CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering

BACHELOR'S THESIS

Data movement in hybrid clouds

Pohyb dat v hybridním cloudu

Author: Matej Uhrín
Supervisor: Ing. Tomáš Vondra
Academic year: 2013/2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Matej U h r í n

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Data Movement in Hybrid Clouds

Guidelines:

1. Study the field of private and public cloud computing, mainly functionality of Amazon Web Services compatible clouds. Study the possibilities for backup and on-line replication of databases, e.g. MySQL or PostgreSQL. Verify the functionality of replication with an open-source web application of your choice.
2. Propose a method of profiling the demand of read/write and replication of a database on disk and network traffic. Perform measurements on more than one chosen web application.
3. Design a method to decide, whether it is more economical to run an application in multiple locations in hybrid cloud mode with all database acceses going through an internet line, replicate the database between locations, or when it is not economical at all.
4. The inputs should be the amount of read and write requests, the amount of data stored, the price for data storage (on both sites), price of data transfer and disk access.

Bibliography/Sources:

[1] Gunther, Neil J.: Analyzing Computer System Performance with Perl: PDQ. Springer 2011, chapter 11.
[2] Menascâe, Daniel A., et al.: Performance by design: computer capacity planning by example. Prentice Hall Professional, 2004, part I.
[3] Bossche, Van den, Kurt Vanmechelen, and Jan Broeckhove: "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds." 2013.
[4] Li, Ang, et al.: "CloudCmp: comparing public cloud providers." Proceedings of the 10th ACM SIGCOMM konference on Internet measurement. ACM, 2010.
[5] Hajjat, Mohammad, et al.: "Dealer: application-aware request splitting for interactive cloud Aplications." Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012.
[6] Lněnička, Martin: "The Design of User Interface for a Cloud Storage Selection with AHP in Python." Proceedings of 4. Masarykova PhD., Hradec Králové, 2013.

Bachelor Project Supervisor: Ing. Tomáš Vondra

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Matej U h r í n

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Pohyb dat v hybridním cloudu

Pokyny pro vypracování:

1. Prostudujte problematiku privátního a veřejného cloud computingu, zvláště funkce cloudů kompatibilních s Amazon Web Services. Nastudujte možnosti zálohování a on-line replikace databází, např. MySQL nebo PostgreSQL. Ověřte funkčnost replikace na open-source webové aplikaci dle svého výběru.
2. Navrhněte metodiku profilování náročnosti čtení/zápisů a replikace databáze na diskový a síťový provoz. Měření proveďte na více než jedné zvolené aplikaci a vyhodnoťte.
3. Navrhněte metodu rozhodování, zda je při provozu aplikace využívající relační databázi a běžící na více lokalitách v režimu hybridního cloudu výhodnější používat pro všechny přístupy internetovou linku či provádět mezi lokalitami replikaci databáze, případně kdy není vůbec výhodné aplikaci takto provozovat.
4. Vstupy by měly být množství přístupů pro čtení, množství zápisů, množství uložených dat, cena uložení dat (na obou místech), cena přenosu dat a diskových přístupů.

Seznam odborné literatury:

[1] Gunther, Neil J.: Analyzing Computer System Performance with Perl: PDQ. Springer 2011, chapter 11.
[2] Menascâe, Daniel A., et al.: Performance by design: computer capacity planning by example. Prentice Hall Professional, 2004, part I.
[3] Bossche, Van den, Kurt Vanmechelen, and Jan Broeckhove: "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds." 2013.
[4] Li, Ang, et al.: "CloudCmp: comparing public cloud providers." Proceedings of the 10th ACM SIGCOMM konference on Internet measurement. ACM, 2010.
[5] Hajjat, Mohammad, et al.: "Dealer: application-aware request splitting for interactive cloud Aplications." Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012.
[6] Lněnička, Martin: "The Design of User Interface for a Cloud Storage Selection with AHP in Python." Proceedings of 4. Masarykova PhD., Hradec Králové, 2013.

Vedoucí bakalářské práce: Ing. Tomáš Vondra

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Název práce: **Pohyb dat v hybridním cloudu**

Autor: Matej Uhrín

Obor: Otevřená Informatika

Zaměření: Informatika a počítačové vědy

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Tomáš Vondra, Katedra kybernetiky, FEL, ČVUT

Abstrakt: Tato práce analyzuje problematiku hybridních cloudů a specializuje se na webové aplikace v režimu hybridního cloudu. Obsahem téhle práce je návrh metody rozhodování, zda je při provozu aplikace využívající relační databázi a běžící na více lokalitách výhodnejší používat pro všechny přístupy internetovou linku nebo provádět mezi lokalitami replikaci databáze. Tato práce dál definuje všechny proměnné, se kterými metoda pracuje a popisuje proces profilování obecné aplikace. Práce končí ukázkou použití metody na dvě pravděpodobně nejčastejší webové aplikace.

Klíčová slova: Hybridní cloud, predikce ceny, databázová administrace, databázová replikace, zátěžové testy, systémová analýza

Title: **Data movement in Hybrid Clouds**

Author: Matej Uhrín

Abstract: The main focus of this work is on hybrid clouds and web applications running in hybrid cloud mode. This thesis provides a method to decide, whether it is more economical to run an application in multiple locations in hybrid cloud mode with all database acceses going through an internet line or replicate the database. This thesis further specifies metrics and variables used in the decision method. Moreover, a profiling process on how to perform measurements and collect the mentioned variables is explained. The work finishes with method being tested on two of the most common web applications.

Key words: Hybrid cloud, cost prediction, database administration, database replication, load-testing, system analysis

Acknowledgements

I am thankful to Tomáš Vondra for his patience with me and quick replies to my numerous questions. I am also thankful to my family for all the support they have given me.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržovaní etických principů při přípravě vysokoškolských zavěrečných prací.

V Praze dne 23.5. 2014 .....

Matej Uhrín

Table of content

1 Introduction **1**

2 Theory **3**
2.1 Cloud and Cloud Computing . . . 3
2.1.1 Infrastructure as a service . . . 3
2.1.2 Platform as a service . . . 3
2.1.3 Software as a service . . . 3
2.2 Cloud deployment models . . . 4
2.2.1 Private cloud . . . 5
2.2.2 Public cloud . . . 5
2.2.3 Hybrid cloud . . . 6
2.3 Replication . . . 6
2.3.1 Replication modes . . . 7
2.3.1.1 Synchronous . . . 7
2.3.1.2 Semi-synchronous . . . 7
2.3.1.3 Asynchronous . . . 7
2.3.2 Replication types . . . 8
2.3.2.1 Statement based replication . . . 8
2.3.2.2 Row based replication . . . 9
2.4 Testing replication in the cloud . . . 9
2.4.1 Test design considerations . . . 9
2.4.2 Master/Slave setup . . . 10
2.4.2.1 Backup . . . 10
2.4.2.2 Delayed slave . . . 10
2.4.2.3 Performance . . . 11
2.4.2.4 Load balancing . . . 11
2.4.3 Multi master setup . . . 13
2.4.3.1 Collisions . . . 14
2.4.3.2 Backup . . . 14
2.4.3.3 Performance . . . 14

3 Profiling **15**
3.1 Tools . . . 15
3.2 Metrics . . . 16
3.3 Data transfer . . . 17

3.3.1 Replication traffic 17
3.3.1.1 Traffic uncertainty 18
3.3.2 Remote DB traffic 19
3.3.3 Client traffic 20
3.4 Disk storage and IOPS 21
3.5 Performance 21
3.5.1 Apdex score 22

4 Decision Method **23**
4.1 Definition 23
4.1.1 Instance 23
4.1.2 Traffic 23
4.1.3 Storage 24
4.1.4 Performance 24
4.2 Further specification 24
4.2.1 Revision 25
4.3 E-commerce website 25
4.3.1 Latency vs. Performance 28
4.3.2 Storing bigger data 30
4.3.3 Conclusion 30
4.4 Community blog 31
4.4.1 Conclusion 33

5 Summary **34**

6 Appendix **35**
6.1 Load testing the eCommerce application 35
6.2 Load testing the community blog 37

Bibliography **40**

List of notations

Shortcut	Description
WAN	Wide Area Network
TX traffic	Transmitted traffic
RX traffic	Received traffic
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
HaaS	Hardware as a Service
DB	Database
SBR	Statement based replication
RBR	Row based replication
bin-log	Binary log
IOPS	Input/Output operations per second

Chapter 1

Introduction

The idea of 'renting' computational power has been around for a long time. In the beginning, only few companies in the world owned a computer and not many people actually needed one. For a person to use a computer he had to access mainframe, do what he needed and pay for resources he used. It was either usage-time, storage space or cpu cycles etc. This idea has been reborn on a much greater scale and it has been given a new name: "The Cloud". The Cloud has now the potential to change the World Wide Web as we know it.

This paper's main focus is on database administration in the Cloud. More specifically, this paper concerns about web applications in the hybrid cloud and how well these 2 things go together.

The second purpose of this paper is to provide all the useful ideas and information I have stumbled upon while working with clouds, databases and web applications.

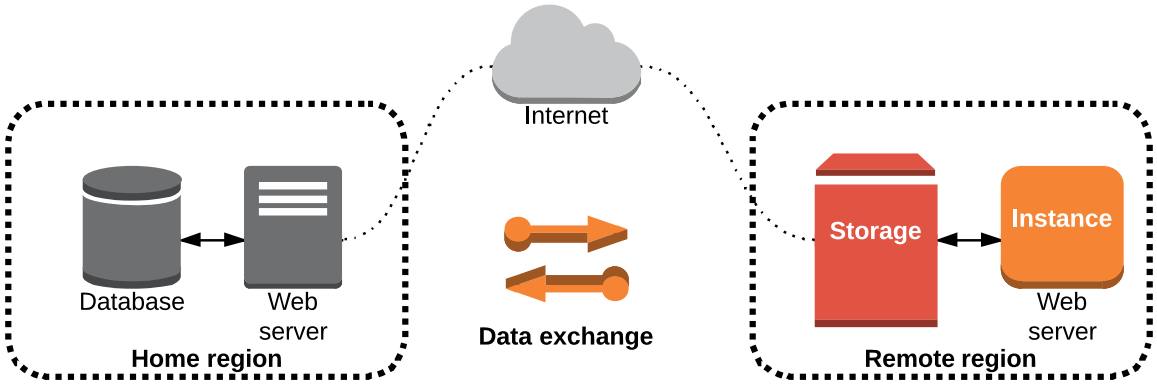

Figure 1.1: Problem definition.

One of the possible set-ups in which a web application can take advantage of the hybrid cloud is when a remote web server needs to be used. This paper aims to mention/solve problems that can arise in this situation and provide a decision method

between 2 possible database set-ups in this scenario:

1. Replicate data between on-premise database and cloud database.
2. Remotely access single database located 'on-premise'.

and discusses the suitability of running web application in the above mentioned mode. To distinguish replication and remote access, I will refer to them as the **Replication set-up** and the **Remote set-up** or scenarios further down in the text.

This thesis is organized as follows: In **chapter 2** I explain basic terms such as: Cloud Computing, Replication, etc . I explain and test different types of replication set-ups. Discussed are performance specifics and backup possibilities in each set-up, furthermore, I give necessary introduction to the load testing. This chapter can be skipped if reader is familiar with replication, load-testing concepts and terms like multi-master set-up and delayed slave.

In **chapter 3** I define system metrics that I will need and mention tools that I will use to collect them. Moreover, I give a step-by-step guide on how to predict metrics that can not be collected or accurately calculated.

In the final **chapter 4** I define the decision method and I use it on 2 applications. In the application section I aim to explain why and when should one of the set-ups be used and what to be aware of when using my method.

Chapter 2

Theory

2.1 Cloud and Cloud Computing

The cloud symbol has been used to portray the Internet or the WAN in network diagrams and flowcharts since the very beginning. As a result, the general term **Cloud Computing** has been adopted to refer to anything that involves delivering hosted services over the Internet. In other words: such services that happen in the **Cloud**. [10] These services are broadly divided into three categories:

2.1.1 Infrastructure as a service

IaaS, sometimes referred to as Hardware as a Service(HaaS), is a provision model in which the client pays for virtual hardware.The provider owns and manages the real hardware and is responsible for housing, running and maintenance. Most of the time virtual hardware means virtual server instance, disk storage, virtual network, network devices etc. The client pays per-use of resources both allocated and consumed.

2.1.2 Platform as a service

As the name suggests, PaaS provides a computing platform; be it database, web server or programming language execution environment. It is useful mainly for developers, as they can focus on developing their web application and do not have to worry about management or execution.The platform is often flexible with respect to hardware resources.(e.g. increase CPU if the traffic is higher). Heroku is a great representation of PaaS.

2.1.3 Software as a service

This acronym basically stands for a service running online with some form of access and a level of customization for end users. Access is often web browser based. Service can be either free or it can have a pricing model e.g. pay-per-use or free-premium. The main advantage of SaaS is the total absence of management costs and easy access from all around the world. Its disadvantage could be that end users cannot see what

the application really does or that the server operators are given the power to change the software in use. Well known examples of SaaS services are Google Apps and Spotify.

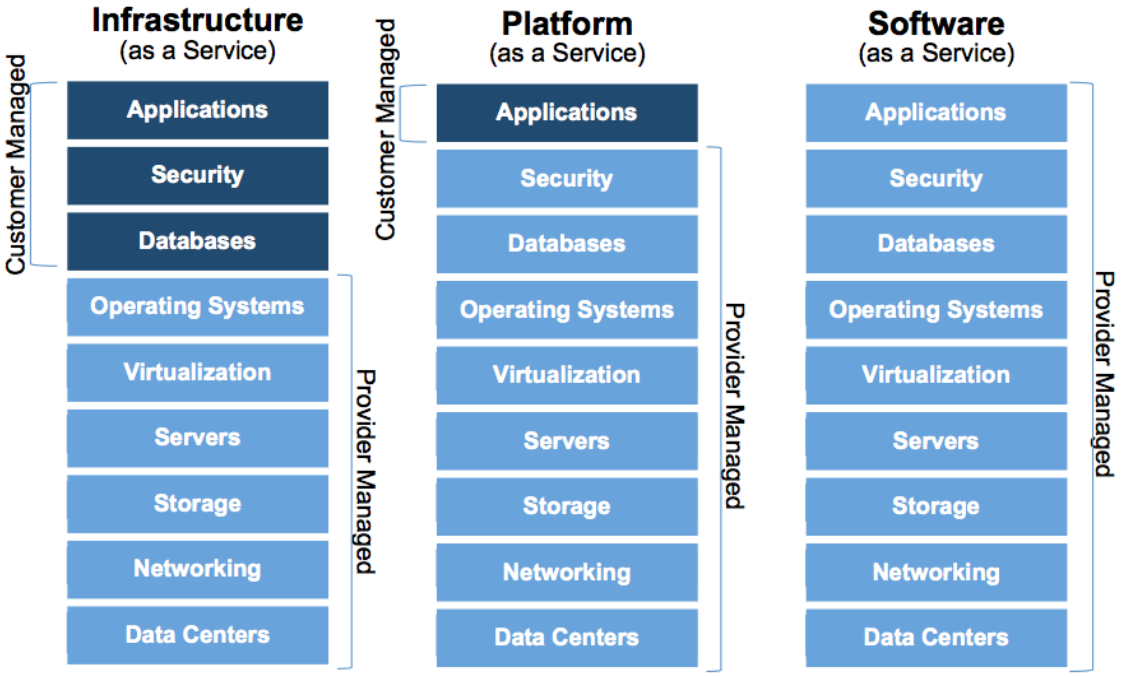

Figure 2.1: Level of involvement comparison[13]

It is clear that if I decide to self-host my application, I have to manage all of the resources displayed in 2.1.

Bottom-line:
To differentiate a cloud service from traditional hosting: Cloud service has three distinct characteristics [10]:

1. It is sold on demand, typically by the minute or the hour.
2. It is elastic: a user can have as much or as little of a service as they want at any given time.
3. The service is fully managed by the provider (the consumer needs nothing but a personal computer and Internet access).

2.2 Cloud deployment models

There are numerous approaches to clouds and how they can be used to the customer's advantage. For a very specific application a specific deployment model can be used. The most general ones are:

2.2.1 Private cloud

It is a model in which cloud infrastructure is used by a single organization meaning that it is a proprietary network running cloud computing technologies. Examples of such technologies are distributed computing and virtualization. Private cloud is sometimes referred to as Corporate cloud or Internal cloud.

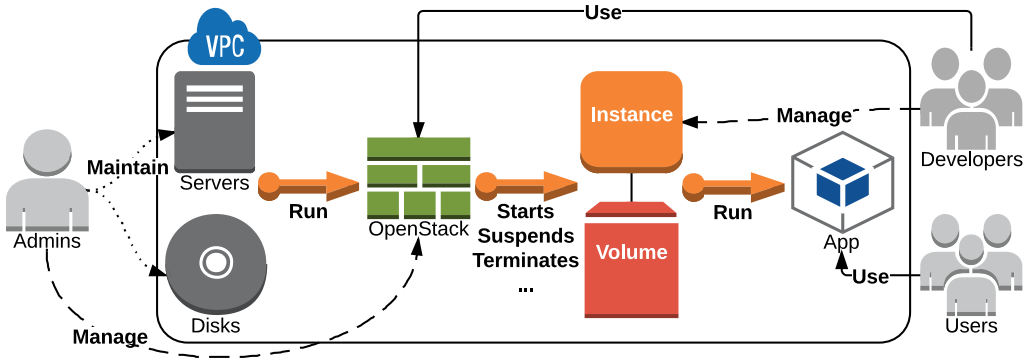

Figure 2.2: Private cloud running OpenStack as an IaaS

Private cloud has been heavily criticized. The main reason for this criticism is that you still have to own and therefore buy and maintain all the hardware. You still have to provide place and most probably upgrade hardware in the long run. In exchange you will get secure environment and more control over your data storage thanks to absolute control over your servers.

2.2.2 Public cloud

Public cloud provider makes resources(instances, volumes, applications...) available to the general public over the internet. Customers usually pay per usage of instances, storage... .

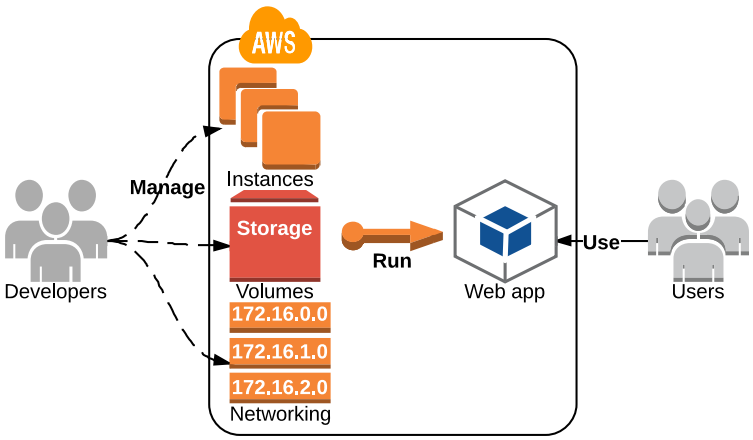

Figure 2.3: AWS(Amazon web services) as a public cloud

Public cloud seems to provide a perfect solution for start-ups. Inexpensive set-up and zero termination costs make it a wise choice for someone who is 'trying his luck' with a start-up and seemingly unlimited computational power could help also.

2.2.3 Hybrid cloud

Hybrid cloud is what you get when you combine public and private cloud. It is an environment in which an organization manages some resources on-premises and some are managed by a public cloud provider. It can be very useful if one wants to keep sensitive data 'home' and still use public cloud resources. One of the disadvantages might be the additional complexity of the development. One has to take care of both 'home' servers and AWS instances.

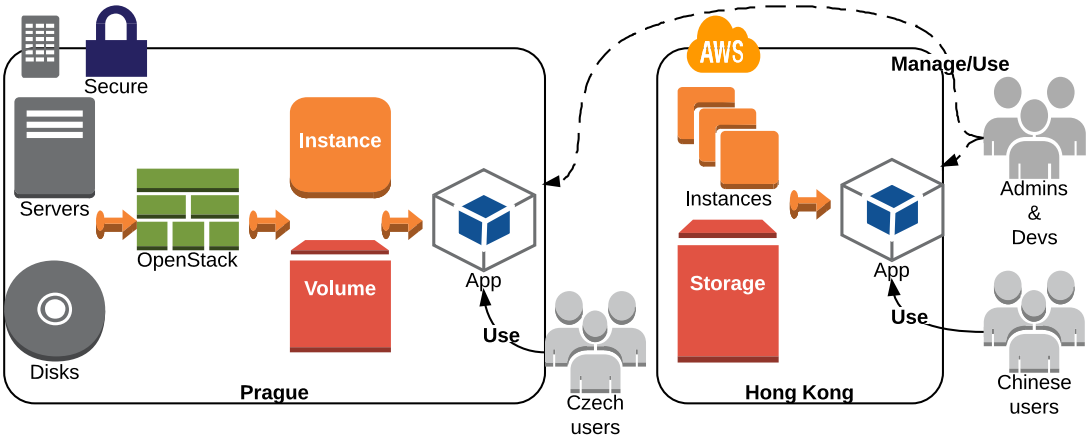

Figure 2.4: Hybrid cloud combining servers on-premise and payed instance from AWS

Hybrid cloud can also be used to run web application in a remote location while still having the same application running back home.

Suppose I am running an e-shop in the Czech Republic and I want to open a 'branch' for my Chinese customers. All I need to do is to buy a virtual server in Hong Kong and install my application. What will happen if I decide to share data between them ? What should I use and most importantly, how much will it cost ? I will answer these questions in the following chapters.

2.3 Replication

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. [12]

If correctly set up, replication will improve the performance and protect the availability of the application. In this chapter I will briefly explain various types of replication and discuss possibilities for backup and high-availability with different set ups.[2]

2.3.1 Replication modes

2.3.1.1 Synchronous

All servers are always synchronized in synchronous replication. It is due to the fact, that the transaction will be committed if and only if the disk writing was successful on both sides. The slave sends the acknowledgement(ACK) after the data has been successfully received and saved and the master has to receive acks from all slaves to proceed. It is clear that this behavior negatively affects performance, however, synchronous replication is useful if one seeks a guarantee of a zero-data loss.[15] [16]

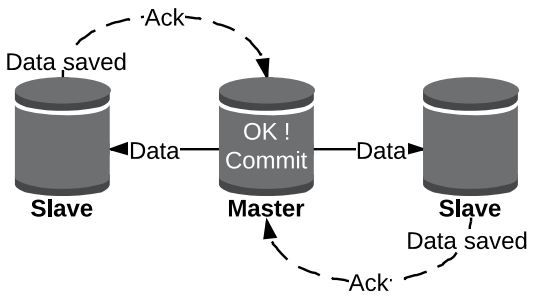

Figure 2.5: Synchronous replication. Inspired by [15]

2.3.1.2 Semi-synchronous

The difference between semi-synchronous replication and synchronous replication is that the slave sends acknowledgement 'sooner' than the actual disk write. Therefore, the transaction is also committed 'sooner' by master. In the case of MySQL, slave 'acks' master after "data has been written to slave's relay log and flushed to disk" and "master does not wait for all slaves to acknowledge receipt".([9] semi-synchro. ch.).

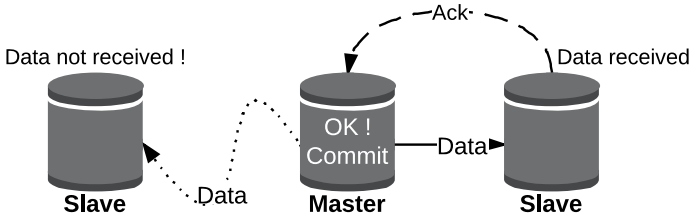

Figure 2.6: MySQL semi-synchronous replication

2.3.1.3 Asynchronous

There is no delay in the master's transaction commitment in case of the asynchronous replication."The master writes events to it's binary log and slaves request them when they are ready. There is no guarantee that any event will ever reach any slave."([9] semi-synchro. ch.) On the other hand, slave or number of slaves do not greatly affect the master's performance. Asynchronous replication is the one to be discussed in

this paper, mainly because of it's performance superiority in 'geo-replication' and the possibility to work with high latency internet connection.[19]

Figure 2.7: MySQL asynchronous replication

2.3.2 Replication types

Before I discuss 2 possible replication types, let me briefly explain the replication process in MySQL. There are 2 log files involved in the process:

1. binary log
2. relay log

The binary log and also relay log contains events describing database(DB) changes. Master's bin-log holds data that the slave reads and copies into it's own local relay log, which is being processed and containing events are being executed. The bin-log may consist of a single binary log file or a set of numbered log files(The same goes for relay log). In MySQL, there are 2 possible ways in which the events in bin-log can be stored. It translates into 2 possible replication types.

2.3.2.1 Statement based replication

In case of SBR, actual statements are written into the bin-log. With that being said, there are things one should keep an eye on: Non-deterministic statements(aka unsafe st.) are statements which may not have the same result on master and slave. Examples of such statements are[20]:

- UPDATE and DELETE using LIMIT clause and not using ORDER BY
- Statements using SYSDATE(), USER(), VERSION() ...

If any statement is considered unsafe it is logged with a warning.

When using SBR, I would also consider an advantage that the binary log is very easy to interpret using **mysqlbinlog** utility and one can clearly see table changes thanks to the statement format. See example below:

```
# at 26837146
%#140516 15:50:42 server id 2  end_log_pos 26837292 CRC32 ...
SET TIMESTAMP=1400255442/*!*/;
DELETE FROM wp_options WHERE option_name='auto_updater.lock'
```

2.3.2.2 Row based replication

As the name suggests, 'row' changes are logged in case of RBR. This makes the whole replication process safer, because all changes can be replicated.[20], although it usually result in bigger log files, because all changed rows have to be logged. Suppose I have issued the following statement

```
DELETE * FROM user WHERE name LIKE "J%"
```

In case of SBR, bin-log would only contain the actual statement(+ some additional lines). On the other hand, RBR bin-log would contain all the affected rows.

2.4 Testing replication in the cloud

In this section I would like to describe the actual replication set-ups and test their usability for back-up and high-availability. Before I do so, let me answer the obvious question: How do I test web applications ?

2.4.1 Test design considerations

Load testing is a broad subject and deserves a thesis of it's own. There are however few basic things one MUST(or should) take into consideration when designing load tests. For instance, there is no way a user can fill in the form in under 2 seconds unless it is an automated script. Furthermore, users usually use advanced web browser with caching and HTTP header management.

A proper test should try to mimic the user's interaction with the application to get as accurate behavior as possible and therefore a proper test should have natural thinking time to each user's action e.g. 3 seconds for **login**, 5 seconds for **form submit**, 2 seconds for typing **search** phrase etc. Furthermore, if one wishes to mimic user's actions, he should also add some form of cache so that the load-test will not load all of the page content from scratch every single time. Moreover, a cookie management is necessary for a load-tester who wishes to test any kind of admin or logged-in actions.

Why should one care about using HTTP headers in his tests ? Web browsers indicate support for compression with the **Accept-Encoding** header in the HTTP request.

```
Accept-Encoding: gzip, deflate
```

If a web server sees **gzip** in this header, it will compress it's responses. As a result, up to 70% reduction of a web application's 'weight' is possible.[3] and that surely is a significant difference.

This thesis is about cloud based web applications where outgoing traffic has major impact on the resulting cost. If I am going to predict cloud costs based on the result of a load-test, I might as well want to get correct predictions.

2.4.2 Master/Slave setup

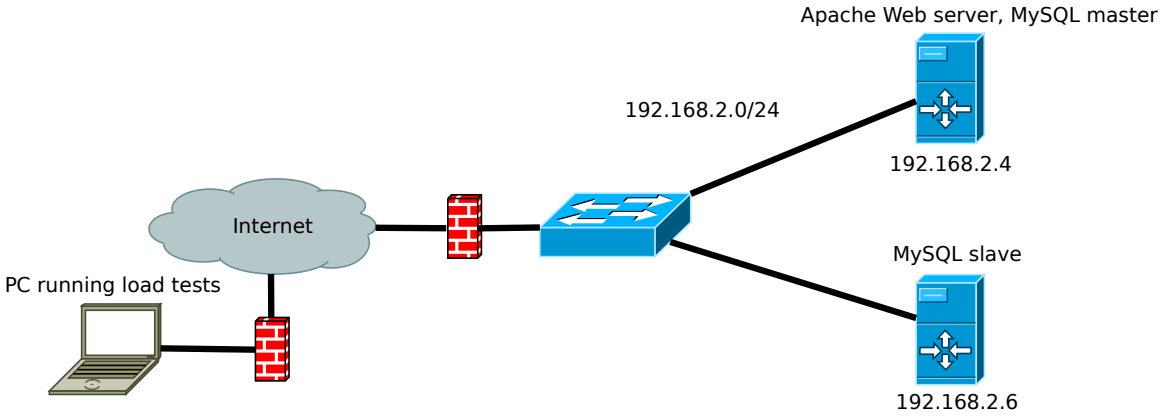

Figure 2.8: Master/Slave setup, application running on 192.168.2.4

2.4.2.1 Backup

This is the very basic replication set-up. Although, there is no load balancer neither as a stand alone server, nor as a built in web application feature, there are number of ways this particular setup can be used. Suppose I need to create a full snapshot of the current database using **mysqldump** without affecting the performance and/or making my database inconsistent.

"To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of "live" data that would otherwise require the master to be shut down."([9] repli. sec.)

Therefore, my best bet is to run the dump against the slave. It is possible to write complex statistical scripts that would result in mayor performance decrease if they were running on the master.It is however essential to not modify the slave data in any way apart from master updates as the inconsistency can cause trouble.

To make this clear: I firmly believe that a running MySQL slave should not be considered as a backup. Why ? If I run **DROP DATABASE** command on master, it will be replicated and executed on slave as well. Hence, my data is lost in a blink of an eye unless I used delayed slave.

2.4.2.2 Delayed slave

Delayed replication means that a slave purposely lags behind the master by at least a specified amount of time.([9] delayed slave sec.) It can be very useful for disaster roll-back, testing of system behavior when there is major lag between master and slave or for comparing data changes over the day. Delayed slave is available on most

DB engines: MongoDB, MySQL since v. 5.6. In MySQL setting up delayed slave by N seconds is done like this:

CHANGE MASTER TO **MASTER_DELAY** = N;

2.4.2.3 Performance

In this case I used a simulation system which consists of ubuntu 12.04.3 LTS servers connected together within a single private subnet. Each server has 1 virtual CPU, 4 GB hard-drive and 1 GB ram. I managed all of my instances through **openstack** dashboard. There is firewall with custom access-list protecting my subnet and there is public-ip address assigned to my application server. Each instance has MySQL server installed and there is always **asynchronous** MySQL replication running between servers.

My sample application has all the basic functions that most web applications have nowadays i.e. (form submit, home page visit, login, tag search). All the load tests were fired from my home computer. I have used Apache Benchmark for simple testing and JMeter as a load testing framework for custom actions that user might take.

It is clear that different actions require different time to be processed. Internet connection, number of DB queries or plain old fashioned code complexity are things that impact performance the most(In terms of response time).

2.4.2.4 Load balancing

I wanted to test whether having my mysql requests divided into read and write requests will be beneficial to my application i.e.: All the reads go to slave and all the writes go to master. I wanted to test this even though my servers have default settings and my application is not optimized.

I solved the load balancing problem at application level by using php script called **hyperdb**. This script takes care of choosing proper database server according to a database definition file. *Hyperdb* supports configurable priority for read and write servers, failover scenario for downed host and many other options.

If we compare **login** responses(see 2.9 & 2.10), it is clear that load balancing between master and slave did benefit my application a little, but I believe the real strength of load balancing shows when there are multiple slaves and heavy load of users.

I experimented with *hyperdb* priority groups a little and gave the same read priority to both master and slave and logically kept the write permission only on master. This way *hyperdb* chose read servers on random. Results were not much different. Further inspection on *hyperdb* proved that it also adds up small amount of time and resource usage to each request.

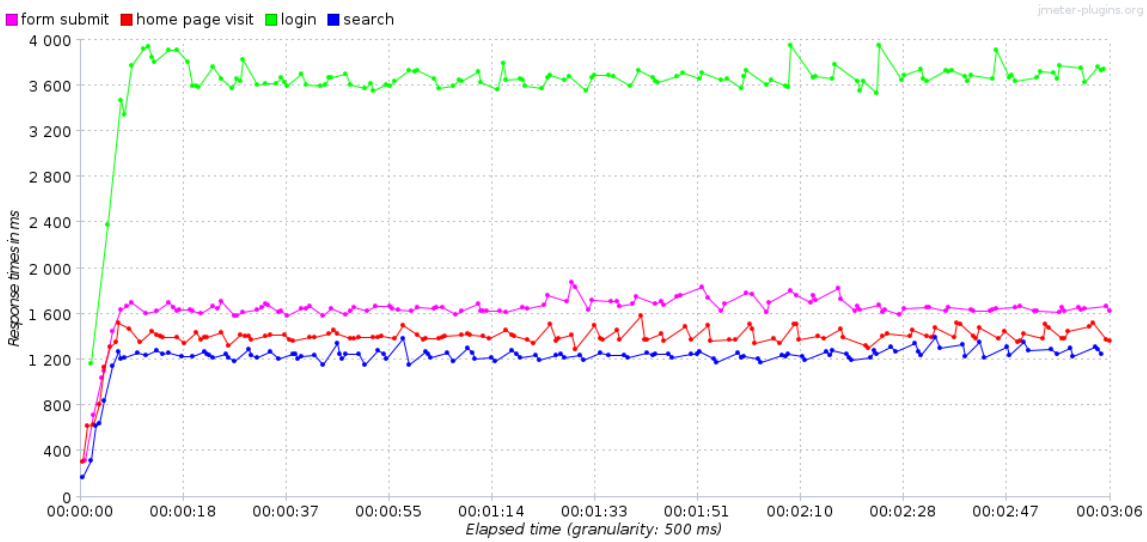

Figure 2.9: Response over time without *hyperdb*

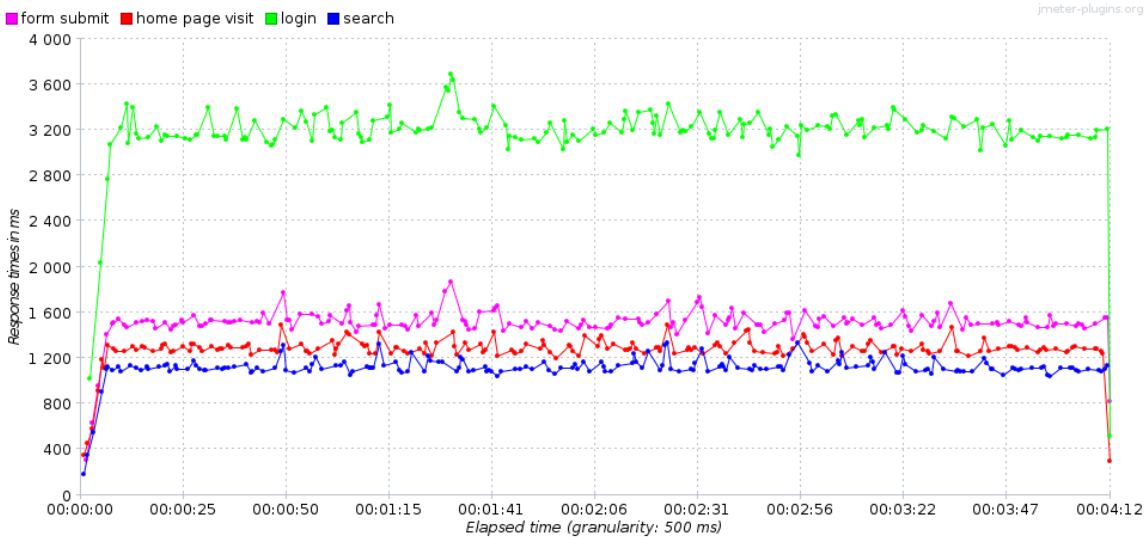

Figure 2.10: Response over time with *hyperdb*

2.4.3 Multi master setup

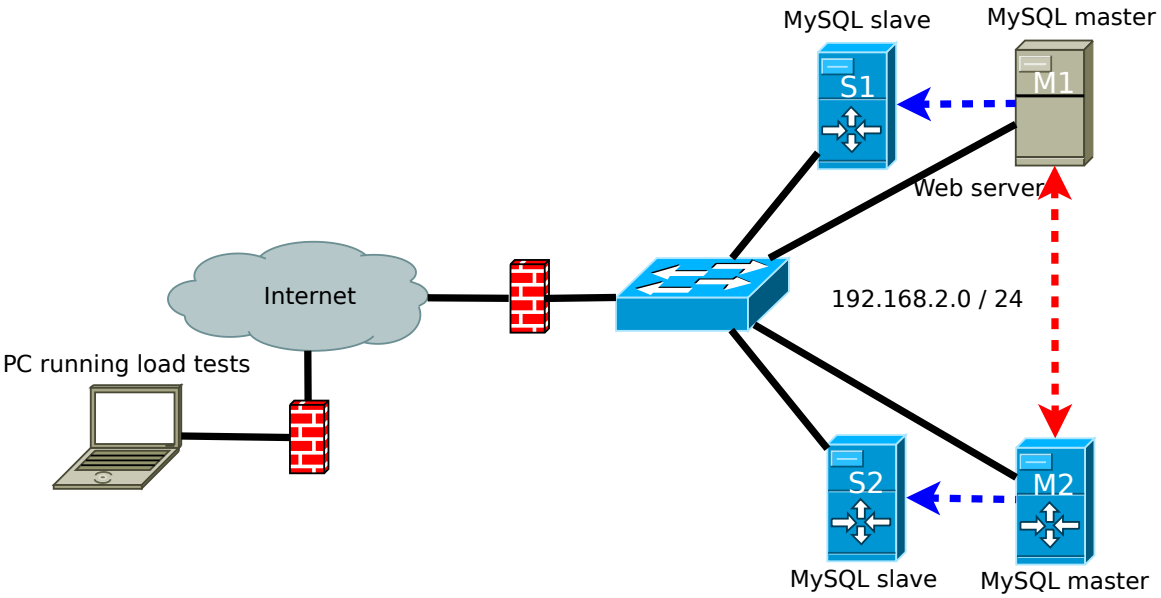

Figure 2.11: Multi master set up

Both masters are slaves to each other in case of MySQL asynchronous multi-master replication. MySQL replication works by copying master's bin-log contents into slave's relay-log and slave then begins to read and execute the updates present in the relay log. Hence, If I want to redistribute updates further e.g. from **M2** to **M1** and then to **S1**, I have to make sure that **M1** will also "binlog" these updates. The thing I need to add to each master's **my.cnf** configuration file is:

```
[mysqld]
log-slave-updates
```

The question that arises from this statement is: If **M1** sends update to **M2**. Will **M2** send the same update back to **M1**? Will it go on forever? The answer to the first question is: "Yes, **M2** will send the same update back to **M1**." **M1** will, however, make sure not to execute the update. (Replication sec. in [9] and [21])

Every bin-log and relay-log event includes **server-id** of the event. Server will only process relay-log event if it has different server-id than the server has. Furthermore, if **log-slave-updates** option is turned-on, relay-log event will also be recorded in local bin-log and distributed further.

Bottom line: If **log-slave-updates** is enabled, the replication traffic between servers will be doubled.

2.4.3.1 Collisions

In general, there are few crucial things that has to be checked in multi-master setup. One of them is to be aware of possible collisions if for instance

AUTO_INCREMENT

option is used in queries. In my application, I only needed to set up:

1. **auto_increment_offset**
2. **auto_increment_increment**

variables on both masters to ensure there will always be unique primary key for my insert query. However, in many situations developers may be unable to use master-master set-up, because of collisions that can not be sorted out so easily.

```
M1:
SET @@auto_increment_increment=2;
SET @@auto_increment_offset=1;
M2:
SET @@auto_increment_increment=2;
SET @@auto_increment_offset=2;
```

2.4.3.2 Backup

In the Master/Slave set-up 2.4.2, I have already mentioned using slave as some kind of backup "partition" where I can create snapshot without affecting the overall performance. It can come handy, but the real deal is consistent backup that will be ready to be used the second my primary database server goes down and able to catch-up if primary server ever goes back up. Such backup is often referred to as 'hot-standby'. This term is used for a server that "can take over for the other node with minimal to no warm up time".[21] It is clear that such server should be able to handle all of it's requests + requests from downed server.

If thoroughly load-tested, this simple setup(2.11)(together with the *hyperdb* application load balancer) can be considered a 'hot-standby' backup scenario.

2.4.3.3 Performance

After running a few test scenarios against multi master setup(2.11) with different priorities, I have come to conclusion that this setup benefits application's performance a lot. What I am basically doing is called **horizontal scaling** - "hiring more workers to do the task" or subtask if we speak about read-only slaves.

Chapter 3

Profiling

Measuring a web application can be a tricky process, because of the inability to measure specific metric or that the specific application may require specific measurements. The main purpose of this paper is a general approach to how a web application should be measured in order to decide between the **Replication set-up** or the **Remote set-up**. It is essential to choose the right measuring tools and the correct measurement metrics.

3.1 Tools

Most operating systems already have built-in utilities that can measure system statistics at the very basic level. However, measurement tools should be at least able to[1]:

1. Record and store data over time
2. Build custom metrics
3. Compare metrics from various sources
4. Import and export metrics

For instance: Unix iptables do provide built-in byte and packet counters, but to record and store these numbers over time one will need to install additional software.

The perfect software for this job seems to be unix daemon **Collectd**. It uses Round-Robin Database(RRD) to store data. Data can be stored locally, or it can be sent over the network to remote monitoring server.

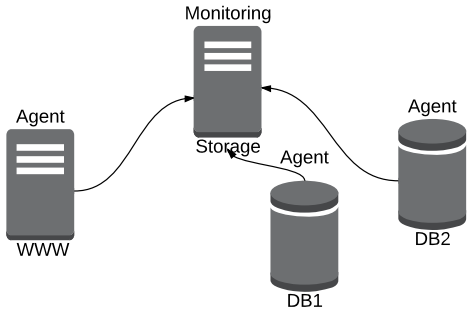

Figure 3.1: Collectd over the network example

Thanks to numerous useful plugins and advanced configuration options, collectd is the main profiling tool I need to use.

3.2 Metrics

Collecting application level metrics is crucial for any advanced web application. Examples of application level metrics are:

- Blog posts per week
- New forum threads per day
- Chat messages per second
- Items purchased per day
- Photos uploaded per week
- Origin of visitors

  ...

These metrics can be used to calculate profit, to predict trends in the future, or even to predict customer's behaviour. What is more, they can be surely transformed into a lower level metrics such as:

1. Data transfer
2. Disk storage needed
3. Disk input/output operations
4. Performance

   etc.

These are exactly the metrics one would need to assume the costs and profits in the cloud(No matter the provider).

I should be able to assume them fairly accurate with very few tools while the application is still running online without significiant performance decrease. In the following section I will define the above mentioned metrics, describe how to efficiently measure them in case of web application powered by MySQL.

3.3 Data transfer

Most cloud providers differ between incoming traffic and outgoing traffic(or **RX** and **TX**). Therefore, I have to make sure that my tools allow me to differ between RX and TX too.

If I decide to move my web server into the cloud, it is obvious that the data transfer costs will be represented by:

1. Replication over the internet or DB requests & data being transferred over the internet.
2. Client traffic.

3.3.1 Replication traffic

To assume the replication traffic I will take advantage of the **binary log**. "The binary log contains events that describe database changes such as table creation operations or changes to table data."[9]. According to the database replication definition: Database changes and table changes are replicated. Hence, the bin-log size more or less tells me how much data one would have to replicate. Inspect the following MySQL set-up of global variables:

```
| binlog_format          | MIXED
...
| log_bin                | ON
| log_bin_basename       | /var/log/mysql/mysql-bin
| log_bin_index          | /var/log/mysql/mysql-bin.index
```

I want to bring more attention to the **binlog_format** variable. Setting it up to MIXED format means that MySQL will decide; whether it is more efficient RBR or SBR. The other options for **binlog_format** are: STATEMENT and ROW. The binary log still fairly accurately reflects the actual interface traffic(Format does not change this fact).

All I need to do is: Record the size change of file **mysql-bin.0000XY** over time to be able to make an educated guess on the replication traffic. I can write a simple

script to do so or I can make good use of **MySQL** plugin in collectd plugin library and plugin's option: **MasterStats**.

```
<Plugin mysql>
        <Database something>
                Host "localhost"
                    Port "3306"
                ...
                MasterStats true
        </Database>
                ...
</Plugin>
```

This option will configure Collectd to periodically collect the results of mysql command **SHOW MASTER STATUS**. Although I did not set-up replication, MySQL server still writes into the binary log and **Position** variable is the exact byte size of the binary log file.

```
mysql> show master status\G
*************************** 1. row ***************************
             File: mysql-bin.000001
         Position: 13542191
                ...
```

i.e. in this particular case the mysql-bin.000001 file holds 13542191 bytes of data to be replicated.

Finally, I end up with fairly accurate assumption of replication traffic without going through the process of setting up the actual replication.

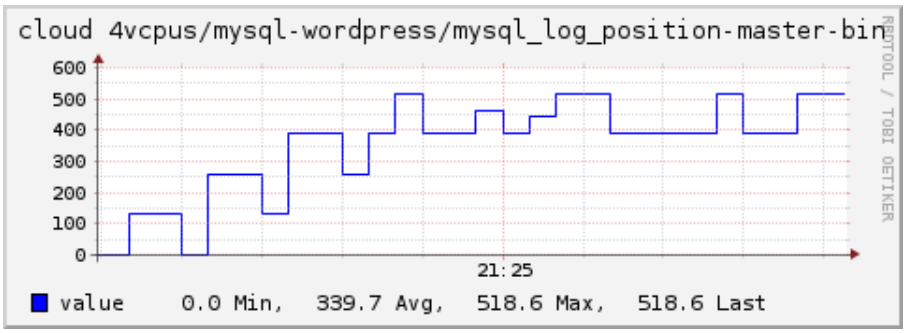

Figure 3.2: Log position over time with noticable ramp-up

3.3.1.1 Traffic uncertainty

Under special circumstances, predicting the replication traffic can be a problem. Specifically, if replication set-up takes advantage of **slave compressed protocol**, the real traffic will be lower than our prediction based on bin-log size. Exactly how

much lower the traffic will be depends on operating system and the actual implementation. For instance: MySQL uses **zlip** compression engine and according to [8] it can compress the data to a third of it's original size.

If we speak about 'geo-replication', the next thing that must be mentioned is **TLS/SSL**. My experience is that the SSL encryption does not significiantly impact traffic when used for replication unless additional compression is used.

3.3.2 Remote DB traffic

It represents the DB requests and DB data that would transform into TX and RX traffic if I moved my database to a remote location.

To guess such traffic I can use MySQL global_status variables: **Bytes_received**, **Bytes_sent** and set up the Collectd to record the change of these variables over time.

```
mysql> SELECT * FROM information_schema.global_status
-> WHERE variable_name IN ('Bytes_received','Bytes_sent');
+-----------------+----------------+
| VARIABLE_NAME   | VARIABLE_VALUE |
+-----------------+----------------+
| BYTES_RECEIVED  | 1534316315     |
| BYTES_SENT      | 22984070129    |
+-----------------+----------------+
```

However, it turns out that these variables are not very accurate when it comes to assuming the actual traffic flowing through the interface. Hence, I need to come up with a better method. Very simple solution to this problem is to use built-in byte/packet counters of unix **iptables**.

Inspect the following simplified iptables set-up.

```
root@m:/# iptables -A INPUT -p tcp --sport 3306 -j ACCEPT
root@m:/# iptables -A INPUT -p tcp --dport 3306 -j ACCEPT
```

The DB is on the same machine as web-server and thus it does not matter whether I use INPUT chain or OUTPUT chain. One more thing I have to look for is how the web-application is logging into the MySQL. If there is **localhost** being used, my rules will not work. Why ? Because localhost is a special value that means "use a Unix socket" to the MySQL client library and therefore it is a bypass of my tcp rules.

I solved this problem by creating virtual interface eth1:2 and configured my web application to use it.

```
root@master2:/# ifconfig eth1:2 192.168.40.1 netmask ...
root@master2:/# ifconfig
...
eth1:2      Link encap:Ethernet  HWaddr fa:16:3e:db:d1:4f
```

```
inet addr:192.168.40.1  Bcast:192.168.40.255  ...
```

Now the byte counters will work. Source port 3306 represents the actual data and destination port 3306 represents database queries.

```
root@master2:/# iptables -L -n -v -x
Chain INPUT (policy ACCEPT 3557 packets, 678567 bytes)
    pkts bytes    target     source     -  destination
   47395 63352592 ACCEPT     0.0.0.0/0  ...  tcp spt:3306
   48194  7972091 ACCEPT     0.0.0.0/0  ...  tcp dpt:3306
...
```

After a straight-forward set-up of collectd iptables plugin I am able to see DB queries and DB data in a nice graph.

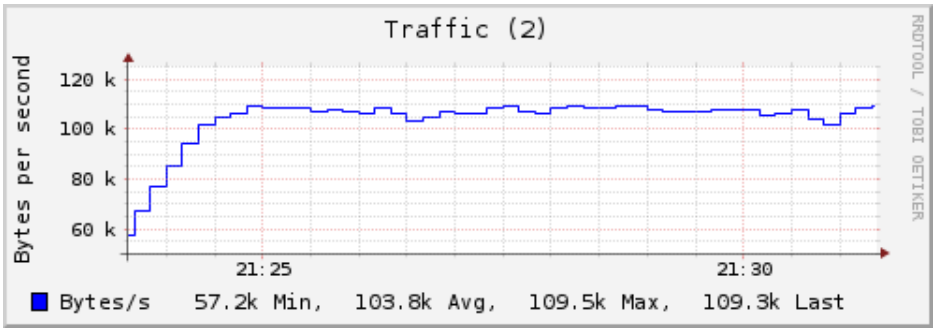

Figure 3.3: MySQL requests(DB queries) over time

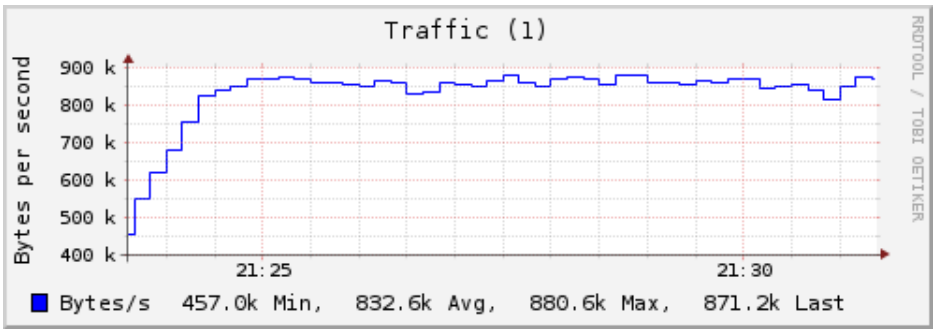

Figure 3.4: MySQL responses(DB data) over time

3.3.3 Client traffic

Client traffic is the traffic generated by clients who use the application over a period of time. It can be easily calculated 'per user'. What I mean by 'per user' is that I can easily predict how much traffic will a single user create. If I decide to include geographic regions into my predictions, I need to know geographic location of my

clients. The perfect tool for this job is **Awstats**. It analyzes server log files and is able to show where users are from. The output in percents can be used for further calculations.

3.4 Disk storage and IOPS

"Volume storage for Standard volumes is charged by the amount you provision in GB per month, until you release the storage. Volume I/O for Standard volumes is charged by the number of requests you make to your volume. Programs like IOSTAT can be used to measure the exact I/O usage of your system at any time. However, applications and operating systems often cache at different levels, so you may see a different number of I/O requests on your bill than is seen by your application."[11]

That is pretty much it! There is perhaps one more thing to add: Most cloud providers charge only for **additional** volumes one might want to mount to an instance. If storage capacity of the instance is sufficient there is no need to calculate cost of additional data storage + IOPS. The difference is: If the instance is terminated, all data saved on the instance is lost, whereas data on additional volume is preserved.

I will, therefore, assume that if customer wants to run DB on the cloud instance, he will pay for additional volume to secure his data from crash. On the other hand, if the only thing he wants to run in the cloud is nginx web-server + php then instance storage will suffice.

I can use Collectd **disk** plugin to measure both needed disk storage and IOPS.

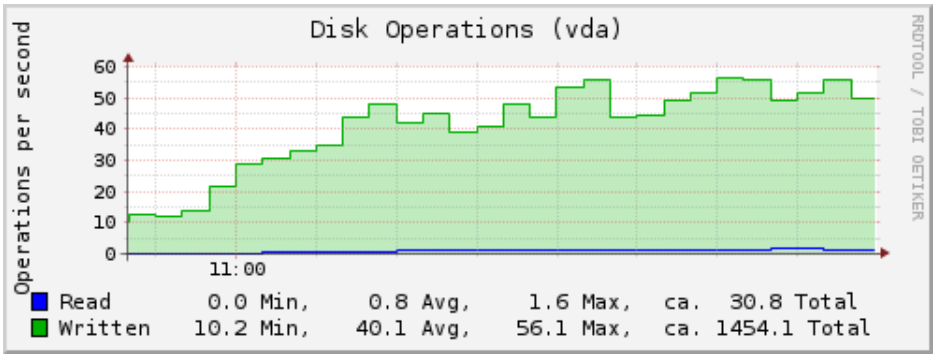

Figure 3.5: Disk IOPS over time shortly after loadtest start

3.5 Performance

Performance of a web application is often directly connected with user experience. How should one measure user experience ? A simple approach is to measure response times and decide whether they are reasonable.

3.5.1 Apdex score

Apdex score is a measure of user satisfaction with application's response time[5]. Apdex is used by todays most famous cloud based load testing tools such as: NewRelic, Blazemeter... . This method uses three performance zones:

1. **Satisfied**
2. **Tolerating**
3. **Frustrated**

With these three zones defined, the apdex score is calculated like this:

$$Apdex = \frac{Satisfied\_Count + \frac{Tolerating\_Count}{2}}{Total\_Samples} \tag{3.1}$$

It is clear that two separating thresholds **T** and **F** need to be defined[5]:

$$Satisfied = \text{zero to T} \tag{3.2}$$
$$Tolerating = \text{greater than T to F} \tag{3.3}$$
$$Frustrated = \text{greater than F} \tag{3.4}$$

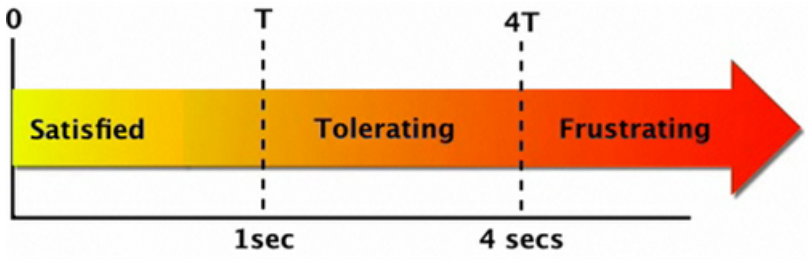

Figure 3.6: It is also recommended to define the F threshold to be $F = 4T$.[14][17]

Response times can be collected over a long period of time or they can be the output of a well designed load test. I will use load-testing, although the long period collection is the better way to go. Load tests should be designed as explained in 2.4.1.

Keep in mind that the interpretation of apdex score should make sense. For instance: If I run 20 minute load test and all of my response times will be in the satisfied zone apart from single one being in the tolerance zone, I will get for example $A = 0.999456...$ instead of $A = 1$ which would translate to all users being satisfied. In such case making a statement that the application with $A = 1$ performs somehow better is wrong. The data set acquired from 20 minute load test is too small to dwell on number precision.

Chapter 4

Decision Method

At this point it should not surprise you that my decision method will be based on the calculation of overall cost of each set-up and choosing the one with the lowest cost. The whole idea is to assume the costs of replication and remote DB access by analyzing chosen system metrics as described in the previous chapter.

4.1 Definition

Basic definition of cost function is:

$$COST_T = INSTANCE_T + TRAFFIC_T + STORAGE_T + PERFORMANCE_T \tag{4.1}$$

Let me break it down and explain the variables on AWS example.

4.1.1 Instance

$INSTANCE_T$ variable represents how much will I have to pay if I use specific instance for specific amount of time T. Let me calculate instance cost of **m3.medium** instance, which amazon offers for $0.070 per Hour.

$$M3.MEDIUM_{month} = 0.070 \cdot 24 \cdot 30 = \$50.4 \tag{4.2}$$

4.1.2 Traffic

$TRAFFIC_T$ variable represents cost of outgoing and incoming traffic during specific amount of time. One should use modified cost function if the billing strategy is different. For instance: Amazon charges only for outgoing traffic.

$$TRAFFIC_T = USER\_TRAFFIC_T + DB\_TRAFFIC_T \tag{4.3}$$

$USER\_TRAFFIC_T$ simply means traffic generated by users using the application over time and $DB\_TRAFFIC_T$ is the traffic generated by either replication or DB requests and responses over the internet.

Modified traffic cost function can be defined as:

$$TRAFFIC_T = TRAFFIC\_OUT_T + TRAFFIC\_IN_T \tag{4.4}$$

4.1.3 Storage

It is obvious that cloud providers charge for data storage. Usually the cost consists of charges for stored data and the cost of input/output operations.

$$STORAGE_T = STORAGE\_SIZE \cdot STORAGE_{COST} + IOPS_T \cdot IOPS_{COST} \tag{4.5}$$

4.1.4 Performance

As I have already mentioned the user experience has significant impact on how popular the web application is; hence, it has direct impact on possible profit. I will, therefore, define the performance costs using the apdex score explained in 3.5.1:

$$PERFORMANCE_T = (1 - APDEX) \cdot PENALTY_T \cdot USERS \tag{4.6}$$

Suppose the team leader of the marketing division informed me that each frustrated user represents 1 dollar loss of profit per day, this leads me to define $PENALTY_{day} = 1\$$.

$$PERFORMANCE_{month} = (1 - APDEX) \cdot 1 \cdot 30 \cdot USERS \tag{4.7}$$

I should define $PENALTY_{day}$ to be even higher considering that a frustrated user might never return to use my application. Even small difference can have major influence on resulting cost, hence, this variable should be well-defined and well thought over.

4.2 Further specification

To further simplify the cost functions, I will assume some variables according to real system metrics.

As explained in 3.3.1, the 'replication' DB traffic cost function for 30 days can be redefined:

$$REPL\_TRAFFIC_{30} \approx LOG\_POS_s \cdot BYTE_{PRICE} \cdot (30 \cdot 24 \cdot 60 \cdot 60) \cdot SLAVES_N \tag{4.8}$$

Where LOG_POS_s represents change of bin-log position per second. If I decide to add multiple slaves, I surely need to multiply the whole equation by number of slaves: $SLAVES_N$.

On the other hand remote DB traffic can be assumed like this(Please see 3.4):

$$DB\_TRAFFIC_{30} \approx IPTABLE\_RULE_s \cdot BYTE\_PRICE \cdot (30 \cdot 24 \cdot 60 \cdot 60) \tag{4.9}$$

Here $IPTABLE\_RULE_s$ is counter change of defined ip-table rule for either outgoing or incoming traffic. Result is clearly either outgoing traffic or incoming traffic.

Next I should assume the storage cost of replication scenario(see 3.4). I don't need to redefine equation 4.5, although I should think it over. If I replicate data between the servers, I can expect the data in the cloud to be roughly the same size and if the user load is similar, I expect IOPS to be similar. At this point please see 3.4 in order to understand why $IOPS_T$ is the hardest variable to assume and can be inaccurate if calculated from the results of short load test and transformed into one month assumption. Thanks to pricing policy of most cloud providers, $IOPS_T$ variable does not play major role in resulting cost, although it should not be left out of an equation completely. If I do not replicate, I can cross the $STORAGE_T$ variable out.

$PERFORMANCE_T$ mostly depends on hardware and server set-up. The only case I can assume $PERFORMANCE_T$ is if I replicate DB to a server with similar system specifications and similar caching set-up. If it is not the case I can only make an educated guess from app observation and common sense i.e. calculating possible response times from number of DB queries per action...

4.2.1 Revision

Taking all of the above mentioned information into account I can design a new cost function for each set-up:

$$\begin{aligned} REMOTE\_COST(T) \approx & INSTANCE_T + USER\_TRAFFIC_T \\ & + DB\_TRAFFIC_T \ + PERFORMANCE_T \end{aligned} \tag{4.10}$$

$$\begin{aligned} REPL\_COST(T) \approx & INSTANCE_T + USER\_TRAFFIC_T + STORAGE_T \\ & + REPL\_TRAFFIC_T + PERFORMANCE_T \end{aligned} \tag{4.11}$$

4.3 E-commerce website

I will verify the profiling and decision method on a sample application by collecting specified metrics 3.2 and creating a monthly cost graph with respect to number of users.

The very first application I am going to profile is regular e-shop. Armed with Jmeter load testing tool, Collectd metric collection program and equations 4.11, 4.10, I am able to calculate the monthly charges of either set-up.

Load tests are designed the way explained in 2.4.1. In case of the e-shop, there are 2 step-by-step scenarios, which the Jmeter does: **Buyer** and **Buddy**. The buyer obviously buys stuff, therefore, he does all the actions one has to do to purchase something from the shop namely: **buy**, **fill-in-form**, **pay** and **confirm**. The buddy only clicks

around, hence, his actions are: browse **home page**, choose random **category**, **tag search** and **random product**.

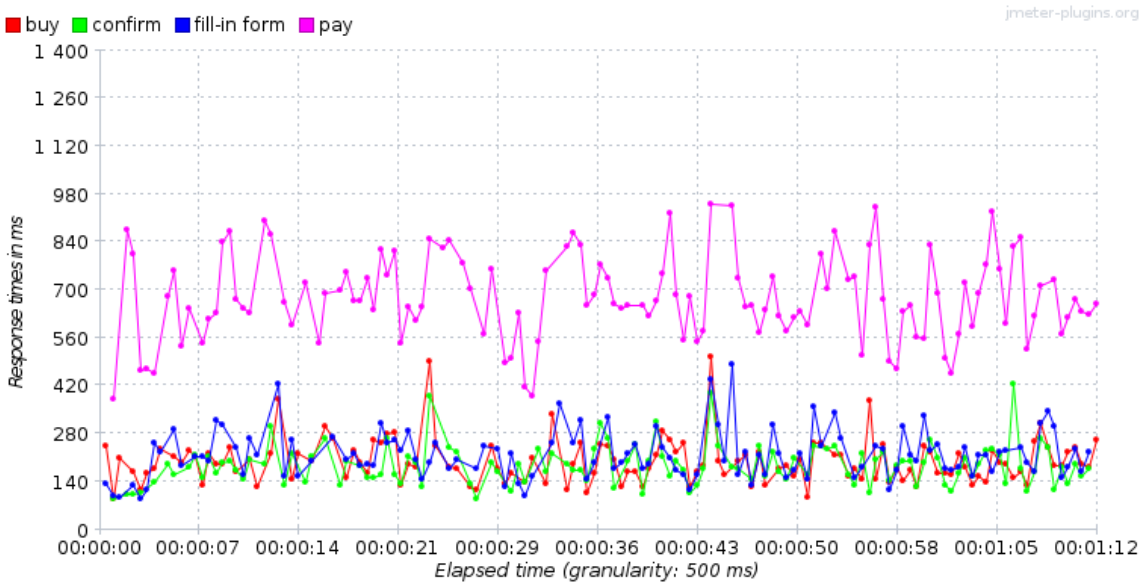

Figure 4.1: Load test result shows response times of buyer's actions

In my case, customers are always: 10 % buyers and 90 % buddies. It translates into 10 concurrent users being equal to 1 buyer plus 9 buddies, 20 = 18+2, 30 = 27+3 and so on. First off, I need to distinguish between 'concurrent' and 'simultaneous'. They are normally very similar terms but in load testing they have different meanings. Simultaneous means two or more requests at the same time. Concurrent is two or more threads running in parallel.

I assume 100GB storage space to be sufficient and I define pricing policy according to AWS to be:

$\mathbf{INSTANCE_{hour}}$	**Traffic_IN/GB**	**Traffic_OUT/GB**	**STORAGE**	$\mathbf{STORAGE_{GB/month}}$	**1M of IOPS**
$0.070	*$0*	*$0.12*	*100GB*	*$0.05*	*$0.05*

Table 4.1: Defined pricing policy

With all the data available I am able to calculate the resulting cost and draw an assumption graph. I left $PERFORMANCE_T$ out of the equations in this case.

Empirical evidence suggests that cost linearly depends on number of users. Hence, I have calculated final $COST(USERS)$ functions by linear regression.

$$COST_{REPLICATION}(USERS_N) = 0.45 \cdot USERS_N + 56.85 \tag{4.12}$$

$$COST_{REMOTE}(USERS_N) = 1.03 \cdot USERS_N + 54.22 \tag{4.13}$$

USERS	INSTANCE_T	$\text{TRAFFIC\_IN}_\text{T}$	$\text{TRAFFIC\_OUT}_\text{T}$	STORAGE_T	Σ
10(9+1)	50.40	0.00	4.48	5.00+2.05	$61.93
20(18+2)	50.40	0.00	8.67	5.00+2.26	$66.32
30(27+3)	50.40	0.00	12.61	5.00+2.62	$70.63
40(36+4)	50.40	0.00	16.89	5.00+2.75	$75.03
50(45+5)	50.40	0.00	19.51	5.00+2.97	$77.88
60(54+6)	50.40	0.00	21.65	5.00+3.41	$80.46
70(63+7)	50.40	0.00	24.60	5.00+3.80	$83.79

Table 4.2: Replication set-up costs for $T = 30$ *days*.

USERS	INSTANCE_T	$\text{TRAFFIC\_IN}_\text{T}$	$\text{TRAFFIC\_OUT}_\text{T}$	STORAGE_T	Σ
10(9+1)	50.40	0.00	12.61	5.00+2.05	$63.01
20(18+2)	50.40	0.00	24.77	5.00+2.26	$75.17
30(27+3)	50.40	0.00	36.52	5.00+2.62	$86.92
40(36+4)	50.40	0.00	48.23	5.00+2.75	$98.63
50(45+5)	50.40	0.00	58.21	5.00+2.97	$108.61
60(54+6)	50.40	0.00	65.22	5.00+3.41	$115.62
70(63+7)	50.40	0.00	72.53	5.00+3.80	$122.93

Table 4.3: Remote DB set-up costs for $T = 30$ *days*.

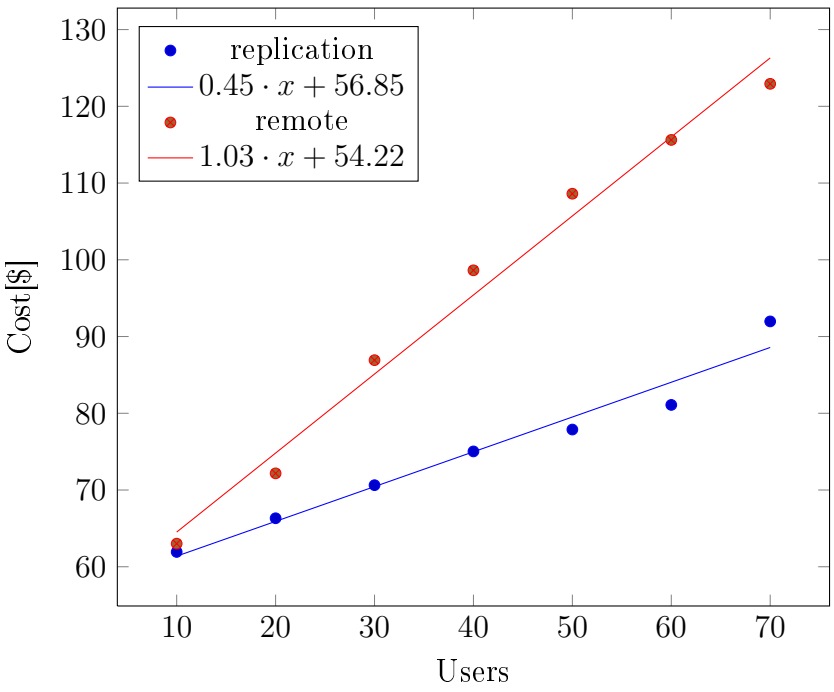

Figure 4.2: Cost assumptions for $T = 30$ *days*

It seems that replicating data between sites is the more cost-effective choice. Why is it so ? As I have already mentioned the **data movement** or traffic is the decisive factor. In this case it is represented by only $TRAFFIC\_OUT_T$. Consider the following DB queries:

```
SELECT * FROM products WHERE...
INSERT INTO orders...
SELECT * FROM chat WHERE...
...
```

In case of replication only the **INSERT** statement will be replicated(statement bin-log format) or affected rows(row bin-log format). On the other hand, if I am not replicating, I need to send all the queries and I should not forget about the fact that only 10% of my users do 'write' actions.

4.3.1 Latency vs. Performance

I did not use $PERFORMANCE_T$ variable in the previous assumption. How might the performance affect the resulting cost ? What will happen if my web-server and DB-server are '50 or more milliseconds apart' ?

To answer the above-mentioned questions, I will first introduce latency into my testing environment. By issuing the following command on both servers, I set up delay to be normally distributed with $\mu = 57.4ms$ and $\sigma = 7.8ms$.

```
# tc qdisc change dev eth1 root netem delay 57.4ms 7.8ms\
    distribution normal
```

I added one more interface **eth1** to each server for DB connection, to make sure that I do not directly delay users of my application(using **eth0**). Delay at both sides sums up to round trip delay(or **Latency**), which also has normal distribution($\mu = 114.8ms$, $\sigma = 15.6ms$). For this particular set-up I took intra region New York to Rio De Janeiro latency data from [18].

It would be a mistake to think that $115ms$ latency will add $115ms$ to app's response times. It turns out that the number of DB queries per user's action will be the crucial factor. If the application queries DB synchronously i.e. the application waits for every single DB response, the time to process all the DB queries per single page/action is:

$$TIME \approx (PREPATE\_ST + EXECUTE\_ST + LATENCY) \cdot QUERIES_N \tag{4.14}$$

My e-shop queries DB synchronously. With that being said, I can assume apdex from response times and number of DB queries per action. By inspecting my application's source code or DB logs, I am able to count $QUERIES_N$. Jmeter has nicely formatted csv output of load test results. All I need to do is: Use response times grouped by action, add $+QUERIES_N \cdot \mu\_latency$ and calculate apdex.

Action	$\mathbf{QUERIES_N}$	Time assumption
Home Page	24	$t_r + 24 \cdot \mu\_latency$
Random product	32	$t_r + 32 \cdot \mu\_latency$
Category	27	$t_r + 27 \cdot \mu\_latency$
Tag Search	24	$t_r + 24 \cdot \mu\_latency$
Fill-in Form	48	$t_r + 48 \cdot \mu\_latency$
Buy	31	$t_r + 31 \cdot \mu\_latency$
Pay	60	$t_r + 60 \cdot \mu\_latency$
Confirm	30	$t_r + 30 \cdot \mu\_latency$

Table 4.4: Response time assumption, t_r represents measured response time

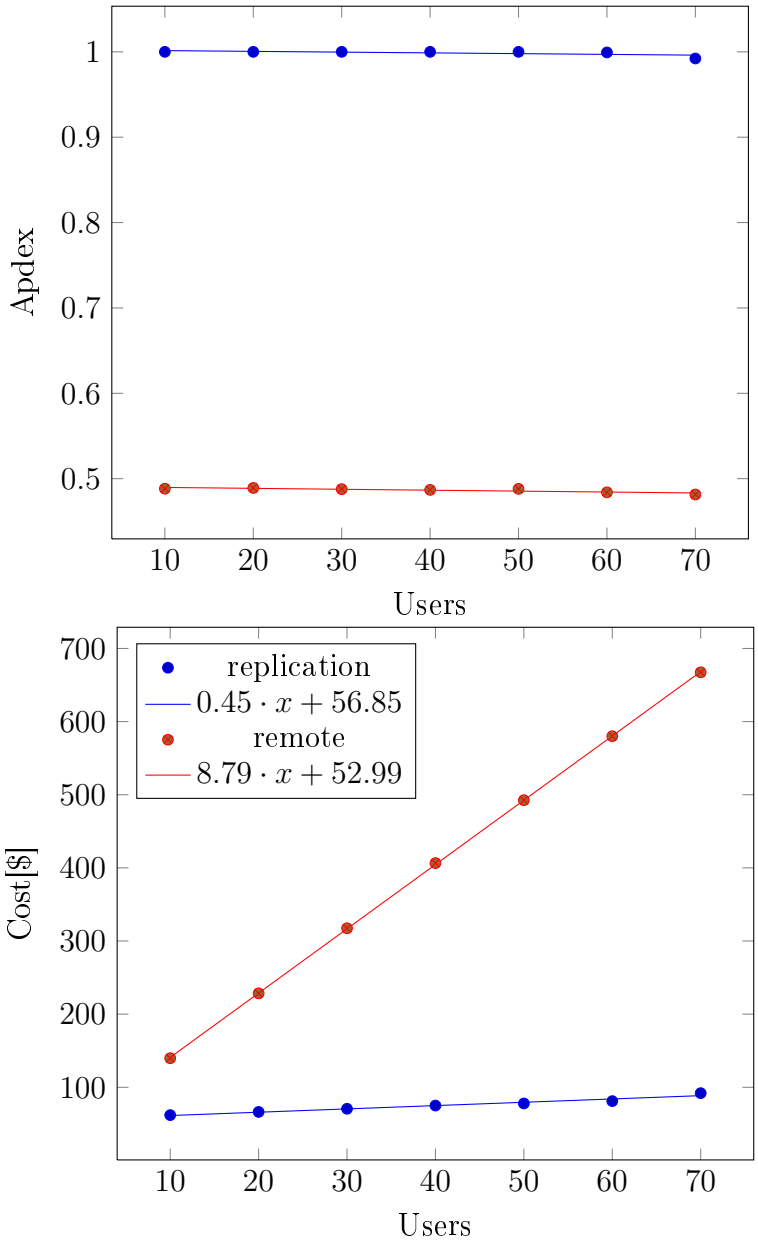

Figure 4.3: Apdex assumption for $\mu\_latency = 114.8ms$ and final cost assumptions for $T = 30$ *days* with $PENALTY_{day} = \$0.5$.

4.3.2 Storing bigger data

What would happen if I needed to store more than 100 GB of data? In today's world, with emerging technologies like big data and seemingly infinite storage capabilities, this is an obvious question.

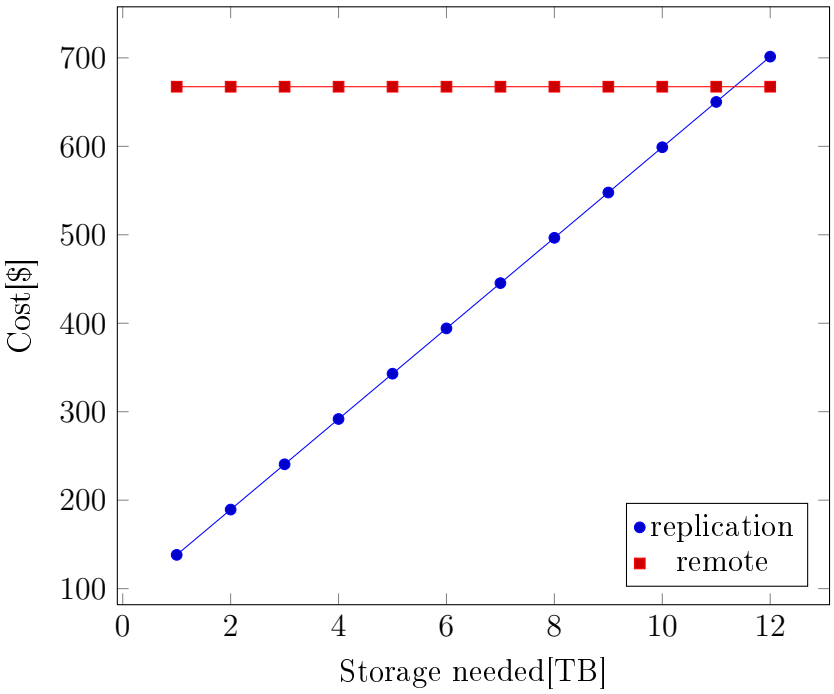

Figure 4.4: Cost assumptions for $T = 30$ *days* and $USERS = 70$.

If I needed to store terabytes of data, the remote-DB access might shine. In a highly unlikely scenario where number of my clients is quite stable and I do not expect to gain more clients. It is because of the absence of cloud storage costs $STORAGE_T$ in remote-DB equation. However, I still believe that the marketing division would be upset with performance of application if half of the clients were frustrated.

4.3.3 Conclusion

By inspecting graphs on the previous page I can conclude that my model behaves according to the definition. The higher the latency the bigger impact on user's satisfaction and profit, therefore the cost of set-up should also be higher. It is clear that if my application access DB remotely, it's apdex score will go down significantly, because of synchronous DB queries. Taking this into consideration, I can safely say that replication is the way to go in this case.

4.4 Community blog

I will repeat the above mentioned process on community blog powered by **Wordpress**. To make it a bit more interesting, I will use **RBR** this time i.e.: I set MySQL to use **row** format for a bin-log:

```
binlog-format = "ROW"
```

I expect the resulting cost to be higher in the replication scenario, because using RBR will result in bigger bin-log and therefore bigger replication traffic.

This time my load test user scenario has only single group called "**Blogger**", unlike in the previous application I did not design my load test to create 90% read requests and 10% write requests. All bloggers do both read and write actions. Blogger's actions are timed the following way:

home page	**category**	**form submit**	**comment**	**tag search**	**browse post**
2 sec	2 sec	5 sec	4 sec	2 sec	2 sec

Table 4.5: Blogger scenario: actions and timing.

Unfortunately, Wordpress also queries DB synchronously and therefore executes queries one by one. With that in mind, one would expect the replication scenario to be the one to choose again.

Wordpress has many useful plugins like **Query Monitor**, which can be used to determine number of DB queries for each action. I decided to use simpler approach and added the following code snippet into theme's **index.php** file.

```
<?php echo get_num_queries(); ?>
```

The results are:

home page	**category**	**form submit**	**comment**	**tag search**	**browse post**
20	30	34	34	26	33

Table 4.6: Blogger scenario: actions and DB queries.

The pricing policy is almost the same as the one I used in the eCommerze application. However, I decided to change the following variables:

- Needed storage capacity: $\mathbf{STORAGE} = 1TB$
- Profit loss for single user per day: $\mathbf{PENALTY_{day}} = \$0.2$

I am planning to use 1 TB disk and I suppose that my blog is not making money, hence, I only want small penalty if my visitors get frustrated with blog's performance.

USERS	INSTANCE_T	$\text{TRAFFIC\_OUT}_\text{T}$	STORAGE_T	APDEX	$\text{PERFORMANCE}_\text{T}$	Σ
10	50.40	5.40	51.20 + 8.72	0.993	0.41	$116.13
20	50.40	9.10	51.20 + 12.79	0.88	14.35	$137.85
30	50.40	9.58	51.20 + 12.43	0.753	44.37	$167.98
40	50.40	9.73	51.20 + 12.82	0.670	78.98	$203.13

Table 4.7: Replication set-up costs for community blog. $T = 30$ *days*.

USERS	INSTANCE_T	$\text{TRAFFIC\_OUT}_\text{T}$	STORAGE_T	APDEX	$\text{PERFORMANCE}_\text{T}$	Σ
10	50.40	11.23	0	0.5	30	$91.63
20	50.40	19.06	0	0.5	60	$129.46
30	50.40	20.08	0	0.499	90.06	$160.53
40	50.40	20.85	0	0.488	123.74	$194.99

Table 4.8: Remote DB set-up costs for community blog. $T = 30$ *days*.

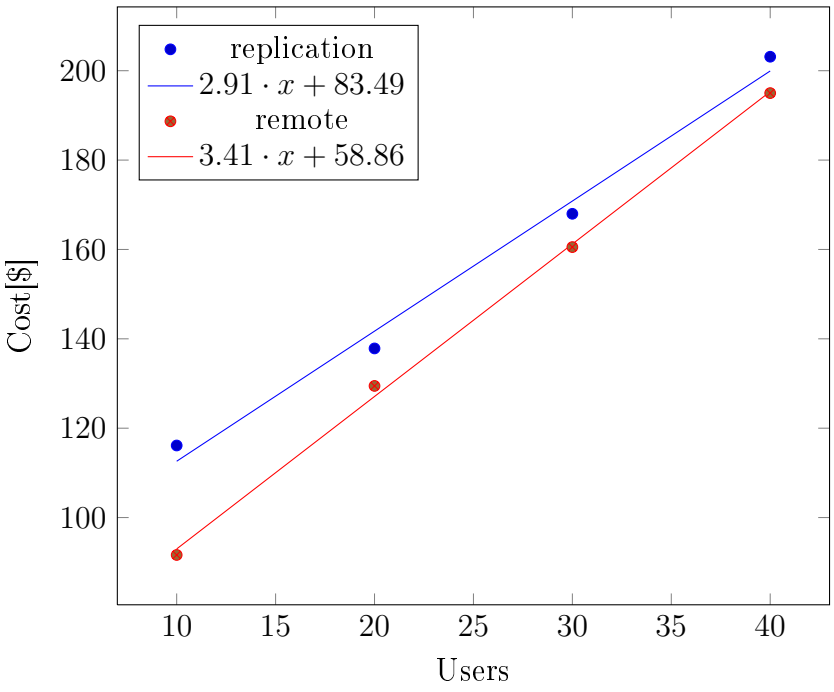

Figure 4.5: Cost assumptions for $T = 30$ *days*

$$COST_{REPLICATION}(USERS_N) = 2.91 \cdot USERS_N + 83.49 \tag{4.15}$$

$$COST_{REMOTE}(USERS_N) = 3.41 \cdot USERS_N + 58.86 \tag{4.16}$$

What is the effect of RBR on the traffic? **In this case**, there is no significant traffic difference between replication using **STATEMENT** format and replication using **ROW** format for a bin-log. It is because none of "blogger's" actions affect numerous rows and therefore both cases are very similar in terms of traffic.

4.4.1 Conclusion

In spite of setting up "frustrated user penalty" for bad performance, it seems that remote database is the way to go in this case. Why is it so? There are 2 reasons:

1. I did not divide my users to do 90% read actions and 10% write actions like I did in the previous example. Hence, outgoing traffic is now significant even in the replication set-up. However, the traffic difference between replication and remote access still remains: Replication needs to transfer only "changes" to the database, whereas remote access needs to transfer everything.

2. Wordpress is quite slow and 40 users doing read and write actions apparently ”cross the limit line”. Therefore, app receives bad apdex score even in the replication set-up.

Chapter 5

Summary

The main goal of this thesis was a method to decide, whether it is more economical to run an application in multiple locations in hybrid cloud mode with all database accesses going through an internet line, replicate the database between locations, or when hybrid cloud is not economical at all.

I completely focused on decision between the replication scenario and the remote DB scenario. The main reason for this is that the decision between using the hybrid cloud or not using the hybrid cloud is very different and it does not share the same metrics. Therefore, I decided to leave the third option of not using hybrid cloud mode for further research.

I discussed a cost prediction method that allow me to decide between either keeping DB home or replicating DB into the cloud. Furthermore, I provided a complete walk-through and explanation of metrics, variables, principles and ideas my method is based on.

My intention to keep the method simple and understandable in a matter of minutes was achieved. However, the method is not a step-by-step "cookbook" and therefore it can not be followed blindly without any form of modification. Special circumstances might require special tuning of equations or collection methods. With that being said, I firmly believe that I accomplished the main goal.

My work can surely be improved in the future. To name a few possible improvements, I would like to be able to predict costs for many different cloud set-ups, I would like to improve my performance penalty function, redesign my method to work with user trends and various database engines... The whole subject of Cloud computing is full of interesting ideas and it is well worth studying.

Chapter 6

Appendix

6.1 Load testing the eCommerce application

The following graph shows all the actions of 2 separate testing thread groups. I named them "**Buddy**" and "**Buyer**". The scenario has only 2 write actions **confirm** and **fill-in-form**

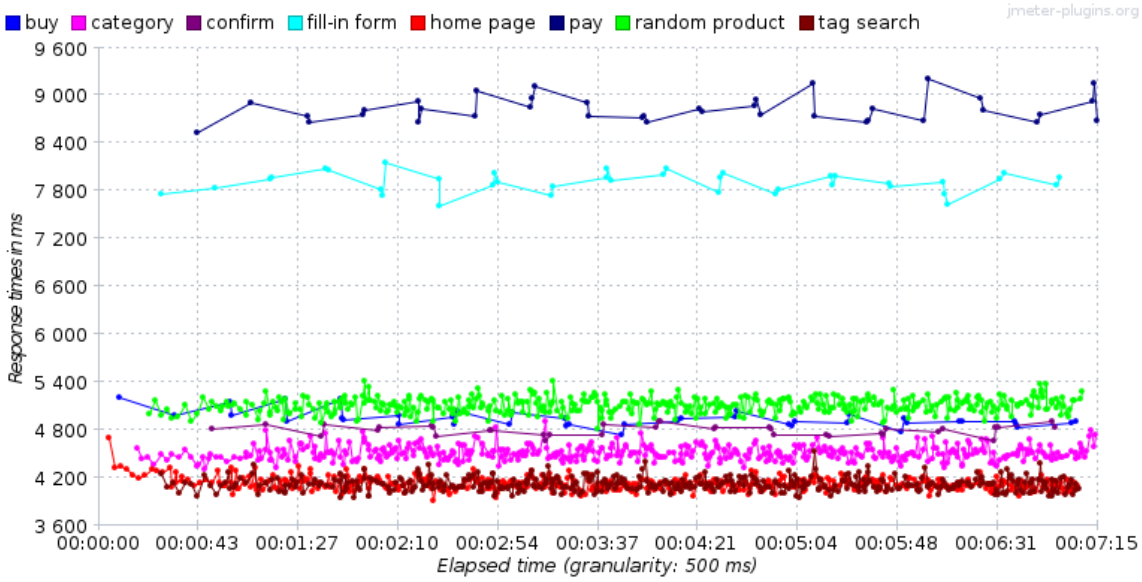

Figure 6.1: 50 user load-test of the eCommerce application with remote DB access and $\mu\_latency = 114.8ms$. Clearly, number of DB queries has major impact on performance. $QUERIES_N(PAY) = 60$ and $QUERIES_N(TAG\_SEARCH) = 24$

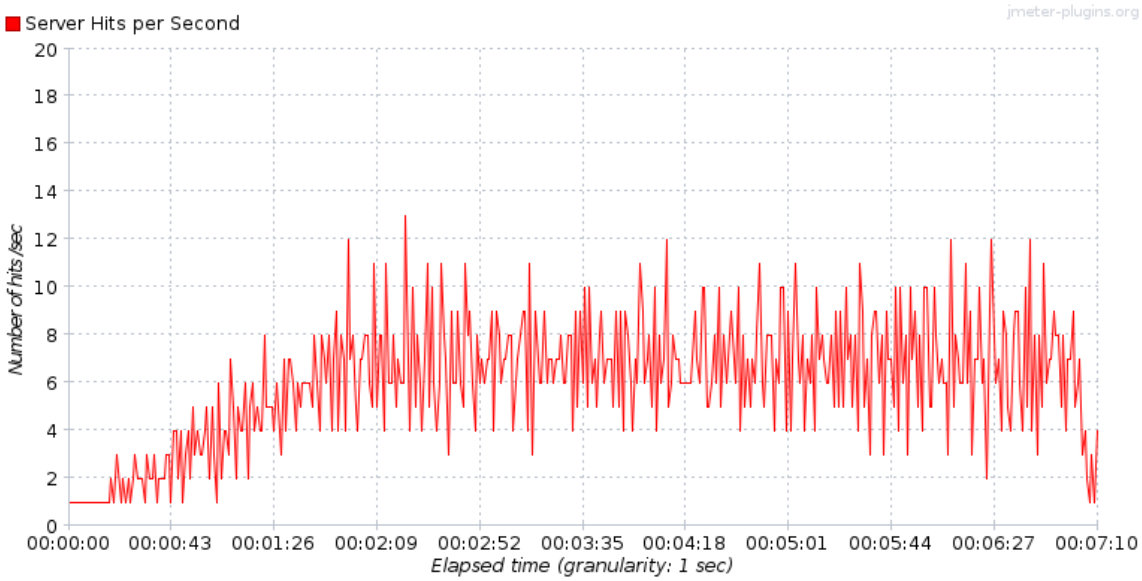

Figure 6.2: 50 user scenario equals max 14 hits/sec

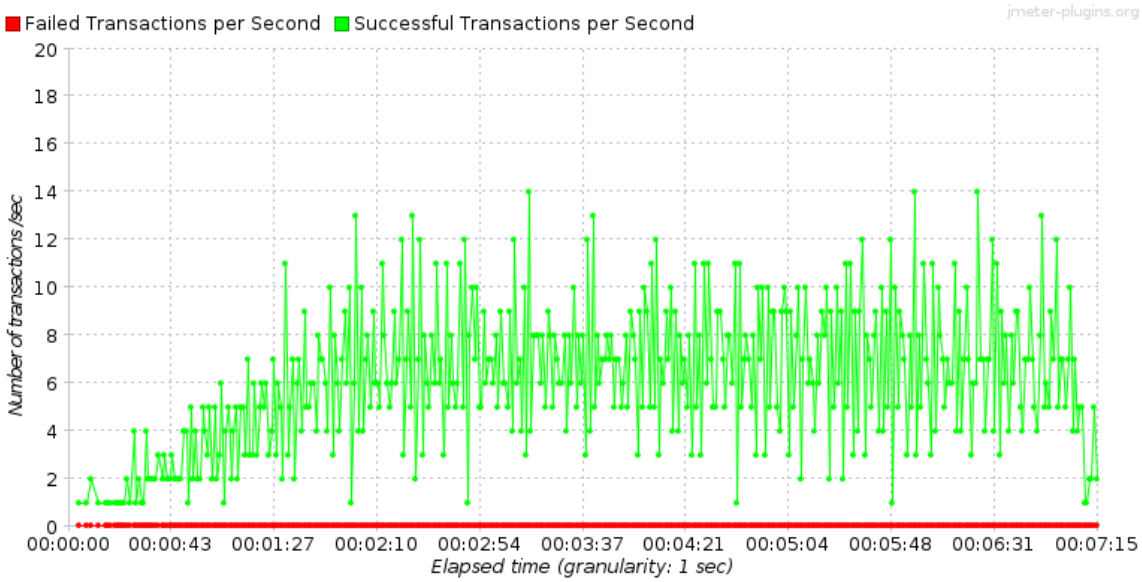

Figure 6.3: When all the transactions are successful, I can safely use the output values.

6.2 Load testing the community blog

The following graphs show one of the tests for community blog cost prediction. The very first condition of a successful test is that all the transactions were successful, meaning there was not a single failed attempt to submit a form or comment on blog post etc. .

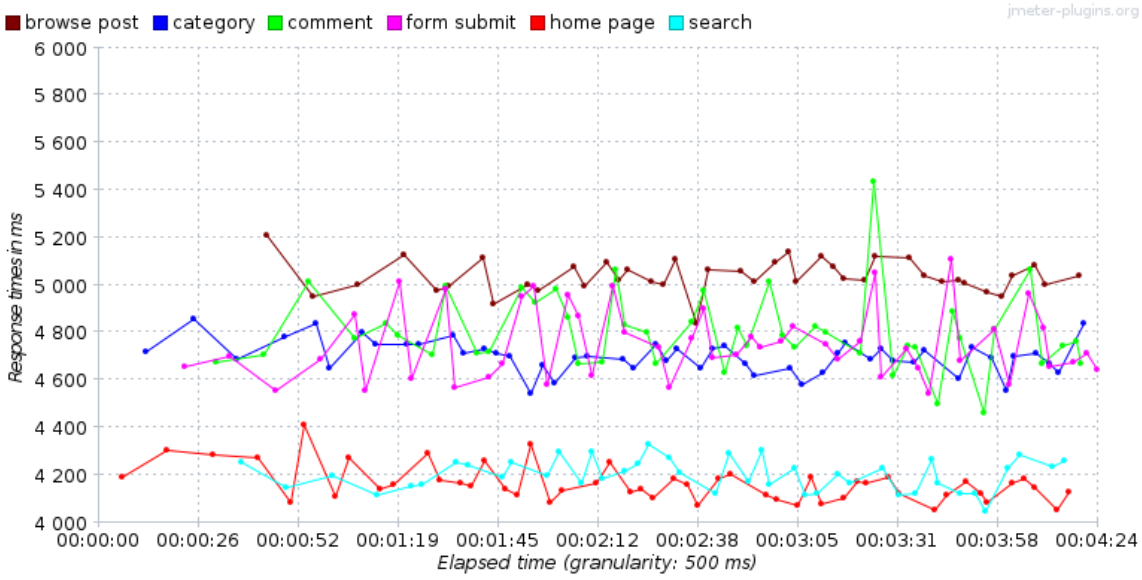

Figure 6.4: Load test of the community blog application with $\mu\_latency = 114.8ms$. $USERS = 10$

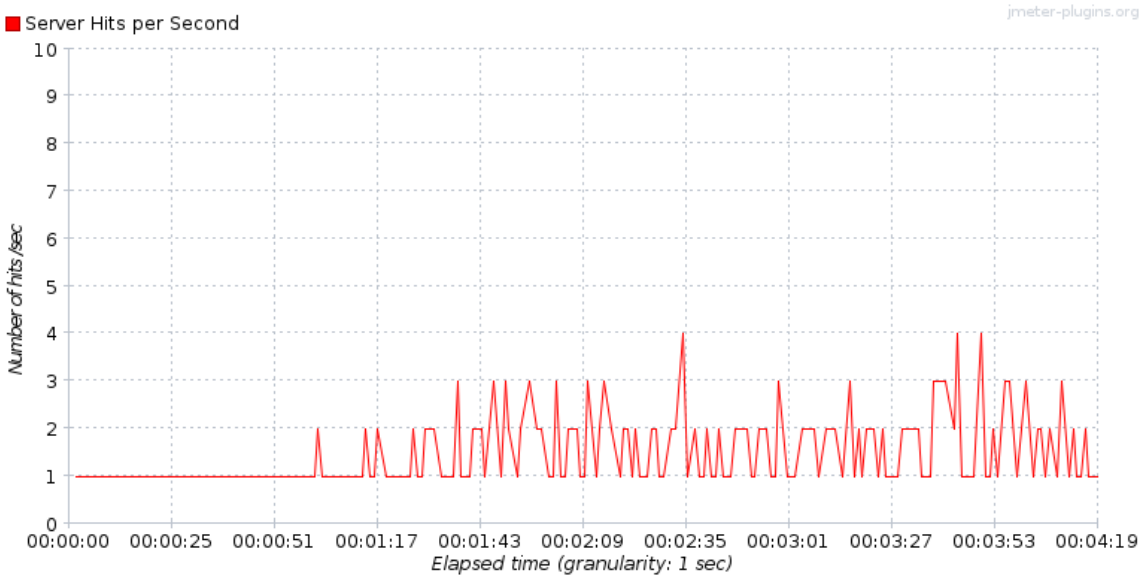

Figure 6.5: Blog load test: 10 users represent max 4 hits/sec

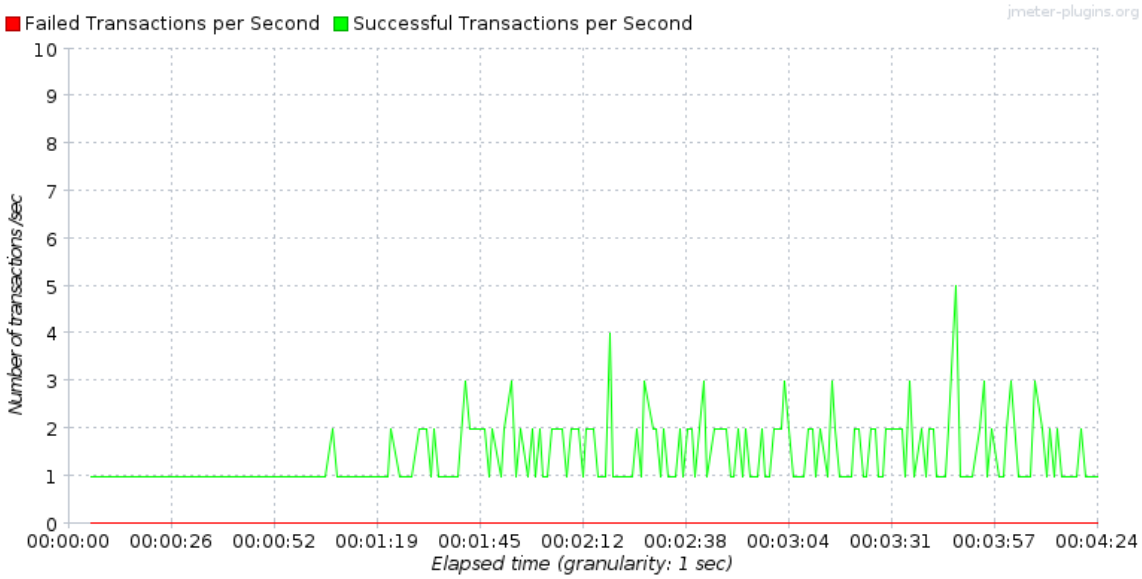

Figure 6.6: All the transactions were successful(the test was also successful) I can safely use the output values.

List of Figures

1.1 Problem definition. 1

2.1 Level of involvement comparison[13] 4
2.2 Private cloud running OpenStack as an IaaS 5
2.3 AWS(Amazon web services) as a public cloud 5
2.4 Hybrid cloud example 6
2.5 Synchronous replication. Inspired by [15] 7
2.6 MySQL semi-synchronous replication 7
2.7 MySQL asynchronous replication 8
2.8 Master/Slave setup, application running on 192.168.2.4 10
2.9 Response over time without *hyperdb* 12
2.10 Response over time with *hyperdb* 12
2.11 Multi master set up 13

3.1 Collectd over the network example 16
3.2 Log position over time with noticable ramp-up 18
3.3 MySQL requests(DB queries) over time 20
3.4 MySQL responses(DB data) over time 20
3.5 Disk IOPS over time shortly after loadtest start 21
3.6 Threshold definition recommendation 22

4.1 Load test result shows response times of buyer's actions 26
4.2 Cost assumptions for $T = 30$ *days* 27
4.3 Apdex and cost predictions 29
4.4 Cost assumptions for $T = 30$ *days* and $USERS = 70$. 30
4.5 Cost assumptions for $T = 30$ *days* 32

6.1 Load test of the eCommerce app with 50 users 35
6.2 50 user scenario equals max 14 hits/sec 36
6.3 When all the transactions are successful, I can safely use the output values. 36
6.4 Load test of the community blog with 10 users 37
6.5 Blog load test: 10 users represent max 4 hits/sec 37
6.6 All the transactions were successful(the test was also successful) I can safely use the output values. 38

List of Tables

4.1 Defined pricing policy 26
4.2 Replication set-up costs for $T = 30$ *days*. 27
4.3 Remote DB set-up costs for $T = 30$ *days*. 27
4.4 Response time assumption, t_r represents measured response time . . . 29
4.5 Blogger scenario: actions and timing. 31
4.6 Blogger scenario: actions and DB queries. 31
4.7 Replication set-up costs for community blog. $T = 30$ *days*. 32
4.8 Remote DB set-up costs for community blog. $T = 30$ *days*. 32

Bibliography

[1] ALLSPAW, John. The art of capacity planning: scaling web resources. " O'Reilly Media, Inc.", 2008.

[2] BRADFORD, Ronald; SCHNEIDER, Chris; CALERO, Nelson. Effective MySQL: Replication Techniques in Depth. McGraw-Hill/Osborne, 2012.

[3] SOUDERS, Steve. High-performance web sites. Communications of the ACM, 2008, 51.12: 36-41.

[4] SOUDERS, Steve. Even faster web sites: performance best practices for web developers. O'Reilly Media, 2009.

[5] SEVCIK, Peter. Defining the application performance index. Business Communications Review, 2005, 20.

[6] GUNTHER, Neil J. Analyzing Computer System Performance with Perl:: PDQ. Springer, 2011.

[7] MENASCE, Daniel A., et al. Performance by design: computer capacity planning by example. Prentice Hall Professional, 2004.

[8] LENTZ, Arjen, et al. High Performance MySQL: Optimization, Backups, Replication, and More. O'Reilly Media, Incorporated, 2008.

[9] ORACLE. MySQL 5.6 Reference Manual [online]. 38742. vyd. 2014, 16.5. [cit. 2014-05-18]. Available at: dev.mysql.com/doc/refman/5.6/en/index.html

[10] ROUSE, Margaret. What is SPI (SaaS, PaaS, IaaS)?. Search Cloud-Computing, Techtarget [online]. 2012 [cit. 2014-05-10]. Available at: searchcloudcomputing.techtarget.com/definition/SPI-model

[11] Amazon EBS Pricing. Amazon Web Services [online]. 2014 [cit. 2014-05-10]. Available at: aws.amazon.com/ebs/pricing

[12] Database replication. ORACLE. Oracle Documents [online]. Release 8.0 A58227-01. 1997 [cit. 2014-05-10]. Available at: docs.oracle.com/cd/A64702_01/doc/server.805/a58227/ch_repli.htm

[13] Who Manages Cloud IaaS, PaaS, and SaaS Services. My Cloud Computing Blog [online]. 2013 [cit. 2014-05-10]. Available at: mycloudblog7.wordpress.com/2013/04/19/who-manages-cloud-iaas-paas-and-saas-services/

[14] HARZOG, Bernd. New Relic RPM Product Review. The Virtualization Practice [online]. 2009 [cit. 2014-05-10]. Available at: www.virtualizationpractice.com/new-relic-rpm-product-review-2588/

[15] LOMNICKI, Michal. Database replication explained. X8 Software Development Blog [online]. 2013 [cit. 2014-05-10]. Available at: x8.io/blog/2013/05/database-replication-explained/

[16] ZAITSEV, Peter. Automation: A case for synchronous replication. MySQL performance blog [online]. 2012 [cit. 2014-05-10]. Available at: www.mysqlperformanceblog.com/2012/09/19/automation-a-case-for-synchronous-replication/

[17] NIELSEN, Jakob. Response Times: The 3 Important Limits. Nielsen Norman Group: Evidence-Based User Experience Research, Training, and Consulting [online]. 1993, January 1 [cit. 2014-05-18]. Available at: www.nngroup.com/articles/response-times-3-important-limits/

[18] Global Network Latency Averages. AT&T. AT&T Global IP Network [online]. 2014 [cit. 2014-05-18]. Available at: ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html

[19] BROUWER, Peter. The Art of Data Replication. ORACLE. Oracle Technical White Paper [online]. 2011, September [cit. 2014-05-18]. Available at: http://www.oracle.com/technetwork/articles/systems-hardware-architecture/o11-080-art-data-replication-487868.pdf

[20] SCHNEIDER, Chris. Comparing MySQL Statement Based and Row Based Replication. *Database Journal* [online]. 2011 [cit. 2014-05-10]. Available at: www.databasejournal.com/features/mysql/article.php/3922266/Comparing-MySQL-Statement-Based-and-Row-Based-Replication.htm

[21] SCHNEIDER, Chris. Working with MySQL Multi-master Replication - Keeping a True Hot Standby. *Database Journal* [online]. 2010 [cit. 2014-05-10]. Available at: www.databasejournal.com/features/mysql/article.php/3896061/Working-with-MySQL-Multi-master-Replication—Keeping-a-True-Hot-Standby.htm

DVD contents

File/Folder	Description
test_scenarios	folder containing testing user scenarios for each tested application: **eshop.jmx**, **wordpress.jmx**
confs	folder containing server configuration files, DB sql files and zipped www apps: **collectd.conf**, **master1.cnf**, **master2.cnf**...
eshop_results	folder containing numbered subfolders with load test results: **response.csv**, **transactions.csv**...
blog_results	folder containing test outputs from blog testing: **response.csv**, **transactions.csv**...
30	folder containing example of test result for 30 users + apdex calculation script in bash
blog.ods	open document file I used for blog costs calculation.
eshop.ods	open document file I used for eshop costs calculation.
thesis.pdf	PDF of my bachelor thesis