resources with minimum provider interaction and minimum administrative effort. As they state, it must be provisioned online, configurable, provided on-demand, flexible and scalable enough to meet the new requirements when the customer increase or decrease the resources used. Also it must be measurable for customers to get reports on the services they pay. These services span a wide range, like network, server, storage, application, and other services.

Armbrust *et al.* (2009, p. 4), explains the cloud computing term as the services that can be in the form of applications or the hardware or the systems software offered through the internet which are hosted on the datacenters of the providers.

Carr (2008, p. 1) describes the cloud computing as a service model that the consumers can run the software, and store their data over the web. He also characterized it such that the applications and the storage are hosted in the datacenters of the provider companies.

Gong *et al.* (2010, pp. 276-278) suggested similar common characteristics of cloud computing. As they listed, it must be service oriented, loosely coupled, and highly tolerable. The business model is also important which gave rise to the cloud sector. They stated that it should be TCP/IP based which is the main protocol for internet infrastructure. Another important characteristic which they listed is virtualization. Virtualization is one of the key enabling technologies of the cloud computing as we know it today.

Geva Perry (2008), CMO at GigaSpace Technologies, lists the main characteristics of the cloud computing as self-healing (failover), Service Level Agreement (SLA) driven, multi-tenancy (running on the same infrastructure with other services without endangering the privacy), service-oriented, virtualized (separated from the hardware infrastructure), linearly scalable, and data processing.[9]

[9] Perry, G., 2008. How cloud & utility computing are different. http://gigaom.com/2008/02/28/how-cloud-utility-computing-are-different/ [accessed 13 January 2013]

2.3 RELATED WORK

PaaS is relatively a new type of cloud computing service. Therefore, there is not much research on the evaluations of PaaS offers. Any company will expect a good computing power and memory performance from the PaaS service they pay. These expectations are valid also in IaaS services. Hence, researches about the IaaS also will be discussed in this section. Some of the researches focus only on the properties of the selected cloud product. They did not perform any test or present performance comparison. Some other studies have performed tests; however, they have not presented a detailed framework for evaluation. The most of them have not applied a statistical method for evaluation.

Cloud Services Measurement Initiative Consortium (CSMIC) determined the Service Measurement Index (SMI), which is a series of Key Performance Indicators (KPI) related to business.[10] These are general categories like agility, accountability, assurance, financial, security, performance, privacy, and usability. As they explained they are business-relevant indicators; therefore, the performance metrics are accuracy, functionality, suitability, interoperability, and service response time. All of them are related to SLA, functions requested and functions serviced to customers. Thus, there is no metrics like the megabytes per second (MB/sec) for memory bandwidth or the Millions of Floating Point Instructions per Second (MFLOPS) for processing power. However, these metrics are being used for a long time in performance evaluations of a wide variety of computers from personal computers to the supercomputer grids. Hence, there are more fine grained metrics for each of these sub topics not officially listed in that work.

There are some works which compare cloud services according to their feature lists. In the work of Peng *et al.* (2009, pp. 23-27), 4 cloud platforms (AbiCloud, Eucalyptus, Nimbus, and OpenNebula) have been listed and their main properties are discussed. They have compared these platforms according to their properties, like deployment type, scalability, virtualization support, operating system support, compatibility, and so

[10] Cloud Services Measurement Initiative Consortium, Service measurement index, version 1.0, 2011, http://www.cloudcommons.com/documents/10508/186d5f13-f40e-47ad-b9a6-4f246cf7e34f.

on in a table. Sempolinski and Thain (2010) also studied Eucalyptus, OpenNebula, and Nimbus cloud platforms. They have also followed a similar structure; first they describe the general structure of the cloud architecture and then listed the properties of the mentioned platforms, like disk image options, storage, virtualization technology, customizability, security, network structure, etc. They discussed the main blocks of the platforms and their components. The comparison is done basically on the property list and the general structure. No performance test or performance comparison is presented.

Voras *et al.* (2011) stated that they have devised approximately 100 criteria. They listed the main categories as storage, virtualization, network, management, security and vendor support. They pointed out that the criteria create a baseline comparison for IaaS solutions. They just listed these sub criteria in their work. They also briefly mentioned about several cloud platform, like Abiquo, Eucalyptus, mOSAIC, Nimbus, OpenNebula, etc. They said that the evaluation project is ongoing; nevertheless, they did not give any detail about how they proceed or what kind of methodology they are applying. They explained that assigning weights to the areas of special interest will enable high-level decision making. There is no clue about this also; however, it sounds like the AHP methodology which is also employed in this thesis work. The work did not mention about any test or performance evaluation.

Another type of property comparison has been done in the work of Wind (2011). Abicloud, Eucalyptus, OpenNebula and Nimbus are selected as the cloud platforms. The platforms are compared in a tabular form according to 13 features, like architecture, programming language, deployment model, virtualization technology, data or VM memory infrastructure, area of application, user interface, operation system license, fault tolerance, interoperability, security, VM management client and compatibility with public cloud providers. Recommendations have been given for each platform for different business needs based on this table. However, neither the performance is detailed in technical functions nor any tests have been performed.

Armbrust *et al.* (2009, pp. 16-17) have discussed the obstacles and the opportunities of the cloud computing in their work. They discuss about the Amazon, Microsoft Azure

and Google AppEngine, as each one of them being an example of different service type of cloud computing. Since their work is focused on the advantages and disadvantages of the cloud computing, they have not performed a comparative performance evaluation. They reported STREAM benchmark tests on 75 virtual machines (VM) of Amazon EC2. They have measured an average value of 1355 MB per second for memory bandwidth. The performance test had been done almost 5 years ago; therefore, the numbers are improved today. In this thesis work, it changes between 3,000 to 20,000 MB/sec for the studied platforms. Besides, they did not report test results or any comparison for other providers. They studied the feasibility and the cost of cloud solutions in their work. They discussed the regular costs of IT department like power, cooling, physical plant cost, and operational costs and compared it to the cost of cloud services.

Binnig *et al.* (2009, pp. 3-4) discussed the Transaction Processing Performance Council Web benchmark (TPC-W), which is an online bookstore consists of several web interactions. They supported the idea that a cloud benchmark should be web based and define interactions like TPC-W benchmark. However, they criticized it about the lack of adequate metrics for scalability, pay-per-use, and fault tolerance. The writer of this thesis agrees with the idea of web based solutions, since it is very suitable for evaluating the performance of different PaaS solutions without installing more complex applications like installing applications to back-end servers and using Java beans or services. It is the main motivation for using Java codes of Whetstone and Stream benchmark that can be triggered by requesting Java Server Pages (JSP) via Hypertext Transfer Protocol (HTTP) in this thesis. The real life workload simulation of the TPC-W benchmark is appreciable; however, it has a very strict architecture which cannot measure the CPU, memory and database operations separately and cannot show which resource limits the overall performance in detail. Therefore, the applied method in this thesis is able to address the different types of performance evaluation needs, since the testing of each resource is decoupled from others. Also sub properties (like conditional jump performance in CPU performance or memory bandwidth performance in small data) of each resource can be evaluated and may be opted for another. This is a very

flexible approach which can be used when deciding to choose the platform for different types of applications.

Iosup *et al.* (2011) have investigated Amazon EC2, GoGrid, ElasticHosts, and Mosso as IaaS providers. They have mainly tested the raw infrastructure properties, like resource allocation such that virtual machine allocation, booting and releasing time. They have submitted bags-of-tasks to simulate high volumes of tasks in Many-Task Scientific Computing. They have used LM bench suite to test the floating point operations. They also have used some parts of HPCC and STREAM suites to compare the results of clusters in evaluation of the performance of CPU and random access to memory. In this thesis, computing power has been broken down in several functionalities and test results are given separately for each of them. Computing performance has been tested not just for floating point operations but also for different programming structures, like if-then-else loops, procedure calls, assignments, fixed point operations, floating point operations, trigonometric functions, exponential and other logarithmic operations. To divide computing power abilities into such decomposed functions and evaluating them by assigning different values according to the needs, gives highly modular and flexible comparison results of different platforms.

Salah *et al.* (2011, pp. 347-349), studied and compared the performance of Amazon EC2, ElasticHosts, and BlueLock as IaaS solutions in their works. They have tested CPU, RAM and disk input output (I/O) performance of the three cloud solution. They have used Simplex for CPU benchmarking, STREAM for memory benchmarking and the FIO for disk I/O benchmarking. They have reported that the best memory result is obtained from BlueLock with a score of 7,460 MB per second. In this thesis the best performance observed is over 20,000 MB/sec which is over 2 times of that value. BlueLock also has the best score for the Simplex CPU performance in their work. Amazon EC2 has been reported the best disk I/O performance result. They have compared IaaS solutions in three main categories; nevertheless, they have not provided a comparison model to decide the best solution overall. Also they have not broken down the performance categories into specific areas as this thesis work does.

Tudoran *et al.* (2012, pp 4-6), evaluated and compared Microsoft Azure platform as a public cloud with Nimbus installed on Grid5000 as a private cloud. They have used A-Brain as a real-life application for evaluation of computation power performance. It is an application for genetic and neuro-imaging. They also reported that they have used synthetic benchmarks to evaluate data transfers and network efficiency. Besides, they transfer files for downloading and uploading I/O performance. As they state, their work considers all basic needs of scientific applications (computation power, storage and data transfers). That is primarily where this thesis differs from their work. Their goal was to evaluate the cloud platforms according to the needs of scientific applications. However, this thesis studies the performance of PaaS applications which may be but not necessarily scientific applications. Another difference is that they reported the CPU and memory as computation power in a combined form and in terms of time for the application execution. In this thesis, computing power and memory performance have been distinguished from each other to be able to evaluate platforms in more detail. Although they criticize that the Linpack benchmark results do not always reflect the performance of application, their CPU and memory tests times are not very explanatory and comparable for other works. Also since they have intended to evaluate IaaS properties, they did not test database functions. They carried out a detailed test although they did not use a method to compare them.

Costa and Cruz (2012) has studied the Azure platform. They compared the performance of a web site instance running on Azure with the performance of another instance running on a local server and with a third instance running on a commercial web hosting company. They have used the web site of a vocational school as the test case or scenario. This case reminds the TPC-W benchmark used in the work of Binnig *et al* (2009). However here, it is a more specific and limited application than TPC-W and there is no detail about the functionalities and the performance requirements of the web site. Therefore, they did not offer a comparable result or methodology for other PaaS platforms.

Garg *et al.* (2013), provides a set of KPIs to compare the cloud services. They used the CSMIC's framework and developed in several aspects. They suggested a user weighted

model for qualitative and quantitative features. They also employed the AHP for numerical indicators to get relative scoring. Same methodology has been used in other works like the work of Tran *et al.* (2009) about ranking the web services. The performance comparison of this thesis also employs AHP and its weight-based evaluation. For alternative services, a set of comparison metrics is proposed which are cost, agility, performance, accountability, assurance, and security. Nevertheless, performance metrics are not studied in decomposed fashion. Response time is the only performance metric used for the tests. This work is more concentrated on the other functional features rather than the performance.

Yu and Molina (2007) applied a modified version of LSP method for evaluation and selection process of web services. They used it to dynamically select the web service based on the XML configuration file supplied as the input to their application. They have given an online payment service scenario. A fictitious company has the choice of selecting one of the 4 different payment services in that scenario. According to the device that customer used to connect to the e-commerce site, the credit card type entered, and the country of user, their application uses LSP method to match or rank the best service and the web site uses that service. Yu and Reiff-Marganiec (2008) also studied dynamically selection of web service employing LSP in their software. They combined the Ordered Weighted Averaging operators with LSP technique.

Although all these works constitute valuable foundations in cloud computing, a comprehensive and detailed performance function set for PaaS alternatives is not addressed. Also corresponding set of benchmark is needed for that set. Another missing issue is that the database performance is not studied together with CPU and memory. Also very few of them applied the statistical methods to compare and evaluate the results.

3. DATA AND METHODOLOGY

3.1 DATA COLLECTION

The data have been collected by testing the performance in different days and at different times of the day. In each test, all three platforms tested at the same time intervals. The only difference in the codes for any of the PaaS functions tested was the connection string and the connection driver in establishing database connections. Same code is implemented for all three providers; therefore, the writer of this thesis believes that an objective measurement of performance is realized.

The 19 sub functions, grouped in 3 main categories, are tested for each of the PaaS providers. The first main category is the computing power measured with Whetstone algorithm. The second category is the database operation times measured with basic read, update, insert, and delete operations for single and multiple records. Object relational mapping tools are not used to eliminate additional performance burden. The third one is the memory bandwidth measured with STREAM algorithm. All functions' code is deployed as JSP applications.

3.2 THE PROPOSED FRAMEWORK FOR FUNCTIONS

The proposed performance criteria framework to test the alternatives and select the best solution has 3 main functions and 19 sub functions as mentioned before. The proposed structure of these functions is as follows:

i CPU Performance
   - a Floating point with array elements
   - b Floating point with array as parameter
   - c Conditional jumps (if then else)
   - d Integer arithmetic (fixed point)
   - e Trigonometric functions (sin, cos etc.)

f Procedural calls with floating point

g Array reference assignments

h Mathematical functions (exponential, square root, logarithm operations etc.)

ii Database Performance

a Single Record

1 Read

2 Update

3 Insert

4 Delete

b Multiple Records

1 Read

2 Update

3 Insert

4 Delete

iii Memory Performance

a Small size

b Medium Size

c Large Size

The metrics used for these functions are as follows:

i MFLOPS: It is the abbreviation for Millions of Floating Point Instructions per Second. It is the number of floating point operations performed by the CPU in one second. Most of the CPU benchmark only measures this floating point calculation performance. This metric is used for Whetstone benchmark results of floating point with array elements, floating point array as parameter, and array reference assignment category results.

ii MOPS: It is the abbreviation for Millions of Operations per Second. It is used for non floating point operations. In general, only floating point operations are measured and this type of operations is not measured. For example in this work, it is used to measure conditional jumps, fixed point

(integer) operations, trigonometric calculations, procedural calls, and mathematical operations like square root, or exponential calculations.

iii Response time: For each function of CPU performance, the Whetstone algorithm uses a routine to complete or a problem to solve when it calculates the MFLOPS/MOPS metrics mentioned above. The response time is the time to complete this routine or problem. This metric is used to show test results in charts; however, since it is closely related to the MFLOPS/MOPS metrics, this is not used in AHP and LSP evaluation methods discussed in following sections. The MFLOPS/MOPS are preferred since the problem of the algorithm may not be interested or adopted in general.

iv Average total operation time for single record (millisecond): For each basic database operations, like read, update, insert, delete, it consist the time passed for connection creation, statement creation, and operation execution. For this metric, the queries used have affected only one single record.

v Average total operation time for 100 records at once (millisecond): Similar to the previous metric, this one also consists the time passed for connection creation, statement creation, and operation execution. However, for this metric, any of the queries used have affected 100 records.

vi Memory bandwidth (MB/sec): It measures the data transfer to and from memory. 3 different sizes of file are used to measure the memory performance under different conditions. These file or memory object sizes are 128 KB, 1 MB, and 100 MB.

3.3 BENCHMARKING

There are several benchmarking concepts in the IT sector for decades. For computing power performance benchmarking, the Whetstone benchmarking algorithm have been used which is adopted by the IT sector leaders for several decades. The Java code adopted from Roy Longbottom's web page.[11] Some changes had been implemented in these tests to use it in JSP. This benchmark is a synthetic benchmark which tries to

[11] Longbottom, R., Roy Longbottom's PC benchmark collection, http://www.roylongbottom.org.uk

solve several different types of mathematical functions. Whetstone benchmarking checks the performance of 8 different computing structures which are also listed in previous section: Floating point with array elements, floating point with arrays as parameters, conditional jumps, fixed point operations, trigonometric functions, floating point operations with procedure calls, assignment operations, and other mathematical functions.

For database operation performance benchmarking, the basic select, update, delete and insert queries are coded by the writer of this thesis. For each of these query types, two different test sets had been performed. In the first query type, each query's target was a single record, and the query had been repeated 100 times in a loop to increase the number of tests. In each loop; the connection, statement and result set are released and created again. In the second type, each query's target was 100 records, and the connection, statement and result set are released and created for each of these queries also. These queries are repeated 20 times in a loop. It is not looped 100 times in order to prevent the time out. The basic commands used are *DriverManager.getConnection()*, *connection.createStatement(),* and *statement.executeQuery()* of *java.sql* package. The test database has single table with 5 columns containing varchar and integer type fields. It is an example table holding fictitious employee data. Each record in the table has approximately 150 bytes.

For main memory performance benchmarking, STREAM benchmark algorithm is used. STREAM benchmark is created by John McCalphin. He claims that the algorithm measures the sustainable real world bandwidth as real users faced, not the theoretical peak values. It is a widely known and applied algorithm in memory bandwidth performance tests. The Java code of the STREAM algorithm is taken from the Virginia University web sites.[12] The code has been modified in order to work with JSP. STREAM is a synthetic benchmark which means that it creates synthetic workloads not real world workloads. The benchmark measures the bandwidth performance by four long vector or matrix operations. "Copy" operation copies or assigns a vector to another with the equal operator. "Scale" operation multiplies a vector element with a scalar

[12] McCalphin, J., The Stream benchmark Java code, http://www.cs.virginia.edu/stream/FTP/Contrib/Java/STREAM.java

value and assigns the result to another vector. “Add” operation adds two vectors and assigns the result to another vector. Lastly, “Triad” operation multiplies a vector with a scalar value, then, adds another vector to this result and then assigns the overall result to another vector. The memory bandwidth tests are performed under 3 sub categories: Small, medium and large data size. For small size, 128 KB of data is tested. 1 MB and 100 MB of data are used to test the performance of medium and large size respectively.

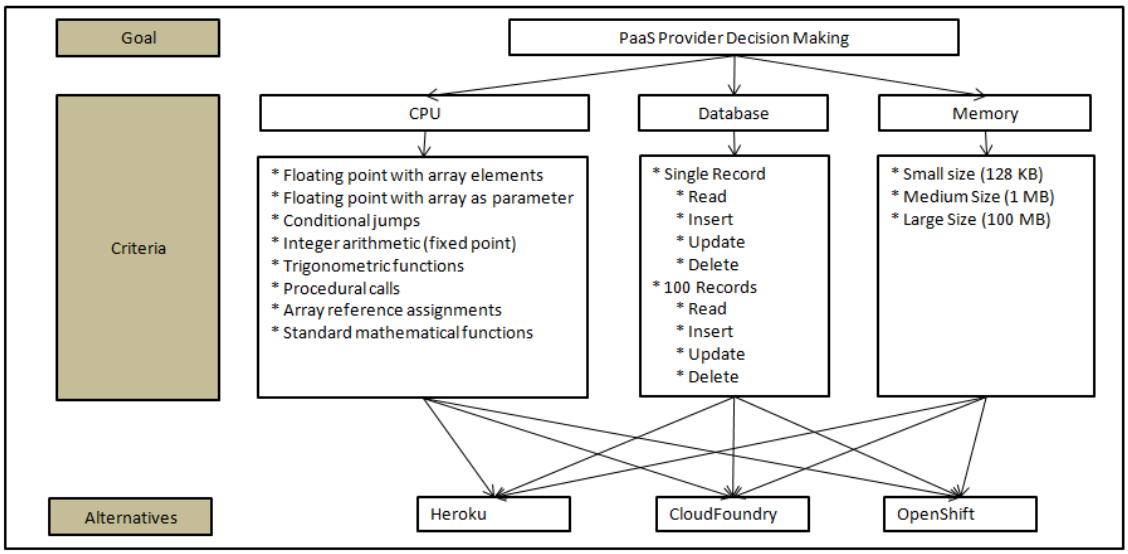
3.4 EVALUATION METHODOLOGIES

3.4.1 Analytic Hierarchy Process

There are 3 main performance functions and 19 sub functions to be evaluated according to the needs of the customer in this thesis’s proposal. Therefore, the problem of deciding which platform performs better is known as Multiple Criteria Decision Making (MCDM). There are many mechanisms for ranking performance indicators of such problems. The first of the two methods employed in this thesis is the AHP mechanism to solve this MCDM problem. It is first proposed by Saaty (1980). The reason to select this method is that it has been widely used in many different research areas as well as computer sciences. It also suits the problem of this thesis work, since there should not be any dependency between the criteria, which is the subject of Analytic Network Process. AHP assigns weights to the items being compared; hence, every performance item can be fine tuned to match the needs.

AHP model first splits the problem into parts structured hierarchically as its name implies. These components are goal, criteria, and alternatives. The AHP hierarchy model of the problem of this thesis is shown in Figure 3.1. In this thesis the goal is obviously to select the best PaaS provider. The main criteria are CPU, database (DB) and memory performances. Their sub criteria are also shown in the figure. The provider alternatives are shown at the bottom of the figure.

Figure 3.1: AHP model of the PaaS provider decision making problem



AHP method uses pair wise comparison and eigenvector method to calculate the relative ranking. The functions and sub functions criteria are accompanied with weights and therefore it becomes flexible. AHP will be employed along with Relative Service Ranking Vectors (RSRV) for performance evaluation. This method is derived from the work of Garg *et al.* (2013, pp. 1018-1020).

Firstly, relative values are calculated for sub function of test results. For CPU and memory bandwidth categories, higher is better; thus, for each provider, the test result of the provider is divided by other providers result respectively. However, for database tests, the time for each operation is measured; therefore, the lower is better. For that reason, the division is reversed, i.e., the other providers' test values are divided by the provider's test results.

Let $P(p_1,\ldots,p_n)$ be the set of PaaS providers, where n is the number of providers. Let $X(x_1,\ldots,x_k)$ be the set of performance items' test values, where k is the number of performance items, e.g. like computing power, memory, or their sub functions such that floating point performance, or copying big sized file for memory. Let $W(w_1,\ldots w_k)$ be the set of the weights determined by the user. The value of performance item's test result of a provider relative to another one will be:

$$RSRM_k = \begin{bmatrix} x_1/x_1 & x_1/x_2 & \dots & x_1/x_n \\ x_2/x_1 & x_2/x_2 & \dots & x_2/x_n \\ \dots & \dots & \dots & \dots \\ x_n/x_1 & x_n/x_2 & \dots & x_n/x_n \end{bmatrix} \quad (3.1)$$

Let $\boldsymbol{V_k^i}$ be the Eigen vector for the *i*'th square of the Relative Service Ranking Matrix (RSRM) matrix for the *k*'th performance item. RSRM matrix is squared successively. The RSRV matrix is obtained when the difference of two successive Eigen vectors, i.e. $\boldsymbol{V_k^i}$ and $\boldsymbol{V_k^{i-1}}$, is negligible (Equation 3.2a and 3.2.b). This process repeated for each performance item in a node. Then, all of the RSRVs of items in a node are combined as columns of the RSRM vector of that node (Equation 3.2c). After that, the RSRV of that node is calculated as multiplying the RSRM of the node by the weight vector for the items in that node, i.e. W_k. Weight vector is shown in Equation 3.2d and the resulting RSRV of the node is shown in Equation 3.2e. This procedure of calculating RSRV of items, then constituting the RSRM of the node and then calculating the RSRV of the node is repeated until the overall RSRV of the system is found.

$$RSRV_k = \boldsymbol{V_k^i},\ \boldsymbol{V_k^i - V_k^{i-1}} \simeq 0 \quad (3.2a)$$

$$RSRV_k = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} \quad (3.2b)$$

$$RSRM_{node} = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1k} \\ v_{21} & v_{22} & \dots & v_{2k} \\ \dots & \dots & \dots & \dots \\ v_{n1} & v_{n2} & \dots & v_{nk} \end{bmatrix} \quad (3.2c)$$

$$W_{node} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_k \end{bmatrix} \quad (3.2d)$$

$$RSRV_{node} = RSRM_{node} \otimes W_{node} \quad (3.2e)$$

Calculation process is explained with the memory bandwidth example with the weights shown in Table 4.1 of the next section. This scenario represents a general enterprise

application. In such applications, main concern is CPU, then database operations come and the memory bandwidth is not required to exceed 1 MB in general. The memory bandwidth of the 3 platforms is shown below in Table 3.1 in concise form. The Copy, Scale, Add, and Triad test results of Table 3.6 are averaged here.

Table 3.1: Average system bandwidth in MB/sec

	Platforms		
Test Size	**Heroku**	**OpenShift**	**Cloud Foundry**
128 KB	19,369.39	4,263.70	22,406.34
1 MB	16,531.87	4,462.98	16,270.09
100 MB	7,562.35	3,730.08	5,810.96

The calculation of the RSRV is as follows. In the first step, the RSRM is calculated as relative values of each provider according to others. The first row of the matrix shown in Equation 3.3a below depicts the relative test value of Heroku for 128 KB memory bandwidth. Heroku's 128 KB test result is first divided to itself in the first column of the first row, to get the relative value of it with respect to itself. Then, in the second column of the first row, the value of Heroku's test result is divided by the OpenShift's test result to find Heroku's relative value with respect to OpenShift. Then, in the third column of the first row, to find the relative value of the Heroku with respect to Cloud Foundry, Heroku's test result is divided by the Cloud Foundry's test result. The relative values end up in Equation 3.3b:

$$RSRM_{memory\ (128\ KB)} = \begin{bmatrix} 19{,}369/19{,}369 & 19{,}369/4{,}264 & 19{,}369/22{,}406 \\ 4{,}264/19{,}369 & 4{,}264/4{,}264 & 4{,}264/22{,}406 \\ 22{,}406/19{,}369 & 22{,}406/4{,}264 & 22{,}406/22{,}406 \end{bmatrix} \quad (3.3a)$$

$$RSRM_{memory\ (128\ KB)} = \begin{bmatrix} 1 & 4.54286 & 0.86446 \\ 0.22013 & 1 & 0.19029 \\ 1.15679 & 5.25514 & 1 \end{bmatrix} \quad (3.3b)$$

Then, Eigen vector of this vector is calculated repeatedly, until the difference of consecutive Eigen vectors becomes too small to ignore. The Eigen vector of this matrix is shown in Equation 3.4. This is the RSRV for the 128 KB memory bandwidth:

$$RSRV_{memory\ (128\ KB)} = \begin{matrix} 0.42071 & 0.09261 & 0.48668 \end{matrix} \quad (3.4)$$

Same procedure is repeated for 1 MB and 100 MB test results. Combining the RSRV of 128 KB, 1 MB, and 100 MB in each column, gives the RSRM of memory bandwidth in Equation 3.5:

$$RSRM_{memory} = \begin{bmatrix} 0.42071 & 0.44363 & 0.44215 \\ 0.09261 & 0.11976 & 0.21809 \\ 0.48668 & 0.43661 & 0.33975 \end{bmatrix} \quad (3.5)$$

Then the overall memory bandwidth RSRV is obtained by multiplying the RSRM and the weights of 128 KB, 1 MB, and 100 MB as shown in Equation 3.6a and 3.6b:

$$RSRV_{memory} = \begin{bmatrix} 0.42071 & 0.44363 & 0.44215 \\ 0.09261 & 0.11976 & 0.21809 \\ 0.48668 & 0.43661 & 0.33975 \end{bmatrix} \otimes \begin{bmatrix} 0.84 \\ 0.15 \\ 0.01 \end{bmatrix} \quad (3.6a)$$

$$RSRV_{memory} = \begin{matrix} 0.42437 & 0.09794 & 0.47770 \end{matrix} \quad (3.6b)$$

For CPU, the MFLOPS/MOPS values are selected as the measurement values. Same procedure is repeated for sub functions of CPU to get the Eigen vectors of each sub function, and then the RSRM for the CPU is obtained by combining the RSRVs. Then, the RSRM of the CPU is multiplied with the weight vector of sub functions, and the RSRV of CPU is found as in Equation 3.7:

$$RSRV_{CPU} = \begin{matrix} 0.35907 & 0.18824 & 0.45269 \end{matrix} \quad (3.7)$$

Lastly, for database operations, the total execution time is found by adding connection creation time, statement creation time and operation execution time. An average is found for each provider and for each database function (read, update, insert, delete). The same procedure repeated twice for database operations due to a second sub level

classification. Database operations are first divided as single record operations and 100 records operations, and then both are divided again for read, update, insert, and delete operations. That means, first 2 RSRMs are calculated, and then 2 RSRVs obtained from these RSRMs and weights (one for single operations and the other for 100 records operations). The RSRVs are combined to get RSRM for the overall database operations. Finally the RSRV is obtained from this last RSRM and the weights. The RSRV is shown in Equation 3.8a and Equation 3.8b:

$$RSRV_{DB} = \begin{bmatrix} 0.08775 & 0.47079 & 0.44146 \\ 0.08150 & 0.44088 & 0.47762 \end{bmatrix} \otimes \begin{bmatrix} 0.80 \\ 0.20 \end{bmatrix} \quad (3.8a)$$

$$RSRV_{DB} = \begin{bmatrix} 0.08650 & 0.46481 & 0.44869 \end{bmatrix} \quad (3.8b)$$

Finally, since the RSRV's of each main function is calculated, the RSRM of the overall performance can be found by combining them. After RSRM is found, the RSRM and the main function weight vector must be multiplied to find the RSRV as shown in Equation 3.9a, 3.9b, and 3.9c:

$$RSRM_{overall} = \begin{bmatrix} 0.35907 & 0.08650 & 0.42437 \\ 0.18824 & 0.46481 & 0.09794 \\ 0.45269 & 0.44869 & 0.47770 \end{bmatrix} \quad (3.9a)$$

$$RSRV_{overall} = \begin{bmatrix} 0.35907 & 0.08650 & 0.42437 \\ 0.18824 & 0.46481 & 0.09794 \\ 0.45269 & 0.44869 & 0.47770 \end{bmatrix} \otimes \begin{bmatrix} 0.50 \\ 0.30 \\ 0.20 \end{bmatrix} \quad (3.9b)$$

$$RSRV_{overall} = \begin{bmatrix} 0.29036 & 0.25315 & 0.45649 \end{bmatrix} \quad (3.9c)$$

Final RSRV scores are adjusted as the higher is the better in this work. Therefore, this last RSRV shows that if the customer's needs are parallel to the weights given in the Table 4.1, then, the Cloud Foundry has the first rank with a score of 0.45649, followed by the Heroku with a score of 0.29036 and OpenShift has the third rank with a score of 0.25315.

3.4.2 Logic Scoring of Preferences

LSP method was first introduced by Dujmovic (1996) in his work of selection of hardware. It is a generalized and extended version of different scoring methods. It has its roots in continuous preference logic. First, each performance variable is decomposed until they cannot be decomposed any more or each item can be measured and evaluated by itself. For this thesis work, these performance variables and their decomposition structure are given in the proposed framework which is presented in sub section 3.2.

Then, for each performance variable, X_i *(i = 1,…n)*, an acceptable range of value is defined, i.e. X_{min}, X_{max} and the values between them. The elementary preference, E_i, indicates the ratio of the satisfaction of the variable according to the needs. Therefore, the value of performance variable X_i should be converted to the E_i to be ordered for each system being compared. This mapping function is called elementary criterion function G_i. It is preferred to be as a piecewise linear function for simplicity. Thus, elementary preference is calculated as in Equation 3.10:

$$E_i = \begin{cases} 0, & X_i \leq X_{min} \\ G_i(X_i), & X_{min} < X_i < X_{max} \\ 100\%, & X_i \geq X_{max} \end{cases} \tag{3.10}$$

Therefore, the elementary preference value will be *0 <= E_i <= 100%*. The maximum and minimum points are also referred as cut-off points. The elementary criterion function can be selected as a preference scale as shown in Equation 3.11:

$$G_i(X_i) = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{3.11}$$

The preference scale for the performance items are show in Table 3.2 below. These values are subjective to the requirements of the company and/or project. For CPU and memory, the higher is better and for database the lower is better.

Table 3.2: Scales for elementary preference scores

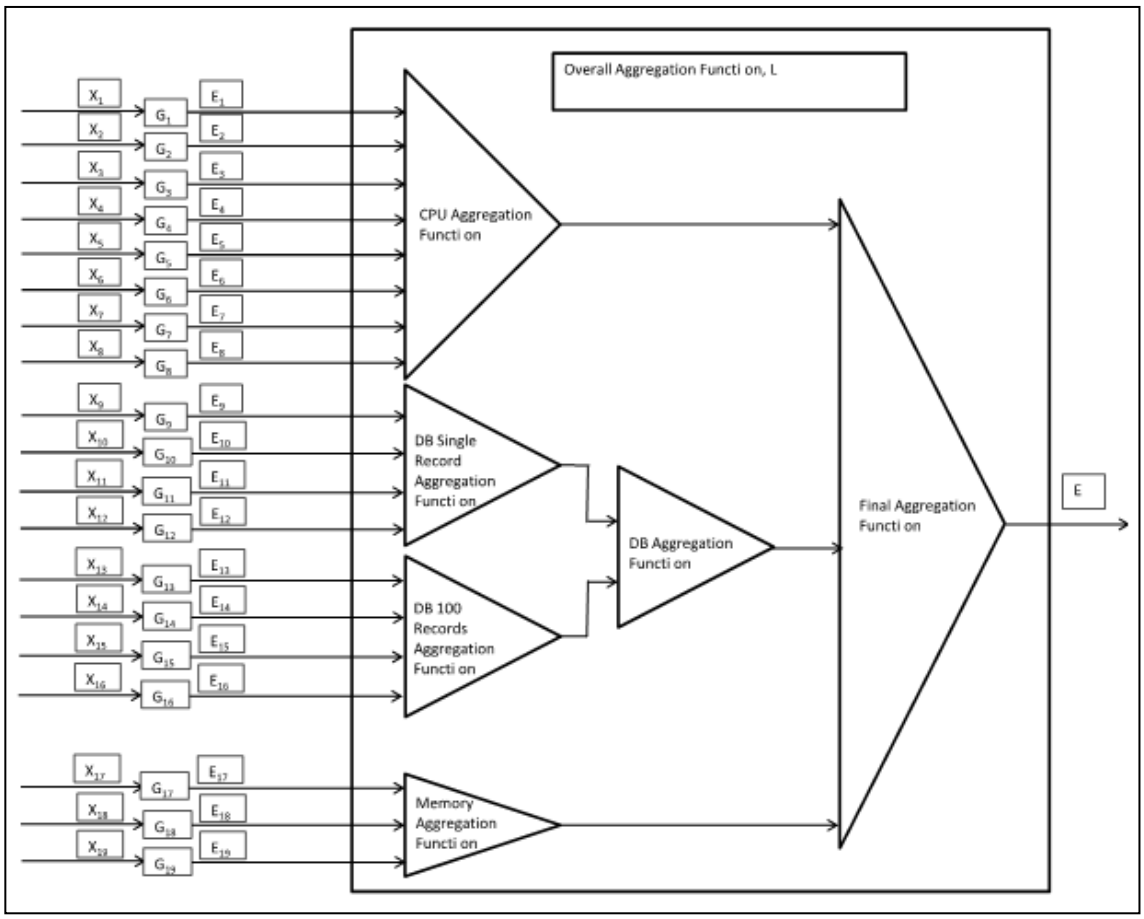
Test Item	Minimum Accepted Value	Maximum Accepted Value
CPU (MFLOPS/MOPS)	**MFLOPS/MOPS**	**MFLOPS/MOPS**
Floating point with array elements	100	1,500
Floating point with array as parameter	100	1,500
Conditional jumps (if then else)	100	1,500
Integer arithmetic (fixed point)	500	2,500
Trigonometric functions (sin, cos etc.)	10	60
Procedural calls	100	1,000
Array reference assignments	200	1,000
Standard mathematical functions (exp, sqrt etc.)	10	60
DB (µs)	**µs**	**µs**
Single Record Read	40,000	2,000
Single Record Update	40,000	2,000
Single Record Insert	40,000	2,000
Single Record Delete	40,000	2,000
100 Records Read	80,000	5,000
100 Records Update	80,000	5,000
100 Records Insert	80,000	5,000
100 Records Delete	80,000	5,000
Memory (MB/sec)	**MB/sec**	**MB/sec**
Small size (128 KB)	2,000	30,000
Medium Size (1 MB)	2,000	25,000
Large Size (100 MB)	2,000	10,000

Therefore, the elementary preference values are calculated by using the test values, Equation 3.11 and the scales listed in Table 3.2. By using these *n* elementary preferences for *n* performance variables, the global preference, *E*, is calculated using a stepwise aggregation technique. This will be a function of all preferences as shown in Equation 3.12:

$$E = L(E_1, \dots, E_n) \tag{3.12}$$

The aggregation function is employed for each sub performance item. Then, another aggregation function is employed to calculate the preference of the categories. The aggregation continues until the preference of the top node is obtained. Hence, the function L represents the aggregation of all aggregations. The overall aggregation scheme of the system is shown in Figure 3.2.

Figure 3.2: System's LSP aggregation scheme



The function indicated as L can be modeled as weighted power mean of these preference values. The power is chosen appropriately to reflect the relations of inputs. Assuming that $e_1, \ldots, e_k$ are input preferences of the aggregation block and $W_1, \ldots, W_k$ are weights which depicts the relative significance of these inputs, the output preference, e_0, will be:

$$e_0 = (W_1 e_1^r + \ldots + W_k e_k^r)^{\frac{1}{r}} \qquad (3.13)$$
$$W_1 + \ldots + W_k = 1,\ W_i > 0,\ i = 1, \ldots, k$$

Here the power *r* is a real number. It must be chosen accordingly such that it indicates the logical structure of the aggregation function. It is a function of *d*, disjunction degree. Here, *d* depicts the average location of e_0, which is between the maximum and minimum of the given interval. *d* should be between 0 and 1. When it is equal to 0, the preference score results in minimum value, i.e. $e_0 = e_{min}$. When *d* equals to 1, the preference score results in maximum value, i.e. $e_0 = e_{max}$. Therefore, the relation between *r* and *d* can be shown as follows: $r = \rho(d)$.

Here ρ is a complex function of *d* and for special values of *d*, the weighting power *r* can reflect the different relations of the performance items. When *r* is -∞, the weighted power mean becomes a pure conjunction. The function is also known as the minimum function. When it is -1, it becomes harmonic mean. When it is equal to zero, that means the geometric mean. The value of 1 makes the relation a square mean. At the opposite side of the spectrum, the +∞ makes the function the pure disjunction. It becomes a maximum function. Hence, the preference aggregation function will be:

$$e_0 = \left(W_1 e_1^{\rho(d)} + \ldots + W_k e_k^{\rho(d)}\right)^{\frac{1}{\rho(d)}} \qquad (3.14)$$

The Equation 3.14 is known as the *generalized conjunction / disjunction* (GCD) and called as *andor* also. Usually the *d* is not calculated, it is already calculated for special cases of the logical function. The names of these special cases, their symbols and the values of *r* are given in Table 3.3. For example when *d* is between 0 and 0.5, it is called as quasi-conjunction. It reflects a logical model when "simultaneity" for the inputs is required. When the system cannot meet the input requirements at the same time, the score is decreased. For *d* between 0 and 0.25, the simultaneity becomes more important and for *d* bigger than 0.25 and close to 0.5, the simultaneity requirement is relaxed.

When *d* is between 0.5 and 1, the logic of *andor* is known as quasi-disjunction. It is used to model the cases when the "replaceability" of inputs is required and it decreases the score of systems which cannot satisfy any of the input criteria. When *d* is between 0.75 and 1, the inputs can compensate or replace each other's score more, and when *d* is between 0.5 and 0.75, it becomes hard for the inputs to replace or compensate each other.

Table 3.3: Logical functions and parameters for aggregation functions

Operation	Symbol	d	r2	r3	r4	r5
DISJUNCTION	D	1.0000	+infinity	+infinity	+infinity	+infinity
STRONG QD (+)	D++	0.9375	20.6300	24.3000	27.1100	30.0900
STRONG QD	D+	0.8750	9.5210	11.0950	12.2700	13.2350
STRONG QD (-)	D+-	0.8125	5.8020	6.6750	7.3160	7.8190
MEDIUM QD	DA	0.7500	3.9290	4.4500	4.8250	5.1110
WEAK QD (+)	D-+	0.6875	2.7920	3.1010	3.3180	3.4790
WEAK QD	D-	0.6250	2.0180	2.1870	2.3020	2.3840
SQUARE MEAN	SQU	0.6232	2.0000			
WEAK QD (-)	D--	0.5625	1.4490	1.5190	1.5650	1.5960
ARITHMETIC MEAN	A	0.5000	1.0000	1.0000	1.0000	1.0000
WEAK QC (-)	C--	0.4375	0.6190	0.5730	0.5460	0.5260
WEAK QC	C-	0.3750	0.2610	0.1920	0.1530	0.1290
GEOMETRIC MEAN	GEO	0.3333	0.0000			
WEAK QC (+)	C-+	0.3125	-0.1480	-0.2080	-0.2350	-0.2510
MEDIUM QC	CA	0.2500	-0.7200	-0.7320	-0.7210	-0.7070
HARMONIC MEAN	HAR	0.2274	-1.0000			
STRONG QC (-)	C+-	0.1875	-1.6550	-1.5500	-1.4550	-1.3800
STRONG QC	C+	0.1250	-3.5100	-3.1140	-2.8230	-2.6060
STRONG QC (+)	C++	0.0625	-9.0600	-7.6390	-6.6890	-6.0130
CONJUNCTION	C	0.0000	-infinity	-infinity	-infinity	-infinity

Source: Jozo J. Dujmovic, (1996) *A Method for Evaluation and Selection of Complex Hardware and Software Systems*. The 22nd International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems

For the aggregation function of CPU parameters, the medium QC (CA) operation is selected from the Table 3.3. For the aggregation sub functions of database, (they are read, update, insert, and delete functions and named as "DB Single Record Aggregation

Function" and "DB 100 Records Aggregation Function" in Figure 3.2), strong QC-(C+-) is selected. For the aggregation function of database overall (which are single record and 100 records nodes and their aggregation function is named as "DB Aggregation Function" in Figure 3.2), and memory functions, the weak QC + (C-+) operation is selected. Finally, for the aggregation function of overall system logic, the strong QC (C+) operation is selected.

3.5 TEST DATA

3.5.1 Whetstone Computing Power Test Results

The test results are shown in tabular format below. They are also represented with charts following the tables of each test group to be easily compared. In Table 3.4, Whetstone test results are shown. The standard deviation is also shown in addition to average of each metric. Result column of the table shows the result of the function solved. Every test on the table is repeated for 355 times. Charts of this table are shown in charts by grouping similar metrics together in each one.

Following the Table 3.4, Figure 3.3 shows the MFLOPS results for N1 floating point operations with array elements, N2 floating operations with arrays as parameters, N6 floating point operations with procedure calls and Millions of Whetstone Instructions per Second (MWIPS). The notation of N1, N2, and so on, are the numbers given to the sub functions to refer them in concise form.

In Figure 3.4, Whetstone time results for N1 floating point operations with array elements, N2 floating operations with arrays as parameters, N6 floating point operations with procedure calls and MWIPS are shown in milliseconds.

Figure 3.5 shows the Whetstone MOPS results for N3 if then else operations, N4 fixed point (integer) operations, N5 Sin, Cos, and other trigonometric operations, N7 assignment operations, and N8 exponent, square root, and logarithmic operations.

Figure 3.6 shows the Whetstone time results for N3 if then else operations, N4 fixed point (integer) operations, N5 Sin, Cos, and other trigonometric operations, N7 assignment operations, and N8 exponent, square root, and logarithmic operations in milliseconds.

Table 3.4: Whetstone test results

			Average				Standard Deviation			
Platform	**Test**	**Test Repeated**	**Result**	**MFLOPS**	**MOPS**	**ms**	**Result**	**MFLOPS**	**MOPS**	**ms**
Heroku	N1 floating pt	355	-1.124750128	840.19		0.0237	0.000000056	116.16		0.0065
	N2 floating pt	355	-1.131330481	824.58		0.1714	0.000000056	115.70		0.0923
	N3 if then else	355	1.000000000		820.55	0.1363	0.000000000		113.42	0.0953
	N4 fixed point	355	12.000000000		1,718.42	0.1890	0.000000000		206.63	0.0565
	N5 sin,cos etc.	355	0.499110132		47.49	1.7973	0.000000000		6.00	0.3898
	N6 floating pt	355	0.999999821	496.63		1.1775	0.000000000	127.95		0.4310
	N7 assignments	355	3.000000000		622.61	0.3193	0.000000000		111.83	0.1848
	N8 exp,sqrt etc.	355	0.827011271		32.79	1.1676	0.014227144		3.78	0.3059
	MWIPS	355		2,063.97		4.9821		289.43		1.0421
Open Shift	N1 floating pt	355	-1.124750137	392.86		0.0511	0.000000000	62.29		0.0159
	N2 floating pt	355	-1.131330490	378.88		0.3679	0.000000000	54.76		0.1045
	N3 if then else	355	1.000000000		421.21	0.2587	0.000000000		61.90	0.1358
	N4 fixed pt	355	12.000000000		1,057.20	0.3100	0.000000000		153.29	0.0995
	N5 sin,cos etc.	355	0.499110132		29.97	2.8956	0.000000002		4.60	0.9263
	N6 floating pt	355	0.999999821	207.20		2.6876	0.000000000	30.26		0.6159
	N7 assignments	355	3.000000000		417.46	0.4568	0.000000000		57.70	0.1112
	N8 exp,sqrt etc.	355	0.850606839		18.98	2.0184	0.046517637		2.73	0.4251
	MWIPS	355		1,129.74		9.0461		143.53		1.5995
Cloud Foundry	N1 floating pt	355	-1.124750099	1,034.17		0.0186	0.000000120	2.79		0.0001
	N2 floating pt	355	-1.131330452	1,007.99		0.1334	0.000000120	13.21		0.0018
	N3 if then else	355	1.000000000		967.17	0.1070	0.000000000		6.08	0.0007
	N4 fixed pt	355	12.000000000		2,095.55	0.1503	0.000000000		11.85	0.0009
	N5 sin,cos etc.	355	0.499110132		53.74	1.5483	0.000000000		0.29	0.0085
	N6 floating pt	355	0.999999821	727.18		0.7427	0.000000000	25.11		0.0282
	N7 assignments	355	3.000000000		773.83	0.2397	0.000000000		41.71	0.0164
	N8 exp,sqrt etc.	355	0.931638988		37.35	0.9959	0.019946517		0.09	0.0024
	MWIPS	355		2,540.96		3.9359		25.78		0.0410

Figure 3.3: Whetstone results for float point operations and MWIPS

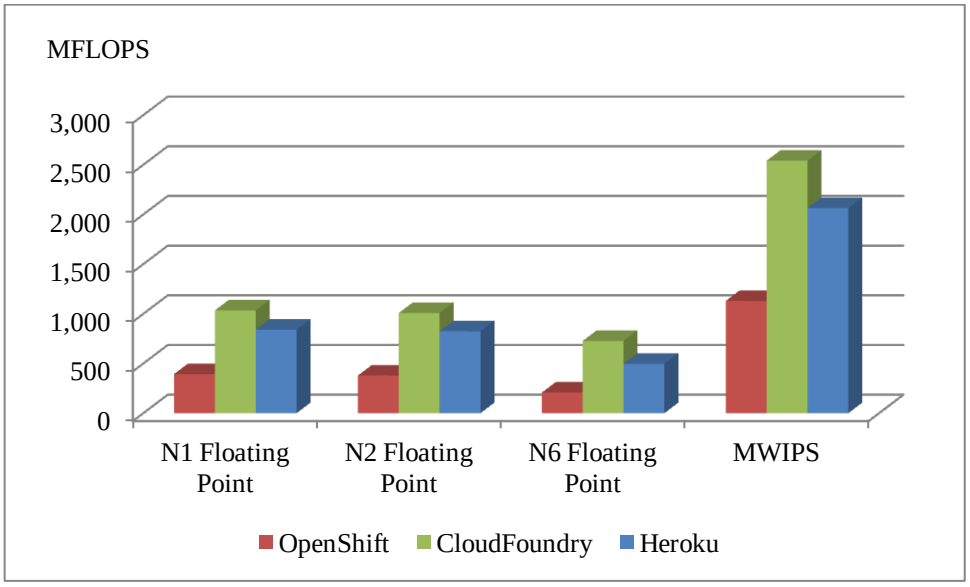


Figure 3.4: Whetstone time results for float point operations and MWIPS

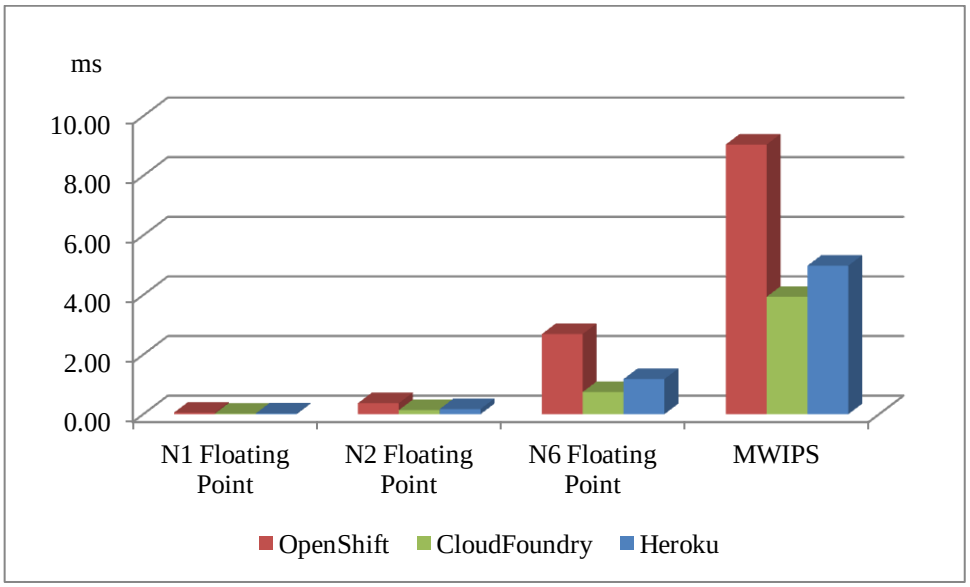


Figure 3.5: Whetstone MOPS results for other operations

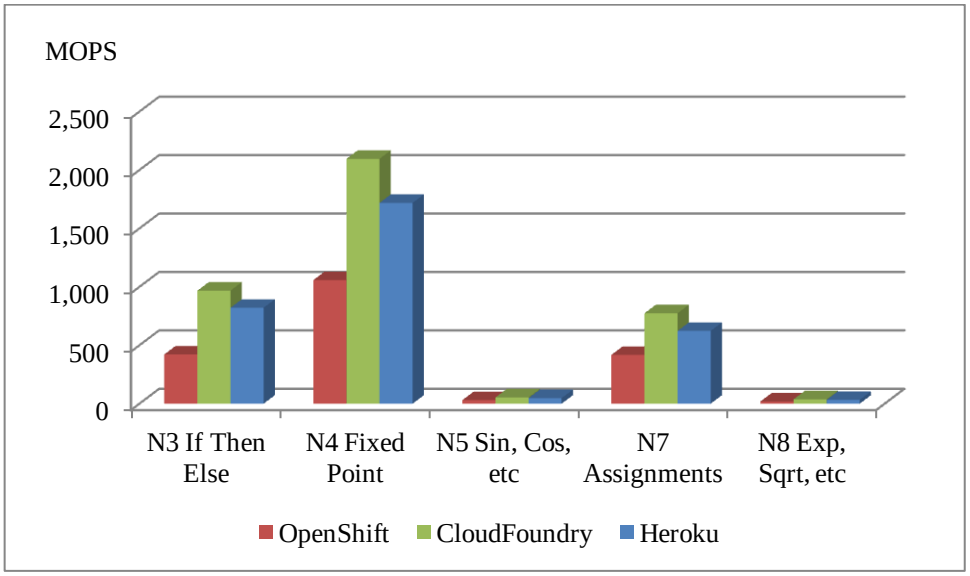
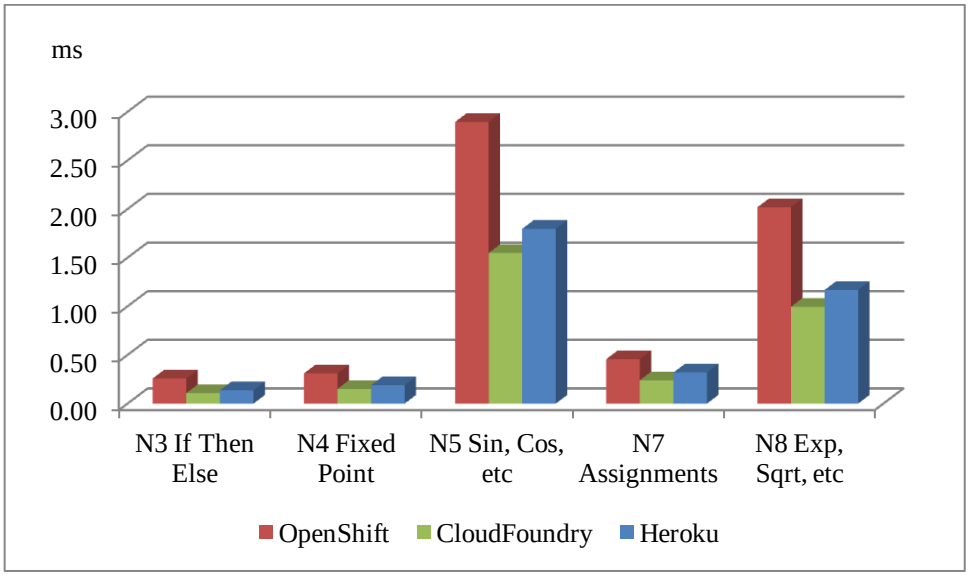


Figure 3.6: Whetstone time results for other results



3.5.2 Database Operations Test Results

The test results performed with both single records and multiple records (100 records at a time) are shown in tabular form. Single record tests are done by querying one single record and repeated for 100 in a loop as explained before. “100 at once query” means that 100 records are affected in each query. The connection, statement, and result set objects are created before each query and they are destructed after the query executed. Therefore, these sub operations are also measured for each operation and query. In Table 3.5, average and standard deviation times for connection creation, statement creation and operation execution are shown separately for each test. High standard deviations show that there are big fluctuations in observed times.

In Figure 3.7, average connection creation times for single record query are shown in microseconds. In Figure 3.8, average statement creation times for single record query are shown in microseconds also. In Figure 3.9, average operation execution times for single record query are similarly shown in microseconds. Figure 3.10 shows the average total operation times for single record query. This total operation times are the sum of connection creation, statement creation, and operation execution times. Figure 3.11 shows the average connection creation times for queries affecting 100 records at once in microseconds. Figure 3.12 shows, similarly, the average statement creation times for queries affecting 100 records at once in microseconds. Figure 3.13 shows the average operation execution times for queries affecting 100 records at once in microseconds also. Figure 3.14 shows the total times, i.e. the sum of connection creation, statement creation, and operation execution, for queries affecting 100 records at once.

Table 3.5: Database operations test results

Platform	DB Operation	Record Count in Operation	Test Repeated	Average Time (Microsecond)			Standard Deviation (Microsecond)		
				Connection Created	Statement Created	Operation Executed	Connection Created	Statement Created	Operation Executed
Heroku	Delete	Single Record	1,400	24,948.61	122.10	8,433.11	15,085.73	1,307.90	5,010.36
OpenShift	Delete	Single Record	1,400	6,219.03	46.03	430.13	8,927.24	241.50	752.96
CloudFoundry	Delete	Single Record	1,400	5,978.04	35.58	448.10	5,507.60	13.58	806.46
Heroku	Insert	Single Record	1,400	25,387.23	73.79	6,073.55	13,736.26	368.26	7,355.59
OpenShift	Insert	Single Record	1,400	5,946.09	38.32	495.79	4,041.85	126.85	1,754.28
CloudFoundry	Insert	Single Record	1,400	6,754.51	36.55	449.43	10,229.84	12.68	858.16
Heroku	Read	Single Record	1,400	30,390.79	105.72	7,596.52	19,869.24	648.60	6,489.29
OpenShift	Read	Single Record	1,400	5,845.10	54.02	432.43	3,834.69	392.55	951.58
CloudFoundry	Read	Single Record	1,400	6,395.39	37.44	597.05	6,286.20	11.49	984.97
Heroku	Update	Single Record	1,400	25,806.83	108.63	8,954.19	13,504.15	941.34	5,612.99
OpenShift	Update	Single Record	1,400	6,220.27	61.26	440.41	3,979.35	494.55	632.64
CloudFoundry	Update	Single Record	1,400	5,962.65	36.32	486.44	5,960.93	10.18	939.77
Heroku	Delete	100 Records At Once	1,100	28,427.27	100.82	29,301.15	23,114.61	711.29	15,096.09
OpenShift	Delete	100 Records At Once	1,100	6,262.37	22.07	4,955.99	6,244.27	8.78	8,644.18
CloudFoundry	Delete	100 Records At Once	1,100	5,712.32	31.60	1,358.60	5,759.31	10.67	841.03
Heroku	Insert	100 Records At Once	1,100	28,468.88	71.10	15,429.79	18,490.14	345.00	10,066.26
OpenShift	Insert	100 Records At Once	1,100	6,696.49	23.93	5,309.92	3,961.37	20.70	4,990.49
CloudFoundry	Insert	100 Records At Once	1,100	6,789.07	34.97	1,908.98	6,546.65	41.53	1,351.57
Heroku	Read	100 Records At Once	1,100	27,060.20	147.26	26,147.27	18,023.69	2,679.14	17,420.71
OpenShift	Read	100 Records At Once	1,100	6,261.68	27.07	2,174.31	4,472.67	19.83	1,466.23
CloudFoundry	Read	100 Records At Once	1,100	5,832.00	33.08	4,959.43	6,024.79	9.73	11,679.82
Heroku	Update	100 Records At Once	1,100	28,633.21	73.98	30,433.18	20,856.70	357.24	19,210.89
OpenShift	Update	100 Records At Once	1,100	5,861.03	36.03	1,599.54	7,097.35	355.76	1,557.89
CloudFoundry	Update	100 Records At Once	1,100	5,360.29	32.56	800.52	4,949.69	9.38	712.30

Figure 3.7: Average connection creation times for single record query

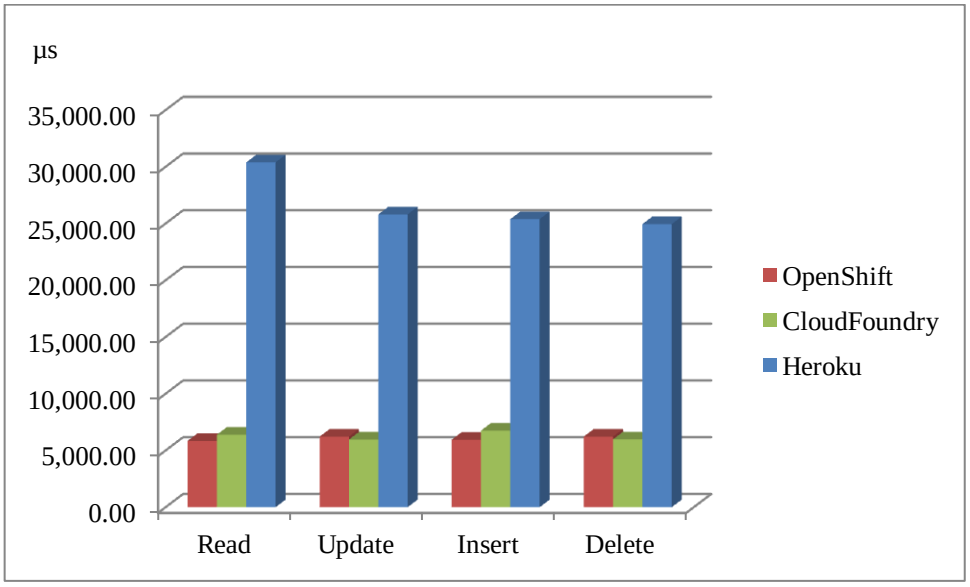


Figure 3.8: Average statement creation times for single record query

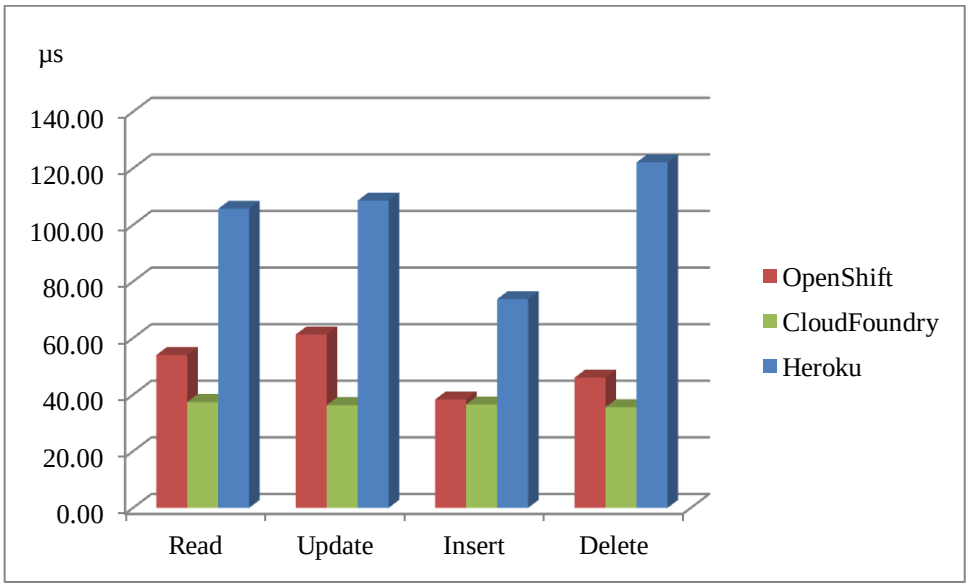


Figure 3.9: Average operation execution times for single record query

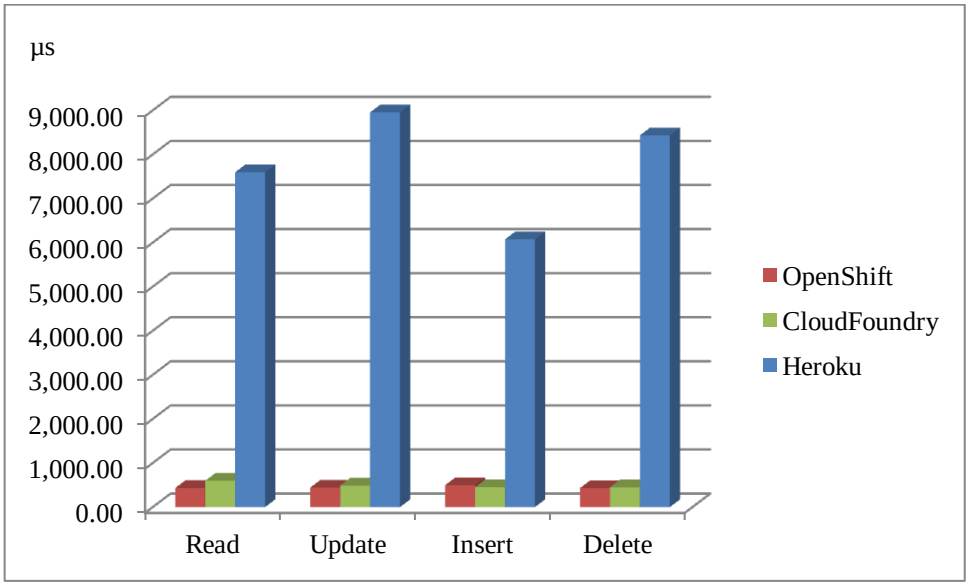


Figure 3.10: Average total operation time for single record query

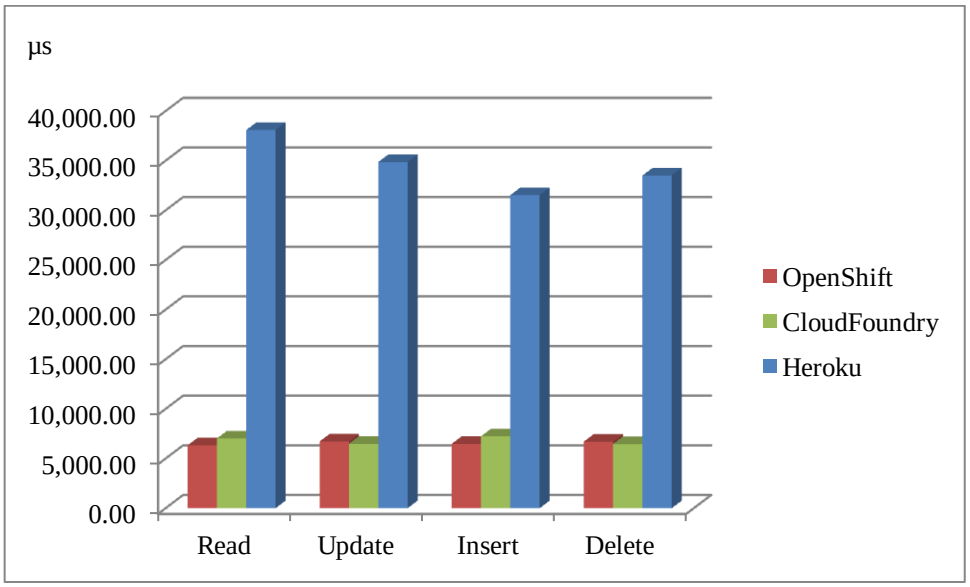


Figure 3.11: Average connection creation times for 100 records at once

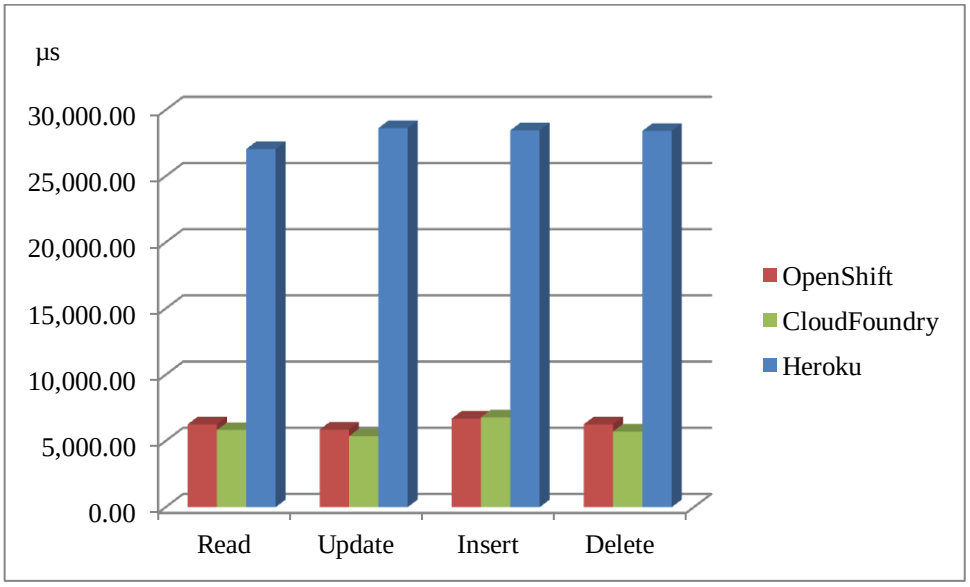


Figure 3.12: Average statement creation times for 100 records at once query

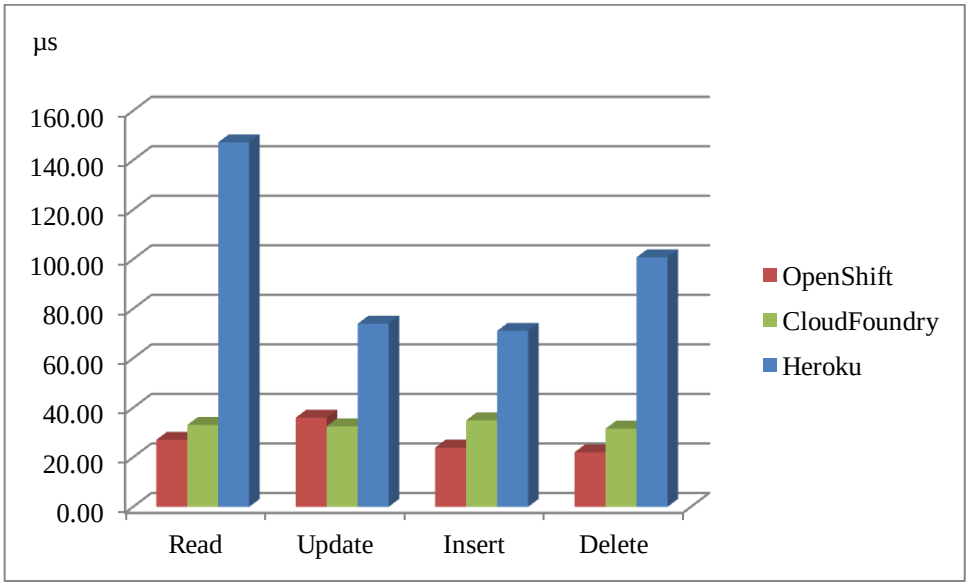


Figure 3.13: Average operation execution times for 100 records at once

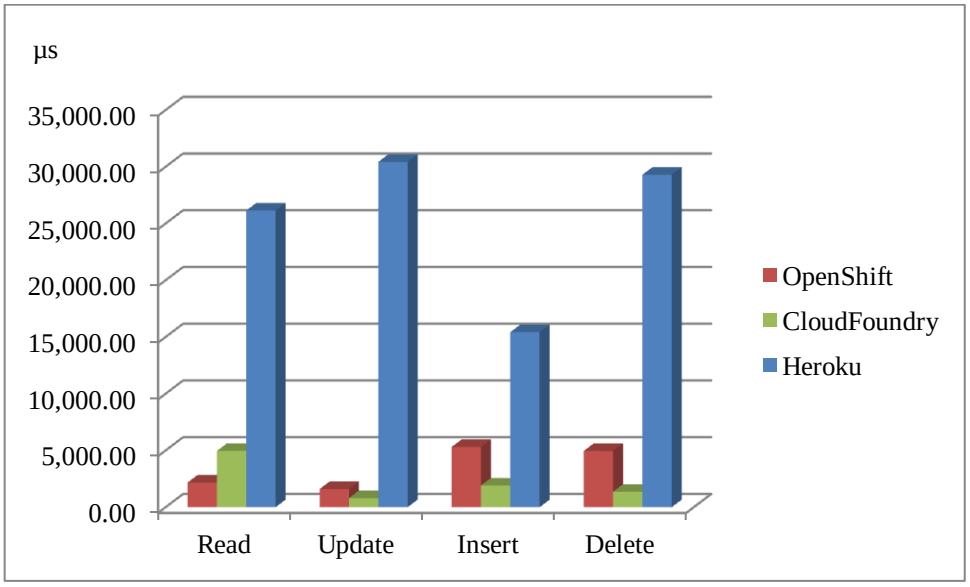
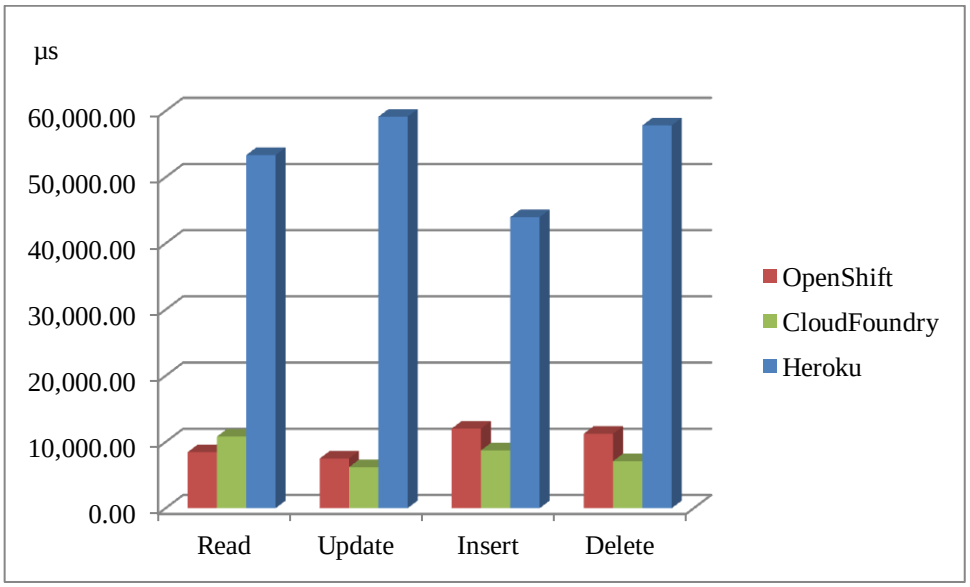


Figure 3.14: Average total operation time for 100 records at once query



3.5.3 STREAM Memory Bandwidth Test Results

The memory tests are performed with 128 KB, 1 MB, and 100 MB workloads by accordingly adjusting the N (the size of the array). The memory workload is calculated in Stream algorithm as in Equation 3.15 where "Bytes per word" is 8 for Java language:

$$\text{Total Memory Required} = ((3.0 * \text{Bytes per Word}) * (N / 1048576.0)) \quad (3.15)$$

In Table 3.6, Stream algorithm test results are shown. 128 KB and 1 MB tests have been performed 1,000 times by looping each test 20 times. However, 100 MB tests have not been performed by looping since the time out threshold is exceeded when looping for such a big workload. Therefore, tests for 100 MB workload have been performed 300 times which is less than the previous two workloads. In all cases, the more bandwidth is better. In this benchmark algorithm, the bandwidth is measured for 4 different types of operations, i.e. copy, scale, add, triad.

Figure 3.15 shows the comparison of bandwidth test results for 128 KB workload for the three platforms. Figure 3.16 shows comparison of memory bandwidth test results for 1 MB workload and Figure 3.17 shows the bandwidth test results for 100 MB workload for the three PaaS platforms.

Table 3.6: STREAM test results

			Average				Standard Deviation			
			Total system bandwidth MB/sec				Total system bandwidth MB/sec			
Platform	**Test Size**	**Test Repeated**	**Copy**	**Scale**	**Add**	**Triad**	**Copy**	**Scale**	**Add**	**Triad**
Heroku	128 KB	1,000	20,990.60	16,175.54	21,970.42	18,341.01	2,236.92	1,343.29	2,226.69	2,350.84
OpenShift	128 KB	1,000	3,989.52	3,987.63	4,747.72	4,329.92	55.63	54.03	66.58	65.44
CloudFoundry	128 KB	1,000	25,012.67	19,317.68	25,265.38	20,029.62	1,863.45	1,415.00	2,100.91	1,581.45
Heroku	1 MB	1,000	17,128.36	14,667.87	17,797.68	16,533.56	1,211.51	1,023.71	1,263.84	1,147.33
OpenShift	1 MB	1,000	4,202.86	4,223.52	4,938.96	4,486.57	55.69	51.29	63.64	63.95
CloudFoundry	1 MB	1,000	17,869.00	14,166.80	17,566.93	15,477.63	1,245.73	936.00	1,155.43	1,017.84
Heroku	100 MB	300	7,600.76	7,204.52	7,889.11	7,555.00	1,375.79	1,451.93	1,723.26	1,704.77
OpenShift	100 MB	300	3,388.11	3,666.76	4,069.52	3,795.94	281.40	265.18	329.65	291.01
CloudFoundry	100 MB	300	6,567.68	5,178.13	6,122.11	5,375.93	364.26	272.55	337.07	320.50

Figure 3.15: STREAM bandwidth test results for 128 MB

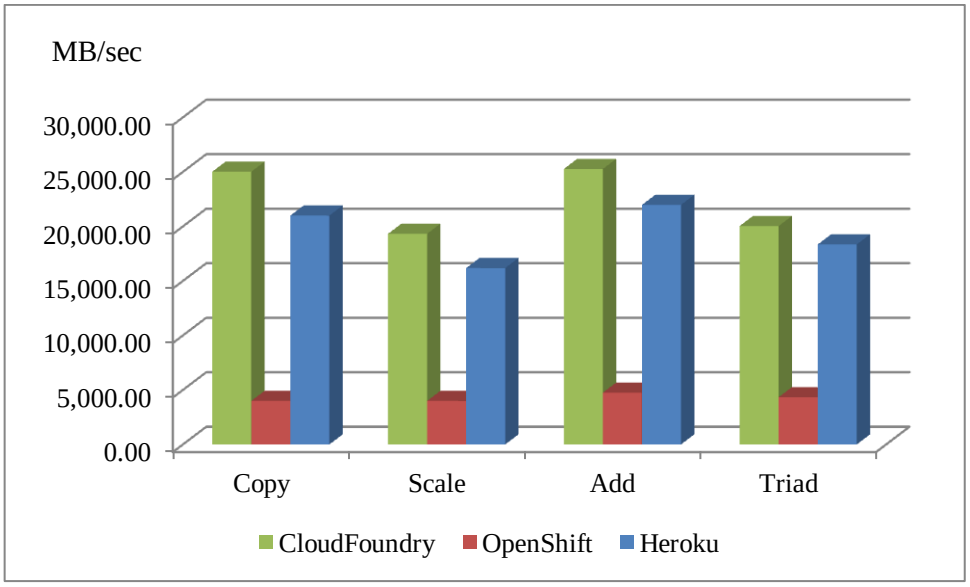


Figure 3.16: STREAM bandwidth test results for 1 MB

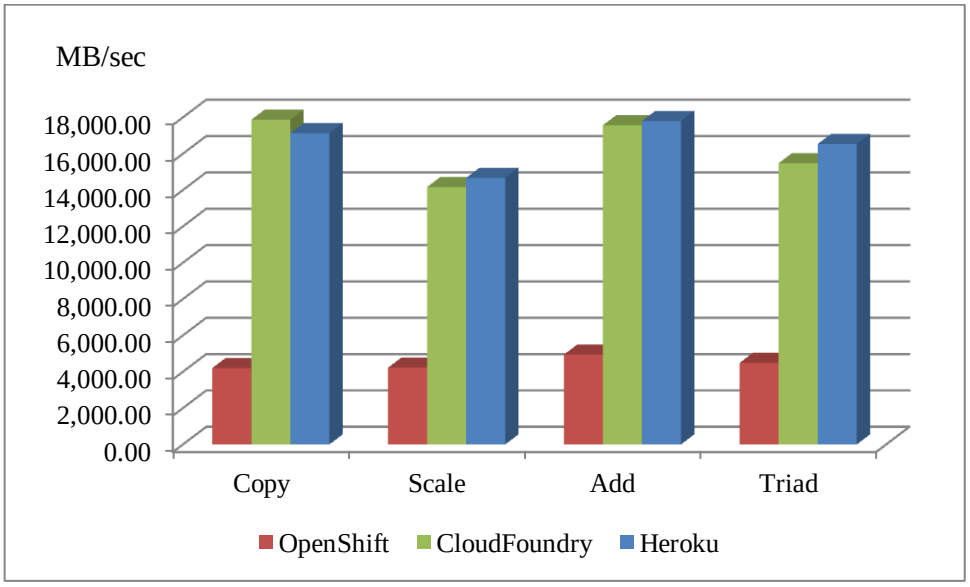
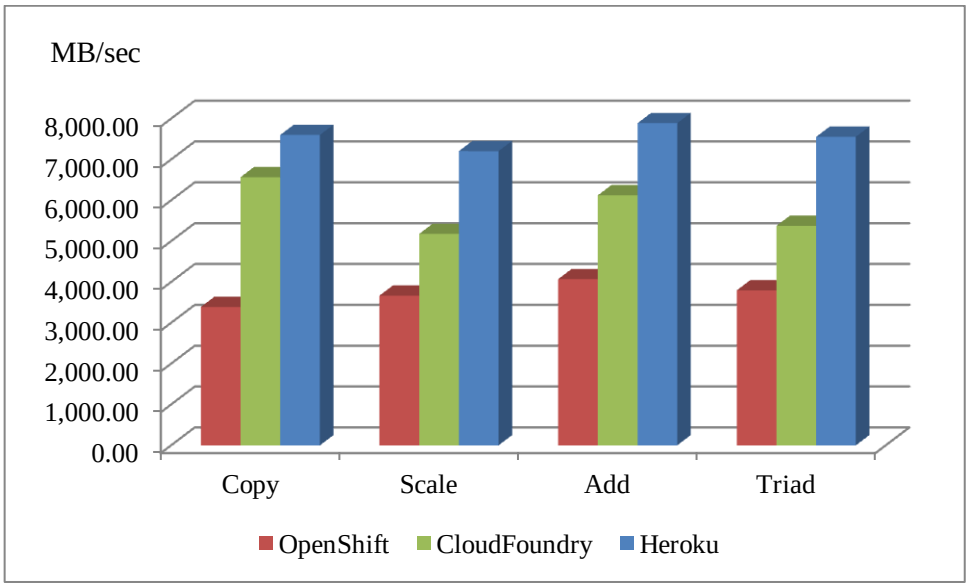


Figure 3.17: STREAM bandwidth test results for 100 MB



4. RESULTS

4.1 EVALUATION OF THE RESULTS WITH AHP

To decide which platform is better based on the intuition is very hard if not impossible. It gets overwhelming to re-evaluate the candidates if the importance of the functions is changed for the consumer. Therefore, there is a need for statistical methods to evaluate the performance test results.

The AHP calculation results for an enterprise application scenario are given in Table 4.1. This scenario is created by assigning weights to functions considering the needs of an average enterprise application. Since the enterprise applications also vary according to their architecture and area of business, it is not valid for every application. However, examples of such applications are frequently seen in banking and insurance sector. Also many CRM and ERP applications or B2B and B2C web applications approximately need such functional resource requirements and weights. The resulting scores of this scenario are 0.45649 for Cloud Foundry, 0.29036 for Heroku, 0.25315 for OpenShift.

The score values of AHP can be between 0 and 1 and they represent the worst and the best scores that can be achieved, respectively. Since the performance values are calculated as relative, for a platform to get the 0 score, it should perform almost 100 times worse than the both of the others. Similarly to get the score of 1, it should almost perform 100 better than the other two alternatives.

In the second scenario, the CPU and the memory bandwidth are the most needed resources. In this scenario, database operations are just for recording the results, which means that there is not much need for them and when needed, the database records are executed as bulk operations. Such applications can be thought as the ones that require high computing power and high utilization of memory bandwidth, like scientific applications, model simulation applications, or graphical processing applications. The resulting scores are 0.41691 for Cloud Foundry, 0.38377 for Heroku, and 0.19932 for

OpenShift. The weights and the RSRVs of this second type scenario are illustrated in Table 4.2.

Table 4.1: The weights and RSRVs for general enterprise application scenario

					Relative Service Ranking Vector		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**	**User Weights for Sub of Sub Functions**	**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.50	FP array	0.10	N/A	0.37058	0.17328	0.45614
		FP parameter	0.10	N/A	0.37287	0.17133	0.45580
		Conditional jumps	0.22	N/A	0.37147	0.19068	0.43785
		Int. arith.	0.20	N/A	0.35277	0.21703	0.43019
		Trigon. func.	0.05	N/A	0.36195	0.22844	0.40960
		Proced. calls	0.22	N/A	0.34705	0.14479	0.50816
		Array ref. assign.	0.10	N/A	0.34324	0.23015	0.42661
		Std. math. func.	0.01	N/A	0.36790	0.21297	0.41913
		Overall			0.35907	0.18824	0.45269
DB	0.30	Single Rec. - Read		0.40	0.08042	0.48382	0.43576
		Single Rec. – Update		0.20	0.08647	0.44858	0.46494
		Single Rec. - Insert		0.30	0.09783	0.47608	0.42609
		Single Rec. - Delete		0.10	0.08937	0.44723	0.46339
		Single Rec. Overall	0.80		0.08775	0.47079	0.44146
		100 Rec. - Read		0.40	0.08174	0.51534	0.40292
		100 Rec. - Update		0.20	0.05424	0.42787	0.51790
		100 Rec. - Insert		0.30	0.10320	0.37719	0.51961
		100 Rec. - Delete		0.10	0.06999	0.36010	0.56990
		100 Rec. Overall	0.20		0.08150	0.44088	0.47762
		Overall			0.08650	0.46481	0.44869
Memory	0.20	128 KB	0.84	N/A	0.42071	0.09261	0.48668
		1 MB	0.15	N/A	0.44363	0.11976	0.43661
		100 MB	0.01	N/A	0.44215	0.21809	0.33975
		Overall			0.42437	0.09794	0.47770
Overall					0.29036	0.25315	0.45649

Table 4.2: The weights and RSRVs for scientific/graphical applications scenario

					Relative Service Ranking Vectors		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**	**User Weights for Sub of Sub Functions**	**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.50	FP array	0.10	N/A	0.37058	0.17328	0.45614
		FP parameter	0.10	N/A	0.37287	0.17133	0.45580
		Conditional jumps	0.20	N/A	0.37147	0.19068	0.43785
		Int. arith.	0.10	N/A	0.35277	0.21703	0.43019
		Trigon. func.	0.20	N/A	0.36195	0.22844	0.40960
		Proced. calls	0.15	N/A	0.34705	0.14479	0.50816
		Array ref. assign.	0.05	N/A	0.34324	0.23015	0.42661
		Std. math. func.	0.10	N/A	0.36790	0.21297	0.41913
		Overall			0.36232	0.19451	0.44317
DB	0.05	Single Rec. - Read		0.20	0.08042	0.48382	0.43576
		Single Rec. - Update		0.30	0.08647	0.44858	0.46494
		Single Rec. - Insert		0.40	0.09783	0.47608	0.42609
		Single Rec. - Delete		0.10	0.08937	0.44723	0.46339
		Single Rec. Overall	0.80		0.09010	0.46649	0.44341
		100 Rec. - Read		0.20	0.08174	0.51534	0.40292
		100 Rec. - Update		0.30	0.05424	0.42787	0.51790
		100 Rec. - Insert		0.40	0.10320	0.37719	0.51961
		100 Rec. - Delete		0.10	0.06999	0.36010	0.56990
		100 Rec. Overall	0.20		0.08090	0.41832	0.50079
		Overall			0.08826	0.45686	0.45488
Memory	0.45	128 KB	0.10	N/A	0.42071	0.09261	0.48668
		1 MB	0.30	N/A	0.44363	0.11976	0.43661
		100 MB	0.60	N/A	0.44215	0.21809	0.33975
		Overall			0.44045	0.17604	0.38350
Overall					0.38377	0.19932	0.41691

The third scenario simulates another application type in which database performance is the most important, and then comes the memory and CPU. The distribution of weights in sub functionalities is also subjective to the requirements of application. These weights are given just for demonstration purpose. The resulting scores are 0.45065 for Cloud Foundry, 0.30342 for OpenShift, and 0.24594 for Heroku. The weights and the RSRVs are shown in Table 4.3 below for this scenario:

Table 4.3: The weights and RSRVs for database intensive application scenario

					Relative Service Ranking Vector		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**	**User Weights for Sub of Sub Functions**	**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.20	FP array	0.10	N/A	0.37058	0.17328	0.45614
		FP parameter	0.10	N/A	0.37287	0.17133	0.45580
		Conditional jumps	0.30	N/A	0.37147	0.19068	0.43785
		Int. arith.	0.10	N/A	0.35277	0.21703	0.43019
		Trigon. func.	0.05	N/A	0.36195	0.22844	0.40960
		Proced. calls	0.20	N/A	0.34705	0.14479	0.50816
		Array ref. assign.	0.10	N/A	0.34324	0.23015	0.42661
		Std. math. func.	0.05	N/A	0.36790	0.21297	0.41913
		Overall			0.36129	0.18741	0.45130
DB	0.50	Single Rec. - Read		0.25	0.08042	0.48382	0.43576
		Single Rec. - Update		0.30	0.08647	0.44858	0.46494
		Single Rec. - Insert		0.40	0.09783	0.47608	0.42609
		Single Rec. - Delete		0.05	0.08937	0.44723	0.46339
		Single Rec. Overall	0.50		0.08965	0.46832	0.44203
		100 Rec. - Read		0.25	0.08174	0.51534	0.40292
		100 Rec. - Update		0.30	0.05424	0.42787	0.51790
		100 Rec. - Insert		0.40	0.10320	0.37719	0.51961
		100 Rec. - Delete		0.05	0.06999	0.36010	0.56990
		100 Rec. Overall	0.50		0.08149	0.42608	0.49244
		Overall			0.08557	0.44720	0.46723
Memory	0.30	128 KB	0.30	N/A	0.42071	0.09261	0.48668
		1 MB	0.40	N/A	0.44363	0.11976	0.43661
		100 MB	0.30	N/A	0.44215	0.21809	0.33975
		Overall			0.43631	0.14112	0.42257
Overall					0.24594	0.30342	0.45065

The last scenario is absolutely imaginary. Every sub functions, and main functions are equally weighted. This scenario is just to show the performance comparison of platforms in which every function evaluated has equal impact on the overall ranking. Again, the Cloud Foundry is the first, Heroku is the second and OpenShift is the third

with the scores of 0.44652, 0.29376, and 0.25972 respectively. The weights and the RSRVs are shown in Table 4.4 below for this imaginary equally weighted scenario:

Table 4.4: The weights and RSRVs for equally weighted functions scenario

					Relative Service Ranking Vector		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**	**User Weights for Sub of Sub Functions**	**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.34	FP array	0.125	N/A	0.37058	0.17328	0.45614
		FP parameter	0.125	N/A	0.37287	0.17133	0.45580
		Conditional jumps	0.125	N/A	0.37147	0.19068	0.43785
		Int. arith.	0.125	N/A	0.35277	0.21703	0.43019
		Trigon. func.	0.125	N/A	0.36195	0.22844	0.40960
		Proced. calls	0.125	N/A	0.34705	0.14479	0.50816
		Array ref. assign.	0.125	N/A	0.34324	0.23015	0.42661
		Std. math. func.	0.125	N/A	0.36790	0.21297	0.41913
		Overall			0.36098	0.19608	0.44294
DB	0.33	Single Rec. - Read		0.25	0.08042	0.48382	0.43576
		Single Rec. – Update		0.25	0.08647	0.44858	0.46494
		Single Rec. - Insert		0.25	0.09783	0.47608	0.42609
		Single Rec. - Delete		0.25	0.08937	0.44723	0.46339
		Single Rec. Overall	0.50		0.08852	0.46393	0.44755
		100 Rec. - Read		0.25	0.08174	0.51534	0.40292
		100 Rec. - Update		0.25	0.05424	0.42787	0.51790
		100 Rec. - Insert		0.25	0.10320	0.37719	0.51961
		100 Rec. - Delete		0.25	0.06999	0.36010	0.56990
		100 Rec. Overall	0.50		0.07729	0.42013	0.50258
		Overall			0.08291	0.44203	0.47506
Memory	0.33	128 KB	0.34	N/A	0.42071	0.09261	0.48668
		1 MB	0.33	N/A	0.44363	0.11976	0.43661
		100 MB	0.33	N/A	0.44215	0.21809	0.33975
		Overall			0.43535	0.14298	0.42167
Overall					0.29376	0.25972	0.44652

4.2 EVALUATION OF THE RESULTS WITH LSP

Table 4.5 shows the preference scores for the enterprise application scenario. The weights used here are the same with the weights used in AHP for the same scenario.

Here, Cloud Foundry is the best, OpenShift is ranked as the second platform, and Heroku is the third.

Table 4.5: LSP results for general enterprise application scenario

					Preference Scores		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**		**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.50	FP array	0.10		0.52871	0.20918	0.66726
		FP parameter	0.10		0.51756	0.19920	0.64856
		Conditional jumps	0.22		0.51468	0.22943	0.61941
		Int. arith.	0.20		0.60921	0.27860	0.79778
		Trigon. func.	0.05		0.74973	0.39942	0.87477
		Proced. calls	0.22		0.44069	0.11911	0.69686
		Array ref. assign.	0.10		0.52826	0.27183	0.71728
		Std. math. func.	0.01		0.45573	0.17958	0.54704
		Overall			0.52343	0.20161	0.69527
DB	0.30	Single Rec. - Read		0.4	0.05018	0.88601	0.86763
		Single Rec. – Update		0.2	0.13501	0.87574	0.88196
		Single Rec. - Insert		0.3	0.22277	0.88210	0.86209
		Single Rec. - Delete		0.1	0.17095	0.87644	0.88259
		Single Rec. Overall	0.80		0.05379	0.70037	0.69113
		100 Rec. - Read		0.40	0.35527	0.95383	0.92234
		100 Rec. - Update		0.20	0.27813	0.96671	0.98409
		100 Rec. - Insert		0.30	0.48040	0.90626	0.95023
		100 Rec. - Delete		0.10	0.29561	0.91679	0.97197
		100 Rec. Overall	0.20		0.27188	0.74831	0.74708
		Overall			0.07221	0.70967	0.70193
Memory	0.20	128 KB	0.84		0.62034	0.08085	0.72880
		1 MB	0.15		0.63182	0.10709	0.62044
		100 MB	0.01		0.69529	0.21626	0.47637
		Overall			0.62275	0.08505	0.70813
Overall					0.10615	0.13551	0.69976

For LSP, the score can be in the range of 0 to 1. If the test results satisfy or exceed the maximum required value, then, its preference score is 100 percent or 1. If its value is equal or lower than the minimum, then, its score will be 0. The scores of LSP will be higher than AHP because AHP starts to calculate with relative values. Therefore, to get

the score 1 in AHP, the performance should be almost 100 times better than the others. Table 4.6 shows the preference scores for the scientific or graphical application scenario. Again, the weights are the same for this scenario with those used in AHP technique in Table 4.2. Cloud Foundry is the best, Heroku is ranked as the second, and OpenShift is the third.

Table 4.6: LSP results for scientific or graphical processing application scenario

					Preference Scores		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**		**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.50	FP array	0.10		0.52871	0.20918	0.66726
		FP parameter	0.10		0.51756	0.19920	0.64856
		Conditional jumps	0.20		0.51468	0.22943	0.61941
		Int. arith.	0.10		0.60921	0.27860	0.79778
		Trigon. func.	0.20		0.74973	0.39942	0.87477
		Proced. calls	0.15		0.44069	0.11911	0.69686
		Array ref. assign.	0.05		0.52826	0.27183	0.71728
		Std. math. func.	0.10		0.45573	0.17958	0.54704
		Overall			0.54109	0.21668	0.69241
DB	0.05	Single Rec. - Read		0.2	0.05018	0.88601	0.86763
		Single Rec. - Update		0.3	0.13501	0.87574	0.88196
		Single Rec. - Insert		0.4	0.22277	0.88210	0.86209
		Single Rec. - Delete		0.1	0.17095	0.87644	0.88259
		Single Rec. Overall	0.20		0.05379	0.70037	0.69113
		100 Rec. - Read		0.20	0.35527	0.95383	0.92234
		100 Rec. - Update		0.30	0.27813	0.96671	0.98409
		100 Rec. - Insert		0.40	0.48040	0.90626	0.95023
		100 Rec. - Delete		0.10	0.29561	0.91679	0.97197
		100 Rec. Overall	0.80		0.27188	0.74831	0.74708
		Overall			0.19033	0.73843	0.73549
Memory	0.45	128 KB	0.10		0.62034	0.08085	0.72880
		1 MB	0.30		0.63182	0.10709	0.62044
		100 MB	0.60		0.69529	0.21626	0.47637
		Overall			0.66783	0.15697	0.53711
Overall					0.43119	0.18174	0.60354

Table 4.7 shows the preference scores for the database intensive application scenario. These weights are also parallel to those used with AHP technique shown in Table 4.3. Cloud Foundry is the best, Heroku is ranked as the second, and OpenShift is the third.

Table 4.7: LSP results for database intensive application scenario

					Preference Scores		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**		**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.20	FP array	0.10		0.52871	0.20918	0.66726
		FP parameter	0.10		0.51756	0.19920	0.64856
		Conditional jumps	0.30		0.51468	0.22943	0.61941
		Int. arith.	0.10		0.60921	0.27860	0.79778
		Trigon. func.	0.05		0.74973	0.39942	0.87477
		Proced. calls	0.20		0.44069	0.11911	0.69686
		Array ref. assign.	0.10		0.52826	0.27183	0.71728
		Std. math. func.	0.05		0.45573	0.17958	0.54704
		Overall			0.51419	0.19940	0.67280
DB	0.50	Single Rec - Read		0.25	0.05018	0.88601	0.86763
		Single Rec - Update		0.30	0.13501	0.87574	0.88196
		Single Rec - Insert		0.40	0.22277	0.88210	0.86209
		Single Rec - Delete		0.05	0.17095	0.87644	0.88259
		Single Rec. Overall	0.50		0.09843	0.88086	0.87036
		100 Rec. - Read		0.25	0.35527	0.95383	0.92234
		100 Rec. - Update		0.30	0.27813	0.96671	0.98409
		100 Rec. - Insert		0.40	0.48040	0.90626	0.95023
		100 Rec. - Delete		0.05	0.29561	0.91679	0.97197
		100 Rec. Overall	0.50		0.35489	0.93585	0.95380
		Overall			0.18131	0.90788	0.91098
Memory	0.30	128 KB	0.30		0.62034	0.08085	0.72880
		1 MB	0.40		0.63182	0.10709	0.62044
		100 MB	0.30		0.69529	0.21626	0.47637
		Overall			0.64655	0.12015	0.60028
Overall					0.22458	0.16954	0.71637

Table 4.8 shows the preference scores for the imaginary equally weighted functions scenario. Weights are parallel to the application of AHP technique shown in Table 4.4. Cloud Foundry is the best, OpenShift is ranked as the second, and Heroku is the third.

Table 4.8: LSP results for equally weighted functions scenario

					Preference Scores		
Functions	**User Weights for Functions**	**Sub Functions**	**User Weights for Sub Functions**		**Heroku**	**Open Shift**	**Cloud Foundry**
CPU	0.34	FP array	0.125		0.52871	0.20918	0.66726
		FP parameter	0.125		0.51756	0.19920	0.64856
		Conditional jumps	0.125		0.51468	0.22943	0.61941
		Int. arith.	0.125		0.60921	0.27860	0.79778
		Trigon. func.	0.125		0.74973	0.39942	0.87477
		Proced. calls	0.125		0.44069	0.11911	0.69686
		Array ref. assign.	0.125		0.52826	0.27183	0.71728
		Std. math. func.	0.125		0.45573	0.17958	0.54704
		Overall			0.53163	0.21465	0.68510
DB	0.33	Single Rec. - Read		0.25	0.05018	0.88601	0.86763
		Single Rec. – Update		0.25	0.13501	0.87574	0.88196
		Single Rec. - Insert		0.25	0.22277	0.88210	0.86209
		Single Rec. - Delete		0.25	0.17095	0.87644	0.88259
		Single Rec. Overall	0.50		0.05379	0.70037	0.69113
		100 Rec. - Read		0.25	0.35527	0.95383	0.92234
		100 Rec. - Update		0.25	0.27813	0.96671	0.98409
		100 Rec. - Insert		0.25	0.48040	0.90626	0.95023
		100 Rec. - Delete		0.25	0.29561	0.91679	0.97197
		100 Rec. Overall	0.50		0.27188	0.74831	0.74708
		Overall			0.11522	0.72388	0.71848
Memory	0.33	128 KB	0.34		0.62034	0.08085	0.72880
		1 MB	0.33		0.63182	0.10709	0.62044
		100 MB	0.33		0.69529	0.21626	0.47637
		Overall			0.64792	0.12120	0.59925
Overall					0.16378	0.16418	0.65962

5. DISCUSSION AND CONCLUSION

5.1 DISCUSSION

As shown in the experiment results section, Cloud Foundry performed as the best in all categories of Whetstone benchmark, both in MFLOPS of mathematical operations and times to complete the solution. Cloud Foundry performs better than the two other PaaS platforms. MFLOPS results for floating point operations of Cloud Foundry are better than its nearest competitor Heroku by 23 percent, 22 percent, and 46 percent in N1, N2, N6 floating point operations respectively. Heroku is better than OpenShift by 114 percent, 118 percent, and 140 percent in N1, N2, and N6 floating point operations respectively. Cloud Foundry is also better than Heroku in MOPS by 18 percent, 22 percent, 13 percent, 24 percent, and 14 percent in N3, N4, N5, N7, and N8 operation categories respectively. Heroku is again better than OpenShift in MOPS by 95 percent, 63 percent, 58 percent, 49 percent, and 73 percent in N3, N4, N5, N7, and N8 operation categories respectively.

The case is almost the same in timings of Whetstone test results. Cloud Foundry performs 14 percent to 37 percent better than Heroku in timing. Heroku performs 30 percent to 56 percent better than OpenShift.

Therefore, Cloud Foundry is the best in computing power with at least 13 percent difference according to its competitors. OpenShift is the worst in these results with high percentage difference according to other two platforms.

In database operations, the situation is not the same. Cloud Foundry and OpenShift perform almost at the same levels. Nevertheless, Heroku performs far worse than the other two platforms. The difference of Cloud Foundry and OpenShift is almost zero on the average in connection, statement and operation execution time. However, Heroku is worse 344 percent, 183 percent, and 1124 percent than other two platforms in

connection creation, statement creation, and operation execution respectively. Heroku's big percentage difference in database tests is very obvious.

In memory bandwidth tests, the situation is changed again. Cloud Foundry and Heroku performs almost same in 1 MB workload, whereas Cloud Foundry is better in 128 KB, and Heroku is better on 100 MB workloads. In 128 KB workload Cloud Foundry is better than Heroku by 16 percent in average. However, in 1 MB workload Heroku is better than Cloud Foundry by 2 percent. In 100 MB workload Heroku is better than Cloud Foundry by 30 percent in average. OpenShift is the worst in these tests. The nearest competitor is better than OpenShift by 56 percent to 426 percent.

The results are interesting. In almost each category, the ranking differs. In database operations, Cloud Foundry is the best; however, OpenShift is very close to it, and Heroku is the worst with a big difference. In computing power, Cloud Foundry is the best, Heroku is ranked as the second, and OpenShift is the worst of three. In memory bandwidth, Heroku is the best; however, Cloud Foundry is very close to it, and OpenShift is the worst of three. The best platform seems to be the Cloud Foundry according to these results even though it is not the best in memory bandwidth. Heroku may be the choice after Cloud Foundry, if there is not much work with database operations but computing power is important. OpenShift may be the choice after Cloud Foundry, if there is not much need for computing power but there is need for database operations.

The suggestions in the previous paragraph were based on intuition. Therefore, it is not easy to see that which provider will perform better if the importance of the functions is changed for the consumer. Hence, there is a need for statistical methods to evaluate the performance test results. That is why, AHP and LSP are used to select and rank the alternatives. However, selection and ranking highly depend on which function is more important than the others, according to the needs. That is the reason for presenting different scenarios having different weights for each resource and sub function.

The first scenario simulates the needs of an average enterprise application. Even though such enterprise applications differ in structure for different business sectors, it can be accepted roughly. AHP and LSP give different ranking for this scenario. AHP ranks the providers as Cloud Foundry 0.45649, Heroku 0.29036, and OpenShift 0.25315 whereas LSP ranks the providers as Cloud Foundry 0.69976, OpenShift 0.13551, and Heroku 0.10615. LSP gives a better score for Cloud Foundry according to AHP relative to the other providers. This is due to the relative calculation logic of AHP and the maximum and minimum acceptable value ranges used in LSP.

In the second scenario, the most interested resources are CPU and the memory bandwidth. DB operations come after them or sometimes have very little importance in this scenario. AHP ranks the providers as Cloud Foundry 0.41691, Heroku 0.38377, and OpenShift 0.19932 whereas LSP ranks the providers as Cloud Foundry 0.60354, Heroku 0.43119, and OpenShift 0.18174. Both method ranks the platforms in same order; however, LSP again gives better score than the AHP for Cloud Foundry. LSP and AHP's scores are close to each other for other two providers.

The third scenario simulates the application type that performs database intensive operations. Here the most important resource need is for database. The second important thing is memory and then CPU is the third important resource. AHP ranks the providers as Cloud Foundry 0.45065, OpenShift 0.30342, and Heroku 0.24594 whereas LSP ranks the providers as Cloud Foundry 0.71637, Heroku 0.22458, and OpenShift 0.16954. Platforms are ranked in different order; however, Cloud Foundry is again the best for both methods and LSP gives better score than the AHP for it. Then it ranks Heroku as the second and OpenShift as the third with lower scores.

The last scenario is absolutely imaginary as mentioned before. Every item in each category is weighted equally. This represents a fair comparison on every function. AHP ranks the providers as Cloud Foundry 0.44652, Heroku 0.29376, and OpenShift 0.25972 whereas LSP ranks the providers as Cloud Foundry 0.65962, OpenShift 0.16418, and Heroku 0.16378. Here, LSP gives almost the same score for OpenShift and Heroku;

however, AHP gives almost 3.3 percent difference between them, ranking Heroku better than the OpenShift.

Both methods point out that the Cloud Foundry is the best platform among the three platforms. This is expected because Cloud Foundry is ranked as the first or the second in almost each category, and in the cases that it is the second; there is a small difference with the first platform. On the other side, the other two platforms perform as the worst in some categories and in some of those categories they have big differences with the closest one.

From the viewpoint of stability, the best platform is again different in each main category. Cloud Foundry is the most stable platform and it has very little deviation in CPU test results. The deviation is between 0.2 and 5.4 percent with an average of 1.5 percent. Heroku's deviation is between 11.5 to 25.8 percent with an average of 15 percent. OpenShift's deviation is between 12.7 and 15.9 percent with an average of 14.5 percent.

In database operations, all of the platforms have high deviations. The minimum deviations belong to Heroku. Heroku's deviation is between 57.5 to 71.5 percent with an average of 66.7 percent. Then OpenShift comes. Its deviation is between 70.4 to 148.2 percent with an average of 99.4 percent. The worst is Cloud Foundry. Its deviation is between 90.9 to 163.7 percent with an average of 112.6 percent.

In memory performance, the deviations are again low. The most stable platform is OpenShift. Its deviation is between 1.3 to 7.8 percent with an average of 3.5 percent. Then Cloud Foundry comes. Its deviation is between 5.6 to 7.8 percent with an average of 6.7 percent. The worst is Heroku. Its deviation is between 7.0 to 20.7 percent with an average of 12.7 percent.

Since Cloud Foundry is still in beta edition, the enterprise pricing is not available yet. Heroku charges $0.05 per hour for each dyno after the first free one. OpenShift charges $0.04 per small gear after the first three ones. Therefore, for the cases which have small

difference in scoring, OpenShift may be preferred since it has a lower price. However, since the price difference is not high, the cost may not be an important factor between Heroku and OpenShift. Even for 7/24 hour utilization, the cost difference of $0.01 will be $87.60 in total for one year. Nevertheless, if there is high need for resources and high number of dynos or gears is required, then this difference will rapidly increase. Of course pricing is highly variable and subject to change according to the market competition.

The summary of the evaluations of the performance tests is as follows:

i The best performer is Cloud Foundry. The second platform changes according to the weights of resource needed. LSP scores OpenShift as the second platform in two of the scenarios and it scores Heroku as the second platform in the other two scenarios. AHP ranks Heroku as the second for three of the scenarios and as the third in one of the scenarios.

ii The stability in database operations is an important issue for all of the platforms. They need to be improved significantly. The consumers will demand more stable operation times for database operations since it is almost inevitable for a large percent of the applications. The platforms have acceptable levels of stability in CPU and memory bandwidth.

5.2 CONCLUSION

The cloud computing is a relatively new topic. There have been a number of researches made in this area and many new researches are going on. The most of the studies are about IaaS solutions. Therefore, there is still much work to be done in PaaS and other types of cloud computing. Even though there are some researches about proposing constituted performance item categorization and using appropriate method to evaluate the platforms, this area still needs improvement.

To address these needs, a detailed and fine grained function categorization is proposed in this work. For evaluating and proposing the best platform, AHP and LSP methods are

used. These methods and test results are employed with four different scenarios to understand the behavior of methods better.

AHP is widely used as an evaluation method in different research areas. Its relative calculations and matrix based structure normalizes and refines the performance values of alternatives. LSP is another method used in many different disciplines as AHP. It normalizes the performance values via weighted powers. Both contain user weights to evaluate the effect of the performance items on the result in different scenarios. However, LSP's structure seems to be more reliable according to the writer of this thesis, since it employs weighted powers and those powers can be selected according to the relation between items. In AHP, if an alternative has lower values in an important category with respect to other alternatives but have much higher values in another category, it can score better than the other alternatives. This means that the better side of the platform will compensate the bad sides. This issue is decreased with the use of weights. However, it may still lead a wrong decision in some cases where a function that is not as important as the other one can compensate the performance of another important function. Therefore, in this work LSP is applied to the results in addition to AHP method.

In summary, this performance study shows that Cloud Foundry is the best platform among the three PaaS platforms according to the test results. The reason is that it is the best in many of the functions, and in functions that it is not the best; it has small difference with its alternative. The platform which ranked as the second, changes according to the evaluation method and the resources needed. AHP ranks Heroku as the second and OpenShift as the third for all the scenarios except the database intensive scenario. That is because of the very bad database results of Heroku. LSP ranks Heroku as the second for scientific application scenario and database intensive application scenario, whereas it ranks OpenShift as the second for enterprise application and equal weight application scenarios.

In stability side of the performance, the winner changes in each category. Cloud Foundry is the most stable platform in CPU performance. Then Heroku comes and

OpenShift is the worst. In database operations, the best one is Heroku. OpenShift is ranked as the second performer and the worst is Cloud Foundry. However, all of the three platforms need stability improvements in database operation times. When it comes to memory stability, the deviations are again low as in CPU. The best platform is OpenShift. Cloud Foundry is ranked as the second and Heroku is the third.

Overall, this thesis provides a comprehensive set of performance items. Database operations are also added to the performance items which are inevitable functions for PaaS solutions. Appropriate benchmarks for this set of features are also provided. Three public PaaS platforms have been tested in these performance functions which are Cloud Foundry, Heroku, and OpenShift. Test results are evaluated with four different scenarios to give an insight for the alternatives under different resource needs. Two different comparison methods, namely AHP and LSP, are applied to the results to reinforce the decision between the alternatives. According to the best knowledge of the writer of this thesis work, the LSP method is not applied to the selection of PaaS platform problem in previous studies before. AHP and LSP results show that the Cloud Foundry is the best PaaS alternative among the three platforms inspected.

The writer of this thesis believes that this thesis provides valuable information about comparison of performance of PaaS platforms. Of course, the PaaS is continuously improving and new features can be added to the function framework provided here. However, the ones provided will not change very much in near future. Also, the applied evaluation methods will be useful for future works. Another topic for future work is to inspect the performance of platforms in parallel computing. Such a study will show the dynamics of high computing resource utilization in PaaS platforms and how they perform under high volume of processing requirements.

REFERENCES

Books

Garfinkel, S. L., and Abelson, H., 1999. *Architects of the information society: Thirty-five years of the laboratory for computer science at MIT*. Cambridge, Massachusetts, U.S.A: MIT Press

Parkhill, D. F., 1966. *The challenge of the computer utility*. U.S.A: Addison-Wesley Publishing Company

Saaty, T.L., 1980. *The analytic hierarchy process*. New York, U.S.A: McGraw-Hill.

Periodicals

Garg, S. K., Versteeg, S., and Buyya, R., 2013. A framework for ranking of cloud computing services. *Future Generation Computer Systems*. **29** (4), pp. 1012-1023.

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J., 2011. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*. **22** (6), pp. 931 - 945.

Tran, V. X., Tsuji, H., and Masuda, R., 2009. A new QoS ontology and its QoS-based ranking algorithm for web services. *Simulation Modelling Practice and Theory*. **17** (8), pp. 1378–1398.

Wang, L., Laszewski, G. V., Kunze, M., and Tao. J., 2010. Cloud computing: A perspective study. *New Generation Computing.* **28** (2), pp. 137-146.

Other Sources

A complete history of cloud computing [online], 2013. http://www.salesforce.com/uk/socialsuccess/cloud-computing/the-complete-history-of-cloud-computing.jsp [accessed 13 January 2013]

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M., 2009. Above the clouds: A Berkeley view of cloud computing. *Electrical Engineering and Computer Sciences Technical Report*. University of California, Berkeley.

Binnig, C., Kossmann, D., Kraska, T., and Loesing, S., 2009. How is the weather tomorrow? Towards a benchmark for the cloud. *2nd International Workshop on Testing Database Systems, DBTest*, 29 June 2009, New York, USA

Carr, N., 2008. Cloud computing. *Encyclopedia Britannica*, [online] http://www.britannica.com/EBchecked/topic/1483678/cloud-computing [accessed 20 January 2013]

Chellappa, R., 1997. Intermediaries in cloud-computing: A new computing paradigm. *INFORMS meeting*, 26-29 October 1997, Dallas, U.S.A.

Cloud Services Measurement Initiative Consortium (CSMIC), Service measurement index, 2011. Carnegie Mellon University Silicon Valley, California, USA, [online] http://www.cloudcommons.com/documents/10508/186d5f13-f40e-47ad-b9a6-4f246cf7e34f. [accessed 27 January 2013].

Columbus, L., 2012. Cloud computing and enterprise software forecast update. *Forbes.com*, [online]. 8 November 2012, http://www.forbes.com/sites/louiscolumbus/2012/11/08/cloud-computing-and-enterprise-software-forecast-update-2012/ [accessed 20 January 2013].

Costa, P. J. P. d., and Cruz, A. M. R. d., 2012. Migration to Windows Azure - Analysis and comparison. *4th Conference of Enterprise Information Systems – aligning technology, organizations and people (CENTERIS 2012)*. Procedia Technology **5** (2012), pp. 93-102.

Dujmovic, J. J., 1996. A method for evaluation and selection of complex hardware and software systems. *The 22nd International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, 10-13 December 1996, San Diego, USA, CMG 96 Proceedings **1**, pp. 368-378.

Foster, I., Zhao, Y., Raicu, I., and Lu, S., 2008. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop*, 12-16 November 2008, GCE'08, Texas, U.S.A, pp. 1-10.

Getting Started - A short history of Git, 2013. [online] http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git [accessed 13 January 2013].

Gillet, S. E., and Kapor, M., 1996. The self-governing internet: Coordination by design [online], University of Massachusetts Institute of Technology, http://ccs.mit.edu/papers/CCSWP197/CCSWP197.html [accessed 20 January 2013]

Gong, C., Liu, J., Zhang. Q., Chen, H., and Gong, Z., 2010. The characteristics of cloud computing. *39th International Conference on Parallel Processing Workshops.* 13-16 September 2010, San Diego, California, U.S.A, pp. 275-279.

Kharif, O., 2012. Kleiner Perkins considering new fund for cloud-computing services startups. *Bloomberg*, [online], 10 February 2012, http://www.bloomberg.com/news/2012-02-10/kleiner-perkins-considering-new-fund-for-cloud-computing-services-startups.html [accessed 20 January 2013].

Kundra, V., December 9, 2010. 25 point implementation plan to reform federal information technology management [online], The White House, Washington, http://www.dhs.gov/sites/default/files/publications/digital-strategy/25-point-implementation-plan-to-reform-federal-it.pdf. [accessed 18 March 2013].

Licklider, J. C. R., and Taylor, R. W., 1968. The computer as a communication device, [online]. *Systems Research Center*. http://sloan.stanford.edu/mousesite/Secondary/Licklider.pdf. [accessed 26 May 2013].

Longbottom, R., Roy Longbottom's PC benchmark collection, [online], http://www.roylongbottom.org.uk [accessed 13 January 2013]

McCalphin, J., The Stream benchmark java code, [online], http://www.cs.virginia.edu/stream/FTP/Contrib/Java/STREAM.java [accessed 13 January 2013].

Mell, P., and Grance, T., 2011. The NIST definition of cloud computing, Recommendations of the National Institute of Standards and Technology, [online]. National Institute of Standards and Technology, U.S. Department of Commerce, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [accessed 20 January 2013].

Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W., and Li, Q., 2009. Comparison of several cloud computing platforms. *2nd International Symposium on Information Science and Engineering*, 26-28 December 2009, Shanghai, Hong Kong, pp. 23-27.

Perry, G., 2008. How cloud & utility computing are different, [online]. http://gigaom.com/2008/02/28/how-cloud-utility-computing-are-different/ [accessed 13 January 2013]

Roth, I., 2011. Announcing OpenShift: The platform-as-a-service for developers who love open source and CDI, [online]. http://cloudcomputing.info/en/news/2011/04/vmware-announces-its-paas-solution-called-cloud-foundry.html [accessed 13 January 2013]

Salah, K., Al-Saba, M., Akhdhor, M., Shaaban, O., and Buhari, M.I., 2011. Performance evaluation of popular cloud IaaS providers. *6th International Conference on Internet Technology and Secured Transactions*, 11-14 December 2011, Abu Dhabi, United Arab Emirates, pp. 345-349.

Salesforce.com signs definitive agreement to acquire Heroku, [online], 2010. http://news.heroku.com/news_releases/salesforcecom-signs-definitive-agreement-to-acquire-heroku [accessed 13 January 2013]

Schmidt, E., Conversation with Eric Schmidt hosted by Danny Sullivan, *Search Engine Strategies Conference*, [online], 9 August 2006, http://www.google.com/press/podium/ses2006.html [accessed 20 January 2013]

Sempolinski, P., and Thain, D., 2010. A comparison and critique of Eucalyptus, OpenNebula and Nimbus. *2nd IEEE International Conference on Cloud Computing Technology and Science*. 30 November 2010 - 3 December 2010, Indianapolis, U.S.A., pp. 417-426.

Surksum, K. v., 2011. VMware announces its PaaS solution called Cloud Foundry, [online]. http://cloudcomputing.info/en/news/2011/04/vmware-announces-its-paas-solution-called-cloud-foundry.html. [accessed 13 January 2013].

Tudoran, R., Costan, A., Antoniu, G., and Bougé, L., 2012. A performance evaluation of Azure and Nimbus clouds for scientific applications. *2nd International Workshop on Cloud Computing Platforms - CloudCP '12*. New York, USA, pp. 1-6.

Voras, I., Mihaljevic, B., and Orlic, M., 2011. Criteria for evaluation of open source cloud computing solutions. *33rd International Conference on Information Technology Interfaces*, 27-30 June 2011, Dubrovnik, Croatia, pp. 137-142.

Wind, S., 2011. Open source cloud computing management platforms: Introduction, comparison, and recommendations for implementation. *2011 IEEE Conference on Open Systems (ICOS2011)*, 25-28 September 2011, Langkawi, Malaysia, pp. 175-179.

Yu, H. Q., and Molina H., 2007. A modified logic scoring preference method for dynamic web services evaluation and selection. *The 2nd European Young Researchers Workshop on Service Oriented Computing*, 11-12 June 2007, University of Leicester, United Kingdom.

Yu, H. Q., and Reiff-Marganiec, S., 2008. A method for automated web service selection, *IEEE Congress on Services 2008*, 6-11 July 2008, Hawaii, U.S.A.