Scalability Guidelines for Software as a Service

Recommendations based on a case analysis of Quinyx FlexForce AB and UCMS Group Ltd.

MIKAEL RAPP

KTH Information and Communication Technology

Bachelor of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:107

Scalability Guidelines for Software as a Service

Recommendations based on a case analysis of
Quinyx FlexForce AB and UCMS Group Ltd.

Mikael Rapp
<mikrap@kth.se>

Bachelor thesis

2010.06.04

Examiner: Professor Gerald Q. Maguire Jr.
Industrial Supervisor: Henrik Mörner, UCMS Group ltd

Abstract

Software as a service (SaaS) has become a common business model for application providers. However, this success has lead to scalability issues for service providers. Their user base and processing needs can grow rapidly and it is not always clear how a SaaS provider should optimally scale their service.

This thesis summarizes the technological (both software and hardware) related solutions to scaling, but will also cover financial and managerial aspects of scalability. Unfortunately, there is no existing out of the box solution for managing scalability, every situation and application is currently viewed as a unique problem, but there exists a lot of good advice from many sources about scaling. Obviously there are practical solutions to scaling, as there are successful SaaS providers, but it is not clear if there exists some fundamental principles that every SaaS provider could use to address the issue of scalability. This thesis seeks to find such fundamental principles though previous research, articles and finally a case analysis of Quinyx FlexForce AB. The thesis concludes that there are many principles of scaling a 3-tier web system and that most principles can be applied by SaaS providers.

Sammanfattning

Software as a Service (SaaS) har blivit en allt vanligare lösning för företag. Detta har dock lett till skalbarhets problem för många leverantörer av SaaS. En SaaS leverantör kan få problem med skalning ifall deras användarbas eller beräkningsbehov växer för snabbt. Denna avhandling sammanfattar de tekniska (mjukvara och hårdvara) relaterade lösningar på skalning. Avhandlingen kommer även kortfattat att omfatta ekonomiska och administrativa aspekter av skalbarhet. Tyvärr finns det inga befintliga universallösningar för hantering skalbarhet utan varje situation och tillämpning måste ses som ett unikt problem. Det finns många goda råd från många källor om skalning och uppenbarligen finns det praktiska lösningar på att skala, då det finns framgångsrika SaaS leverantörer. Det är dock oklart om det finns några grundläggande principer som varje SaaS-leverantör kan använda för att underlätta skalbarhet. Avhandlingen syftar till att hitta sådana grundläggande principer och grundar sig på tidigare forskning, aktuella artiklar och avslutats med en analys av Quinyx FlexForce AB. Avhandlingen drar slutsatsen att det finns grundläggande principer som SaaS leverantörer kan tillämpa vid skalning av ett ”3-tier” webserver system.

Table of Content

Figures

Tables

Acronyms and Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ARP	Address Resolution Protocol
ASP	Active Server Pages
AWS	Amazon Web Services
CEO	Chief executive officer,
CGI	Common Gateway Interface
CPU	Central Processing Unit
CTO	Chief technical officer
DB	Database
DNS	Domain Name System
EMV	Expected Monetary Value
EC	Elastic Computing
ECC	Elastic Cloud Computing
EU	European Union
HR	Human resources
HIPPA	Health Insurance Portability and Accountability Act
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure Hyper Text Transfer Protocol
GHz	Gigahertz
IaaS	Infrastructure as a service
IIS	Internet Information Services
I/O	Input / Output
IP	Internet Protocol
IPV4	Internet Protocol Version 4
MMU	Memory Management Unit
MVA	Mean Value Analysis
PHP	Hypertext Preprocessor
QFAB	Quinyx FlexForce AB.
ISP	Internet Service Provider
R&D	Research and Development
RACI	Responsible, Accountable, Consulted, Informed (chart)
RAM	Random Access Memory
RIA	Rich internet application
SaaS	Software as a service
SLA	Service Level Agreement
SSL	Secure Socket Layer
SMS	Short Message Service
SQL	Structured Query Language
TCO	Total Cost of Ownership
TLS	Transport Layer Security
UCMS	UCMS group EMEA Ltd
US / USA	United States of America
VM	Virtual Machine, a running VI
VI	Virtual Image
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
QFAB	Quinyx FlexForce AB
XML	Extended Makeup Language

About this document

Target audience

The target audience for this thesis is those who want to better understand scalability issues of SaaS services and how to solve scalability problems using available technology. The thesis will also consider business viability, as it must be **economically feasible** for a SaaS provider to offer a service and for customers to contract for these services. The thesis will have a special focus on the 3-tier web server as well as on cloud computing.

To understand the technical aspects of this thesis, the reader should be familiar with the following concepts:

- Rich user internet applications (RIAs)
- Basic understanding of the concepts of technologies such as PHP, ASP, SQL, IIS, Apace, Flash, Silverlight, and JavaScript.

Economical aspects will also be covered, but the reader should note that economical details (such as prices, costs, etc.) quickly become obsolete due to the rapid development in this area. Some legal aspects will be covered but no prior knowledge is needed.

Limitations

The limitations of this thesis explicitly include the following:

- This thesis will not debate best coding practices for implementation of services.
- Some raw data in part of the analysis will be redacted or commented out to protect customer privacy and will not be available for third party review without written consent from the involved parties.
- This thesis will only briefly look at the different database systems that are available and their approach to load balancing.
- This thesis will focus on web applications, more specifically the 3-tier web server.

Acknowledgements

This would not have been possible without the help from my industrial supervisor Henrik Mörner as well as my examiner Professor Gerald Q. Maguire Jr. I would also like to thank all my fellow students whose help with reviewing this thesis has been highly appreciated.

Chapter 1 - Introduction

1.1. Introduction to Software as a Service (SaaS)

The past decade has been full of breakthroughs not only in technology (both software and hardware), but also in how these technologies are provided to users. The long accepted approach of buying one software license per computer and paying for upgrades has been challenged. The first, but not always recognized as SaaS providers, can be said to be the webmail providers that have served the market with mailing functionality for over a decade. The evolution of IT-infrastructure and the emergence of stable web browsers have enabled rich internet applications (RIA) (Mahemoff 2006). Along with this evolution there has been a change in provisioning and licensing, these changes have challenged the earlier price model by offering software as a service (SaaS) instead of a **per computer** based license for the software. Today many providers offer collaboration tools, project planning, customer relations management, scheduling, and many more services as on-line applications. (UCMS Group n.d.) (SalesForce.com n.d.)

1.2. SaaS advantages and problems

SaaS's strength comes from economies of scale as providers can consolidate the support, update, and server infrastructure for all the users of a specific service. However, along with the introduction of SaaS came the expressions: "Software on demand" and "Service on demand". These expressions **implied** that users subscribed for a service and expected to use this service the same day. This expectation allows rapid growth in demand and reduces administrative costs for the users, but can lead to a problem if the SaaS provider is not properly prepared to meet the growing demand. Inadequate server or support infrastructure by the provider can lead to a poor user experience. The poor user experience can be devastating for companies depending upon the service that they are expected to be provided. Poor management by either the provider or their customers can lead to organizational issues for both (here the customers are companies that have contracted for service with the SaaS provider).

1.3. SaaS and beyond - Cloud computing.

The trend that SaaS introduced was expanded upon by Amazon in 2007 when they launched their "Elastic Cloud Computing" service (here after referred to as Amazon EC). In this model customers can rent raw server capacity as "infrastructure as a service" (IaaS). Raw server capacity refers to the ability of the customer to specify the virtual machine images that are to be run when they are needed.

Amazon EC is based on virtualization and the ability to move virtual instances of a running VM to another host computer without interrupting (or only briefly interrupting) the operation of the VM. This facilitates consolidating hardware requirements for many different companies via resource pooling. This resource pooling provides increased resilience and greater efficiency, while lowering capital requirements. Today there are a large number of cloud providers that offer server capacity at a competitive price. Many SaaS providers see cloud computing as their solution to scalability, but there are many issues that SaaS providers have to consider before deciding if a cloud is the right platform for their infrastructure. Some aspects that must be considered are: reliability, lock in effects, legislative requirements, and data security.

1.4. About this thesis

This thesis will focus on SaaS and scalability issues, specifically how SaaS can be scaled. The basis for this thesis will be prior research and a case study made of Quinyx FlexForce AB.

The remainder of this thesis is organized as follows. Chapter 2 - will present the most common architecture for web applications and related queuing theory; Chapter 3 - will present currently available approaches to scaling web applications focusing on software design, hardware configuration and server configuration; Chapter 4 - will focus on financial and managerial aspects to consider; Chapter 5 - will use the conclusions from previous chapters to create a generic framework for scaling web applications; Chapter 6 - will use the framework to analyze and give recommendations for QFAB, chapter 7 will summarize the findings and suggest areas of future research.

Chapter 2 - Background

2.1. Previous work

There are much research within the areas of web applications and multi-tier systems. This chapter summarizes the key results and provides the basis for the framework for the infrastructure planning that will be used in Chapter 5 - of this thesis.

2.2. Web applications –multi tier architecture

In order for web applications to handle many concurrent requests, many companies apply a multi-tier architecture where each tier specializes in providing a certain functionality or service to the next tier. A common approach for a web based service is the three tier solution, see Figure 1. Each tier supplies the preceding tier with services and can in turn use services from the following tier. The first tier is usually the web server that manages the connection and session with each user as well as supporting the user interface (UI), thus making it possible to change, or have multiple UIs without affecting the business logic in the second tier. The second tier consists of the application server with its business logic and procedures to gather, process, store, and return data. Tier 3 is the database containing the raw data used by tier 2.

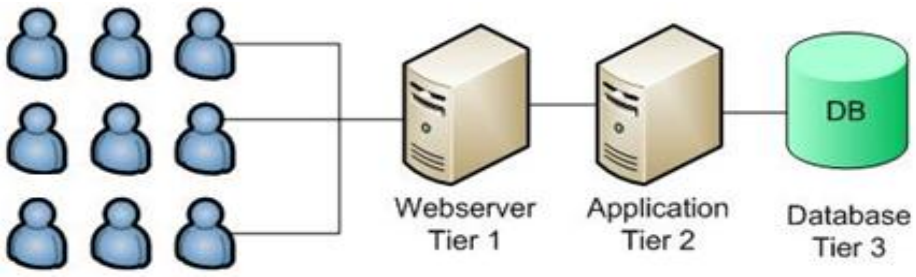


Figure 1 - Generic view of a three tier solution

In the simplest cases all three tiers are consolidated into the same physical server (this is a common practice for smaller webhosting services), but this is inherently not scalable and therefore outside the scope of this thesis. When working with larger systems each tier is implemented as several dedicated servers with load balancers to spread the load across these servers, rather than a single server as shown in Figure 1. Hence the servers in this figure can be regarded as logical servers, rather than physical servers. In even larger configurations different geographically located server parks are used and the initial load balancing is based on an approximate geographic location or network location of each client. This later solution will not be discussed as it involves considerable complexity.

Introducing three tiers offers a number of advantages that has made this a popular architecture, such as:

- The single point of access (the web server) makes integration easier and enhances security.
- The user interface can be updated without changing the business logic in the application server.
- Business logic and process definitions are kept separate from both the raw data and the user interface
- The data can be kept secure behind several layers of firewalls, while allowing easy raw data insertion and extraction by authorized personnel.
- Data backup only has to be done at tier 3*.

* Other servers could be used for file storage – in that case they need to be backed up as well.

2.3. Response time in a multi-tier system

No matter what software runs on each tier there will be many alternative ways to scale a single tier. A detailed description of scaling options and approaches will be presented in Chapter 3 - . For now we assume that each tier can be scaled to meet the needs of the preceding tier. Given the knowledge of how to scale a single tier, the task of scaling a multi-tier system may seem simple, but turns out to be very complex. As a result, the issues of scaling multi-tier systems have been neglected until recently. This thesis will present a method to analyze a multi-tier system developed by Bhuvan Urgaonkar and others (Urgaonkar, et al. 2005). The algorithm will calculate the average response time for a given number of concurrent user/sessions. This response time can be used to find the theoretical limit on the number of concurrent users that an infrastructure can handle within a given service level agreement (SLA). Since this theory is complex and its details are well out of the scope of this thesis only a short summary will be presented. The reader is referred to Bhuvan Urgaonkar's, et al.'s paper for further information.

The Mean Value Analysis Algorithm (MVA) needs the following input data[*]:

- Number of tiers in the web application (M).
- The maximum number of concurrent users / sessions (N).
- The user think time (Z), i.e., the average time it takes for a user to initiate a new request after finishing one.
- The average service time at $tier_m$ during light server load (S_m).
- The visit ratio at $tier_m$ during a time t (V_m). If more than one similar server is used at each tier, the visit ratio can be estimated to be divided by the total number of servers servicing each tier. Depending on the application running on the server, this could hold true for multiple CPUs in one server as well[†].

All of the data needed can be estimated from system or application logs. The algorithm can be extended to include resource limitations and concurrency limits[‡].

input : $N, \bar{S}_m, V_m, 1 \leq m \leq M; \bar{Z}$
output : $\bar{R}_m$ (avg. delay at $\mathcal{Q}_m$), $\bar{R}$ (avg. resp. time)

initialization:

$\bar{R}_0 = \bar{D}_0 = \bar{Z}; \bar{L}_0 = 0;$

for $m = 1$ ***to*** M **do**
  $\bar{L}_m = 0;$
  $\bar{D}_m = V_m \bar{S}_m$ /* service demand at each queue */;
end
/* introduce N customers, one by one */
for $n = 1$ ***to*** N **do**
  for $m = 1$ ***to*** M **do**
    $\bar{R}_m = \bar{D}_m(1 + \bar{L}_m)$ /* avg. delay at each que.*/;
  end
  $\tau = \left(\frac{n}{\bar{R}_0 + \sum_{m=1}^{M} \bar{R}_m} \right)$ /* throughput */;
  for $m = 1$ ***to*** M **do**
    $\bar{L}_m = \tau \cdot \bar{R}_m$ /* update queue lengths (little's law) */;
  end
  $\bar{L}_0 = \tau \cdot \bar{R}_0;$
end
$\bar{R} = \sum_{m=1}^{m=M} \bar{R}_m$ /* response time */;

Symbol	Meaning
M	Number of application tiers
N	Number of sessions
$\mathcal{Q}_m$	Queue representing tier T_m $(1 \leq m \leq M)$
$\mathcal{Q}_0$	Inf. server system to capture sessions
$\bar{Z}$	User think time
$\bar{S}_m$	Avg. per-request service time at $\mathcal{Q}_m$
$\bar{L}_m$	Avg. length of $\mathcal{Q}_m$
τ	Throughput
$\bar{R}_m$	Avg. per-request delay at $\mathcal{Q}_m$
$\bar{R}$	Avg. per-request response time
$\bar{D}_m$	Avg. per-request service demand at $\mathcal{Q}_m$
V_m	Visit ratio for $\mathcal{Q}_m$
$\bar{A}_m$	Avg. num. customers in $\mathcal{Q}_m$ seen by an arriving customer

Figure 2 - Mean Value Analysis Algorithm (Bhuvan Urgaonkar, 2005). Appears here with his permission.

[*] Since the algorithm uses Little's law, use only averages based on long term measurements.
[†] This assumes that the CPU is the limiting resource in servicing requests.
[‡] The reader is referred to (Urgaonkar, et al. 2005) for more information about optimizing the algorithm.

Chapter 3 - Approaches to scalability

This chapter examines three aspects of scaling: designing applications, designing a server architecture, and configuring server systems.

3.1. Application design approach

There are several approaches a developer or system architect can take to increase scalability and lessen the load on the server. The following subsections describe three of these approaches: doing more computing in the clients, using database abstraction, and shifting the time of usage to periods when there is usually low load.

3.1.1. Use client processing capabilities

Since 2005 Asynchronous JavaScript and XML (AJAX), has been used to create a large number of internet applications. Using AJAX web pages allow the user to navigate a webpage containing dynamic content **without reloading each page**, the information is simply fetched and updated using a background JavaScript. The idea of an internet application is simple; load a slightly bigger starting page and enable it as a small web browser based application that handles logic and redrawing of the page (Mahemoff 2006).

All major client side technologies (e.g., Adobe's Flash, Microsoft's Silverlight, and Oracle's Java) support communication with the web browser through JavaScript making it possible to communicate between applications developed with different technologies. These technologies can be used to perform performance demanding tasks on the client instead of the server. It is even possible for the client to perform SQL queries; for example, HyperSQL is a Java based SQL engine that can be run as a background applet and that provides a SQL functionality (HyperSQL 2009).

The process for client side offloading is:

1. Fetch raw data from the server
2. Insert this data into a local SQL database
3. Process data
4. Wait for a user commit command or automatically upload the result to the server
5. The server stores the processed data.

Having a local SQL engine is an extreme example of how processing, that what would have required server CPU cycles, can be moved to a client. However, this approach comes with a set of drawbacks, specifically: data integrity, raw data security, and the fact that the processing of data still requires a validation process at the server. However, this approach shows the possibility of using the client's computational power.

3.1.2. Use an database abstraction layer

In section 3.3.2 a major part of the scaling is achieved by scaling database access. Using an abstraction layer for all database queries facilitates the scaling of database access easier since new scaling principles only needs to be implemented in the abstraction layer. Different levels of abstraction can be applied and must be adapted to each situation. Many publically available abstraction layers focuses on giving developers a unified interface to the database to ease a potential transition to another database application instead of providing scalability features (MDB2 2009).

3.1.3. Caching

Almost every application can be configured or adapted to use caching. The principle of caching is that the probability of data being used increases when it just has been used, i.e., recently used data is the most likely data to be used in the near future. Storing data in a medium holding smaller amounts of data but which can deliver it faster than a secondary (complete) source has been a common practice for a long time. Some applications offer caching as a built in feature and in others caching can be enable by using 3rd party libraries. Memcached is a simple distributed server solution storing data in RAM for fast access and is used by many large service providers such as Twitter, YouTube, Wikipedia, Wordpress, and Digg (Dormando 2009).

3.1.4. Shift batch processing to off-peak hours.

For most services the problem is not cumulative computational power required, but rather the necessary peak computational power. Each service has to be able to handle the peak usage and this is usually what the infrastructure must be scaled to support, even if the peak usage period might only be a couple of hours in a 24 hour day or perhaps even only a few peak hours once a month (Armbrust, et al. 2010). One solution is to save computationally intensive operations into a batch processing queue that can be processed during non-peak hours. However, this approach is hard to apply when data computation is required for continued user interaction. The problem could be smaller in globally used applications as users may be in different time zones - hence spreading the load over the day and making the difference between peak and off-peak smaller.

3.2. Hardware approaches

3.2.1. Scaling out vs. scaling up

When scaling hardware there are two approaches to take: scaling up or scaling out (Kerwin 2003).

- Scaling up or vertical scaling is the process of upgrading the hardware within a single server, for example: upgrading to more or faster RAM memory, faster hard disks, and/or more or faster CPU cores. The main advantage with vertical scaling is, that it in the most cases this not require any software modifications, but it comes with the single point of failure problem. It should be noted that in some cases this scaling up may require a change in software license, as some vendors base the price for a license on the performance of the server that the software is run on.
- Scaling out or horizontal scaling is the process of adding more servers to increase the aggregate computational power. This approach is generally less expensive[*], but requires the software to be adapted for running on several servers in parallel and requires the use of load balancers to spread the load across these servers.

These two approaches have both advantages and disadvantages. Moreover, there are technical limitations of the underlying hardware that set an upper limit on how much you can scale up a single server. With increasing service demand there is frequently a time bound on the speed with which a service provider can scale out. In the next section we will look closer at when to scale and in Section 3.2.3 we look at a currently hot topic for scaling out: Cloud Computing (Google Trends 2010).

[*] In general, hardware gets cheaper with time but, high end hardware tends to be several times more expensive than lower end hardware. Depending on the growth rate in demand several small servers could prove cheaper than one big or vice versa.

3.2.2. When to scale

It is desirable to scale up or out **before** a server reaches the so called "wall". This performance wall exists because of how queues behave and how computers access their resources. It is very hard to analytically analyze resource usage, but queuing theory can be used to make estimates* and aid in infrastructure planning. However, actual tests are always recommended to verify estimates. Most applications and servers usually start off scaling well, often in an almost linear fashion. However, there is a point where adding additional concurrent requests increases the response time **exponentially**. It is important to understand when this exponential scaling occurs, because at some point the additional load needs to be avoided to prevent the user's perceived performance from falling below the performance which is considered acceptable (this might be governed by a SLA). Before the load has reached this level, any additional load must be assigned to other servers. Figure 3 shows a typical wall for different server applications.

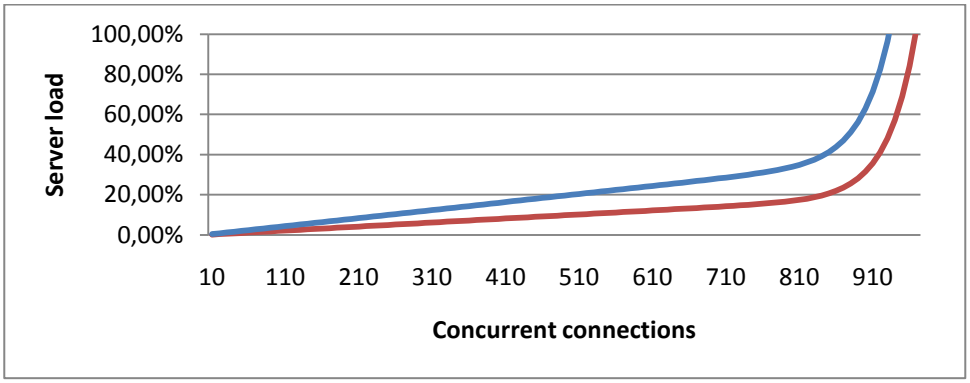


Figure 3 - Example of typical load behavior. The two lines represent load behavior of different server configurations where server load on the y-axis is an abstraction of total utilization. Every system behaves differently but will reach a "wall" with increased load.

3.2.3. Cloud computing

One way to easily scale out is to use a technique called cloud computing. Cloud computing can be described as computing as a utility. In 2004 Reuven Cohen thought of a concept he called Elastic Computing (EC). His idea was simple but genius; use virtualization technology for dynamic large scale deployment of servers. He was far from the first to think about computing as a utility, much in the same way we think about electricity†. He has however become known as the man who together with Amazon was the first to successfully capitalize on the concept. "Amazon web services"-service (AWS) was in 2005 announced and roughly one week later Google Inc. announced their version of elastic computing called cloud computing. (Riboe 17th of Feb 2010).

Cloud computing has in the past three years become a hot topic that has been widely debated. Some see it as the salvation from expensive server halls, while some fear that we have not yet seen and appreciated its disadvantages (Hellström 2010) (Djurberg 2010). However, there can be no questions that there are many who have or will adopt this approach to scaling applications.

* Accurate estimates can usually always be found, **if** there is enough information about the users and the system

† The idea of time sharing of computational power was first invented with grid computing over 2 decades ago. Since then the idea of provisioning the power of the grid as a commodity has been widely debated. (Armbrust, et al. 2010)

3.2.4. Three types of clouds

There are currently three types of clouds that need to be distinguished. These three types of clouds are: Infrastructure, Platform, and Application clouds (Riboe 17th of Feb 2010). Figure 4 shows these three forms and some of the players that are active in these three areas.

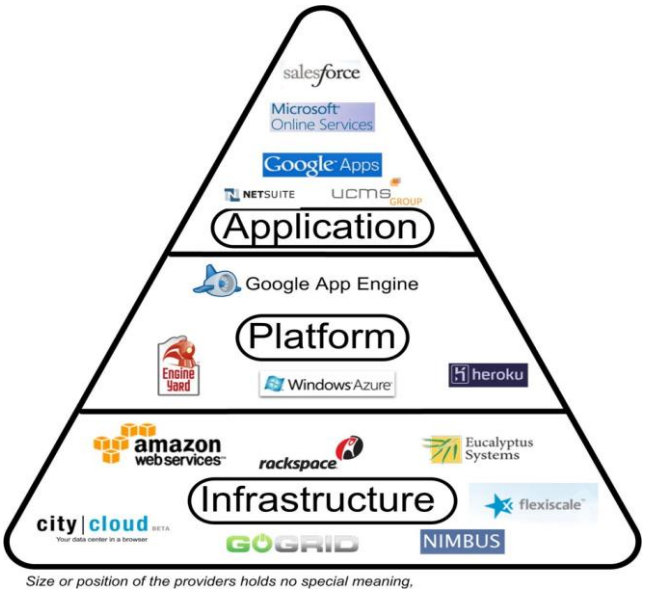


Figure 4 - Some of the market players and their roles, logos are traded marked and belongs to the respective companies.

3.2.4.1. Infrastructure cloud

An infrastructure cloud refers to a service that allows you to run a virtual image of a computer in the provider's server farm as described in section 1.3. This form of cloud provides the most freedom to execute arbitrary code, but requires more administration and tailoring of a service for the cloud. This form of service is also referred to as "Infrastructure as a service" or "Hardware as a service" (Mell and Grance 2009).

3.2.4.2. Platform cloud

A platform cloud offers a hosting environment for code written in a certain language. The hosting environment takes care of the scalability automatically and usually requires less administration once the code has been written. In most cases the code must be written specific for the hosting environment, hence leading to a lock-in effect. This form of cloud is also referred to as "Platform as a service" (Mell and Grance 2009).

3.2.4.3. Application cloud

The application cloud is closely related to SaaS in that sense that **the SaaS provider** develops and hosts an application for the user. To the end consumer the difference is insignificant, but to the provider the difference is apparent in **how the service is hosted** and **how well it scales** with increased load. If the user(s) **experiences near infinite capacity** the SaaS service can be said to belong to a application cloud. Note that in this case the SaaS provider/application cloud provider is responsible for all of the scaling necessary to meet the customer demands. This definition is highly debated and many argue that the correct term is SaaS (Mell and Grance 2009).

3.2.5. Concerns with clouds

As with any new technology there are upsides as well as drawbacks. This section summarizes the most important and the most common issues with clouds.

3.2.5.1. Data security

Keeping data safe has always been a key issue for service providers. Privacy both for individual as well as for corporate data is essential. Today it is unknown how secure a virtual container or virtual network actually is.

Earlier the firewall was considered the first line of defense for keeping networks secure. While some cloud providers claim to offer isolated and secure environments, there is still a great deal of uncertainty concerning data traffic routing and fault isolation when the infrastructure is not physically isolated.

Another issue concerning data security is secure data deletion. When a file is deleted does the provider actually delete the data or has the system just deleted the file entry in the directory? Unless the disk blocks of the file are actually overwritten multiple times with random bits it may be possible for a subsequent process to recover the data that had been written to these disk blocks (Gutmann 2003). For some kinds of applications there are even legal requirements about secure data deletion (see for example the U.S. HIPPA requirements) (Scholl, et al. 2006). Legal Scholar (jurist) Jane Lagerqvist at Datainspektionen (personal communication, March 3rd, 2010) claimed that to the best of her knowledge, in Swedish law, there are no specifications for destruction methods of sensitive data. It is the keeper of personal information's responsibility to ensure a secure deletion.

3.2.5.2. Lock-in effects

Each infrastructure cloud provider has its own API for dynamic deployment of new images. Designing an application for a specific cloud causes a degree of lock-in that most chief technology officers would like to avoid. These lock-in effects will most likely be smaller in the future, since more and more providers are offering an API compatible with Amazon's Web Services (AWS). An AWS compatible open source alternative is available (Eucalyptus Systems 2009) which will further lessen this effect.

3.2.5.3. Terrorist threats

Cloud services consolidate computational power. When a server hall is hosting applications for thousands of businesses, it is likely that such a large server hall will be a target for terrorist activities. Not surprisingly, today server halls are just as an important part of a nation's critical infrastructure as water and power utilities are. With large providers like Amazon this threat can be handled through the use of geographical failover and backup schemas. However smaller providers will probably have less capability to offer the same redundancy which transfers redundancy implementations to the customer.

3.2.5.4. Third party trust

Sensitive data will always be available for the administrators of a network or a server system[*]. Handing over control of data to a third party raises both legal and contractual issues. The European Union has issued a data protection directive that states that data containing personal information must be kept secure and may not be transferred to a country outside the EU unless that country has an adequate "level of data protection". What an adequate level of security is has been widely debated especially since most courts can issue a court order to enable the police to seize any data in order to uphold local laws or to aid in investigations (The European Parliament And The Council, Of The European Union 1995). A rule of thumb for European businesses is to keep data within the EU borders or consult a law firm.

3.2.5.5. Service level agreements are inadequate

Most providers of cloud services state that their services are in a beta stage and that they will take no responsibility in the event of failure. See section 3.2.9.5 for a comparison of service level agreement (SLA) levels for different providers[†].

3.2.5.6. Licenses and licensing costs

Since Cloud computing is a relatively new phenomenon where virtual images can be turned on and off depending on demand, there is a great deal of uncertainty of how to interpret current software licenses. There are even questions whether or not the licenses are compatible with a profitable cloud solution. Some providers have solved this issue by managing licensing for the customers, but only with preselected software.

3.2.5.7. Unpredictable costs

With a self-hosted server environment it is simple to calculate the monthly cost, but for a cloud environment with dynamic loading and termination of servers there is no upper limit on how much the environment can cost nor is there an easy way to calculate future costs in a volatile business[‡]. However, for most services it **may be** reasonable to assume there is a correlation between CPU time and income, hence the more CPU time you need the more likely that your income will be higher than the cost. This can still lead to a cash flow problem – depending upon the time between realizing the income and when you needing to pay the cloud provider. It should also be noted that some cloud providers allow their customers to set an upper limit on costs – thus bounding the customer's payments; however, this may lead to a denial of service of legitimate users when this limit has been reached.

---

[*] This has been true for some time, studies at Princeton University have shown how encrypted hard drives can be decrypted if an attacker gains access to a turned on computer by accessing and dumping the RAM (Halderman, et al. 2008). When working with virtualization technology a user with access to the hypervisor can easily dump the whole RAM of an virtual image while it is still running (this is in fact the basis of a technique used for live migration of virtual images from one host to another and when creating snapshots of a virtual state in the machine). Even with encrypted hard drives physical access or in the case of virtualization, access to the hypervisor, give complete access to data contained in an active machine.

† According to (Gustavsson and Agholme 17th of Feb. 2010) there is an ongoing evolution towards better SLAs. Usually SLAs are negotiable and should not be blindly accepted without legal and technical review.

‡ Human group behavior tends to be predictable and with sufficient statistical data there is usually a good estimate to be found.

3.2.5.8. Network sniffing

Within a cloud the infrastructure is shared, thus there is a risk due to network sniffing, ARP cache poisoning, or a man in the middle attack on cloud servers. See Appendix A for further details on how this easily can be done in the "City cloud" service. Solutions to this problem are described in section 3.2.8

3.2.5.9. Virtualization overhead costs

A cloud solution is by its' nature a virtualized environment and virtualization comes with lower performance. The actual overhead costs for virtualization depends on factors such as technology used, hypervisor used, host machine, parallel guests, configuration and guest application(s) (Tickoo, et al. 2010). Today the overhead has been minimized through the help of MMU virtualization (Adams and Agesen 2006). The largest overhead is found in I/O operations, hence adding delay and limiting throughput (Dong, et al. 2009) sometimes leading to a high variance in performance (Armbrust, et al. 2010).

3.2.5.10. The overall problem

Dave Durkee claims that many of the problems with cloud computing come from the view of computing as a utility where the competition is based on price. Price competition causes providers to race to the bottom by cutting corners on performance to lower costs. Durkee suggests that the solution to these problems is quantitative transparency of the cloud infrastructure. Transparency that will be required from enterprise grade customers with a high demand for reliability and trust (Durkee 2010). Even though it is unlikely that all providers will join his predictions in a race to the bottom it is plausible that many enterprise applications will wait until a more transparent system is presented.

3.2.6. A private cloud

There are concerns that need to be considered when outsourcing infrastructure to a cloud. Many hesitate to use cloud computing due to the concerns discussed in previous section, but these businesses still want to harness the power and flexibility of a cloud infrastructure (Amazon Web Services 2009). A solution to this is to implement a private cloud.

Software, such as Eucalyptus and Enomaly offers a business the ability to run a cloud solution in their own infrastructure (server halls) and, in the case of Eucalyptus, with the same API as AWS. Running a private cloud offers much of the flexibility of a public cloud, but with the enhanced security of controlling physical access to the network, computers, and storage media. Eucalyptus is an open source solution to cloud computing (Eucalyptus Systems 2009) and can, at the time of writing, be obtained as a standard package in the Ubuntu-server distribution. Enomaly is a web based virtualization management platform compatible with most existing virtualization technology that can be used to create a cloud environment. A limited open source version is available for free (Enomaly.com 2010).

Another reason for using a private cloud is to gain experience in how to fully use the dynamic scaling out capabilities that can be achieved with the cloud.

3.2.7. The hybrid cloud

Using a private cloud has two major drawbacks: the cost is fixed no matter what the usage and scaling has to accommodate peak usage. A hybrid cloud solves the later problem by having a public cloud assist the private cloud during peaks. This solution places all of the database servers **within the private cloud**, while optionally placing web servers and application servers in the public cloud during peak hours. The reader should note that transferring data outside EU for storing **or** processing of personal information is still subject to the European Commission Data Protection Directive (The European Parliament And The Council, Of The European Union 1995). As a result it may be necessary to do some of the processing for both the web and application services in the private cloud.

3.2.8. Solutions to Network sniffing

In the case of co-tenants sniffing network traffic there are two solutions where they target different attack techniques.

The second solution manages the so called ARP cache poisoning problem. Using statically configured ARP tables instead of dynamically configured tables prevents an attacker from poisoning the ARP cache of your machine[*]. Although this can be done on all major operating systems, older versions of Microsoft's windows have been reported to ignore the static flags[†]. While static ARP tables could protect a host's outgoing data, it does not protect incoming data unless the same type of static configuration is done at the routers (Goyal, et al. 2005). ARP cache poisoning and other ARP related attack, such as ARP spoofing, is an infrastructural problem that has to be handled by the provider. There are programs for detection and warning of potential attacks, but detection is not prevention.

The third solution and the only solution that a tenant in a infrastructure cloud can use to safeguard against network sniffing is to use **data encryption** of all traffic. Most server applications support encryption of data traffic and they are usually well tested. Though encryption of data traffic is always recommended when transferring data outside a physically controlled environment it comes at a price of performance. Encryption methods such as SSL and TLS have two stages that influence performance. The first stage is the initiation where encryption keys are shared using asymmetric encryption. The second stage is the actual encryption of data using symmetric keys and has an overhead linearly proportional to the data stream size. Studies have shown that depending on configuration, data size, and encryption techniques used, the cost of sending encrypted data compared to unencrypted can reach as high as 9 times (Nachiketh R. Potlapally 2003). The biggest difference is seen in small data streams where symmetric keys are not reused between sessions. (Coarfa, Drusche and Wallach 2002). Worth noting is that when traffic within a cloud infrastructure is insecure the option to use load balancers to offload SSL overheads from the servers is removed since internal encryption is still needed.

3.2.9. Cloud providers as of 2010

This section presents some of the major cloud service providers at the time of writing. Information about services has been gathered from official websites and communication with support personnel (May 2010). However, this information is **subject to rapid change** and should be **revalidated by the reader**.

[*] Microsoft's Windows environments have been reported to ignore the static flag in an ARP table thus, making static tables an ineffective solution to the problem. (Goyal, et al. 2005)

[†] The author could not verify this problem on Windows7. Other versions are untested but Vipul Goyal and Rohit Tripathy claims early versions are to be affected (Goyal, et al. 2005)

3.2.9.1. Amazon

Amazon currently operates four server parks, by Amazon called regions. One of these regions is physically located in Ireland, one in the Northern Virginia, one in Northern California, and the most recent addition is located in Singapore. Amazon offers many services beyond raw computational power such as: **Cloud Front**, a web service used to distribute files; **Virtual private cloud**, a VPN tunnel to the cloud servers so that they seamlessly can be integrated into a customer's existing server park; and **Auto scaling** and **auto load balancing** among other services. Amazon is the provider offering the most competent and most powerful hardware configurations (called "instance type") to run a virtual image in. These instances offer up to 64GB of RAM and 26 cores. Of all of the providers, Amazon offers the most information about its security infrastructure and claims* to be safe from several attack methods, such as promiscuous sniffing, ARP cache poisoning, and IP spoofing. (Amazon Web Services 2009). Among the drawbacks are overall performance and complicated cost schemes.

3.2.9.2. Gogrid

Gogrid focuses its efforts on web applications and offers a hybrid environment through dedicated servers together with a dynamic cloud solution that can be used to handle usage spikes. Load balancers, VPNs, and firewalls are all included in their architecture. GoGrid is also unique in that they are the only provider that provides a **role based access control list** to delegate responsibilities to sub administrators†.

3.2.9.3. Rackspace

Rackspace offers dedicated servers, a cloud infrastructure, an application cloud for web applications, a cloud front like file sharing service, and local RAID 10 hard drives. This makes Rackspace an interesting competitor in the market. While GoGrid offers to make a reasonable effort to restore data in case on an emergency, Rackspace offers a bootable rescue mode with file system access to repair troublesome machines. Rackspace's biggest drawback is that only Linux operating systems are supported‡.

3.2.9.4. Flexiscale

Flexiscale takes a different approach to both network security and payment than most providers. Flexiscale requires you to pay before you use, while the rest of the service providers offer service on a "pay as you go" basis. Flexiscale's approach to securing the network is to use VLANs and packet tagging. However, there are some questions of just how much security this provides since they do not have dedicated hosts for each VM. Thus if a packet separation to the different virtual machines is made in the switch rather than in the hypervisor there is still a risk that a co-tenant (i.e., another customer's VM is running on the same instance as your VM) could eavesdrop on your network traffic. The author has not confirmed if this is a real threat or not. A solution would be to require that only one VM is run on a given physical machine at any given time.

* The author has not been able to disprove their claims.

† Available roles as of the 1st of May 2010 are: Read only, System users, Billing user, and super user.

‡ Virtual Windows machines are provided as a beta service.

3.2.9.5. Comparison of a number of cloud provider's offerings

A comparison matrix of the providers discussed in the previous section is shown in Table 1

Table 1 - Comparison matrix of selected cloud providers

Provider	Standard SLA level[*]	Traffic cost (per GB)	Location	Public IPV4 addresses	Minimum Capacity	Maximum Capacity	Dedicated kernel[†]	Network Sniffing[‡]
Amazon	Level 2	$0.08-$0.15 (Volume discount)	USA, Ireland and Singapore.	5 public	1.7 GB RAM 1x1.1 GHz 64 bit $0.085/h	68.4 GB RAM 26x 1.1 GHz 64bit $2.4/h	N/A	No
Rackspace	Level 2, Credits will max be 100% of a paid fee	$0.22 in $0.08 out	USA	1 per machine	256MB RAM, 4 x 2.4 $0.015/h	15872MB RAM 4x 2.4 GHz $0.96/h	yes	No
CityCloud	Level 2	0.5 SEK	Sweden	1 persistent per VI	0.5GB RAM 1x 2.26 GHz 32 bit 0.185 SEK/h	16GB RAM 8x 2.26 GHz 32 bit 3 267 SEK/h	yes	Yes, Passive (April 2010) [§]
GoGrid	Level 2, credits equal to 100 times downtime	$0.29	San Francisco, USA	16 public	0.5GB RAM 1x 2.26 GHz 64 bit $0.1/h	8GB RAM 1x 2.26 GHz 64 bit $1.52/h	yes	No
Flexiscale	Level 2, Credit will be a maximum of 100% of the fee for the last 30 days	$0.0878	United Kingdom	5 public	0.5GB 1x 2GHz $0.035/h	8GB 4x2GHz $0.35/h	no	[**]

[*] SLA levels are defined as.

Level 1. The provider takes no responsibility for the service
Level 2. The provider offers reimbursement for the paid service, but only in the form of credit for future use and does not include payment for indirect damage
Level 3. The provider takes considerable responsibility for their service and insures against limited indirect damage caused by potential failure

[†] The virtual machines' security and stability could be affected if they share a kernel and are not properly isolated.
[‡] As described in Appendix A
[§] City Cloud has commented (personal communication, May, 2010) that this issue was due to a bug that has been resolved with a system wide upgrade in May 2010. The author has not confirmed their claims.
[**] Have not been conducted due to the risk of breaching their "Acceptable use policy", Section 3 (May 2010).

3.3. Software approach

3.3.1. Web server / application server

There are several choices of software when setting up a web server and most share the same principle for scaling. The most common programs in the market are Apache, Microsoft's IIS, nginx, and lighttpd (Netcraft.com 2010). Apache and ISS are by far the most commonly used. Web servers tend to scale out very easily even though problems can occur with server side session variables (used to store dynamic content between user requests). Load balancers tend to offer a persistent or sticky flag that can be set to always forward packets from a host to a specific server. In the case of HTTPS there is a problem with identifying the current server that a load balancer should forward packets to. Since SSL3.0, this issue has been solved by not encrypting the session key in the HTTP header which allows for a load balancer to keep track of session keys and forward each session associated request to the designated server. This approach has the drawback of only doing load balancing at the initial user request of a webpage. Today many hardware based load balancers uses hardware to perform all SSL encryption on behalf of the other servers (jetNEXUS 2010).

3.3.1.1. Apache

Apache is an open source solution that offers a wide variety of functionality and due to its loadable module system it can be adapted to suit very specific tasks. Apache has support for PHP, Ruby, ASP (not an official Microsoft module) (Chamas Enterprises Inc. n.d.), and CGI-enabled languages. Modules are available for load balancing which makes it easy to set up a cluster of web servers using Apache. Apache comes without any warranty or support from the Apache Foundation (Apache Software Foundation 2009). However, support can be obtained from third party consultants.

3.3.1.2. Microsoft's Internet Information Services (IIS)

IIS was developed and is maintained by Microsoft and natively runs ASP (aspx/dotnet). Additionally, other languages such as PHP are supported through CGI or third party modules. Professional support can be purchased from Microsoft. IIS runs on Microsoft's Windows 2003 server or later and it comes with load balancing capabilities (Microsoft Corperation 2010).

3.3.1.3. Nginx and lighttpd

Nginx, and lighttpd are two of the more successful lightweight web servers that aim to be small, fast, secure, and easily scalable. Lighttpd has been shown to serve static content faster than Apache or with lower CPU usage. Using a lightweight web server to ease the load has been used by large providers such as YouTube and Wikipedia (LigHTTPD 2007).

3.3.2. Database systems

As the last component in the 3-tier web server architecture the database (DB) is responsible for storing, indexing, fetching, updating, and deleting data for the application server. There are many database systems available to suit different kinds of needs. The structured query language (SQL) is by far the best known and most used interface method. DBs tend to be the most complex component of the 3 tier architecture to scale. However, due to their commercial importance database performance and performance tuning methods have been extensively explored both theoretically and practically.

3.3.2.1. Terminology

Before dealing with DBs in a scale out approach we first introduce some key terminology. Three of the most important terms are:

Relation(s)	Used when data in one table refers to data in another table. Most SQL engines provide support for automatic checking of so called "Foreign keys" to make sure relations are maintained.
Transactions	In SQL a transaction is a set of queries that executes or fail together. The most common example of transaction safety is that of the bank: you do not want to credit one account without debiting another at the same time; these operations are performed by two different queries but are part of one transaction.
Data consistency	For data to be consistent it should be the same for all requests at a certain time. During an update read access must be denied since the data is in state of flux.

3.3.2.2. Scale out –Replication

To scale out a SQL server[*], one of the easier solutions is to add a slave server to aid in responding to read only queries. To correctly implement replication requires all add/update/delete queries to be sent to the master server while reads can be executed on any server. Transaction replication is a common replication method for SQL servers. The servers are kept synchronized by a simple mechanism:

1. Create a copy of the master database on each of the slaves; then
2. Forward all update/add/delete queries to the slaves so that they can update their copy of the database.

This solution is simple in that sense that it does not require much adaptation of the application. The drawback is that each update/add/delete query has to be executed in each slave. If a master spends 30% of its time updating the database, then the slaves will spend just as much time updating their copy leaving only 70% extra capacity per server for other operations[†]. This scenario is illustrated in Figure 5. This scenario can be even worse for a system which spends 90% of its time updating the database. As a result in a system that has frequent updates, scaling out using replication is unfavorable (Zaitsev 2006).

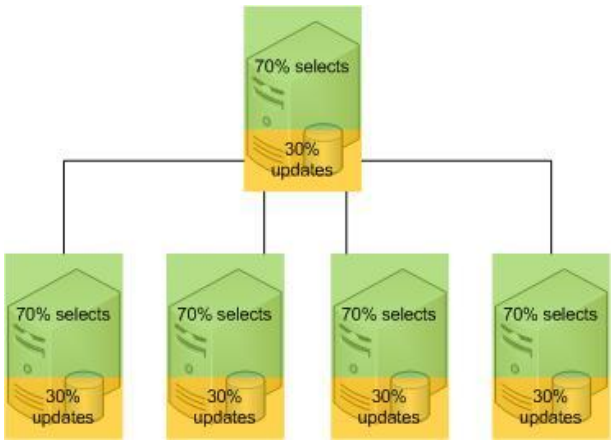


Figure 5 - Replicating a server that spends 30% on updating the database will only add 70% of the potential capacity in every new server.

[*] Servers considered are MySQL and MsSQL

[†] This assumes that the slaves and master server have the same hardware configuration.

3.3.2.3. Scale out – Splitting databases, vertical data partitioning

As the workload grows, transaction based replication is not a viable approach. The alternative is to design the application to direct different queries to different database servers. An application could divide the user interaction data (messages, user names, contact information, departments, etc.) to a server cluster while placing marketing data on another server cluster (Shabat 2009). Splitting databases requires a deeper modification of the application, but can usually easy be achieved by using a database abstraction layer. Both MySql and MsSQL support accessing tables on remote servers as if the data was stored locally[*] which allows for maintaining relations even though tables may be located on a remote machine[†].

3.3.2.4. Scale out – Sharding, Horizontal data partitioning

The most powerful and potentially the most complex approach is for the application to implement sharding. Sharding takes place in the application, but requires a high degree of database planning. To use sharding there must be a logical way of separating correlating data chunks. In a system hosting multiple customers in the same database each customer and all their correlating (relations in a relational database) data could be moved to another server assuming there is no sharing of data between customers. (Shabat 2009) .

3.3.2.5. Clustering

A cluster setup refers to a system where scaling out and splitting data between servers as well as offering data redundancy is functionality provided by the cluster application. The developer or end user of the cluster system will have the scaling abstracted away. In the case of the MySQL cluster, it holds limitations compared to the regular databases. Features like foreign key checks have been omitted in the cluster version (MySQL 2010).

[*] MySQL uses a federated storage engine. MsSQL uses a technique called “Linked servers”.

[†] Automatic foreign keys checks are not supported in MySQL, but this has been discussed as a new feature for version 6.1, (Osipov and Gulutzan 2009)

Chapter 4 -
Economical, legal, and business aspects of scaling

Selling software as a service offers great potential for rapid growth. This chapter examines some managerial and financial aspects that should be considered before and during rapid growth. It may even be favorable to slow down growth in order to handle it properly. (Avila, Mass and Turchan 1995).

4.1. Managerial aspects

In almost every business there are resources that have to scale with the growth of the customer or sales base. In a production based business the resources are assembly machines, personnel, and raw components. If sales increase then more material is likely to be used. The same is true for SaaS providers, but resources translates into server infrastructure (not discussed in this chapter), support personnel, and R&D personnel.

Most requirements are evident but, can easily be forgotten or underestimated if when management focuses on business expansion. Brooks law (Fred P. Brooks August 1995 (first released 1975)). states "Adding manpower to a late software project makes it later" This observation is based on the fact that each new developer has a warm up phase or training phase where s/he is more likely to make mistakes rather than being productive and during this phase requires training from otherwise productive developers, thus reducing overall productivity. This is why it is important to have a plan for recruitment before the need occurs. The same, though to a lesser extent, is true for support personnel.

The development of organizations has been studied by many and can simply be described as an evolution from entrepreneurial cooperation to standardization to bureaucracy (Olsson and Skärvad 2007). Throughout this process, no matter how quick or slow it is, there is a need to maintain active communication between departments so that decisions do not affect customers without notifying them. One solution to ensure that relevant information reach the correct recipients is to use a Responsible, Accountable, Consulted, Informed (RACI) chart (also known as responsibility assignment matrix, RAM). The RACI chart holds information of who should be informed or consulted and who is accountable or responsible for a process. The RACI chart shown in Figure 6 displays an example of processes and information requirements for different departments. It is easily maintained and maintains a clear communication policy and is a common practice in project management (The Project Manangement Hut 2010).

	CEO	CTO	Account manager(s)	Support	R&D	Other departments
New feature requests		A		I	R	
3rd party integrations	C	A		I	R	
Deviations from company standard SLA	I	R	C	C	A	
Fixing bugs		A	I	I	R	
Live launch of updates	I	I	I	I	R	I

Figure 6 - Example of RACI (Responsible, Accountable, Consulted, Informed) chart.

4.2. Financial aspects of Cloud Computing

Michael Armbrust et al. have analyzed the financial implications of cloud computing compared to traditional server hosting (Armbrust, et al. 2010).Their key points are:

- Cloud computing transfers the risk of over and under provisioning of resources to the cloud provider.
- Cloud computing migrates server expenses from being a capital expense to an operational expense by using a "Pay as you go" scheme.
- Most traditional servers run an average 5-20% of maximun capacity to facilitate peak requirements that only lasts for a limited time.
- Scaling up or down can, in a cloud, be done in minutes instead of days or weeks with physical servers in a self-hosted environment.
- Self hosted environments are unlikely to utilize more than 80-90% of the total capacity (See section 3.2.2 "When to scale").
- The cloud offers a way of handling usage surges (from example peaks might be caused by media coverage).

These authors suggested an analytical model for evaluating the economical impact of the cloud compared to the self-hosted alternative. Their model is based on the assumption of a correlation between CPU cycles and income. A more generic model based on their finding will be presented in the next section.

4.3. Economical impacts of the cloud

The model suggested[*] in this section is designed compare the cost of a self-hosted environment to a cloud solution. The approach used is to minimize the total cost of ownership (TCO) and explicitly include risks as a cost using the expected monetary value (EMV) approach.

R_s Risk of surges in demand / under provisioning causing downtime (Could be very hard to estimate, but in an expanding company this factor should be not less than 1%[†].

R_D Risk of downtime in current server setup. The likelihood or maximum expected downtime in a self-hosted environment. Use historical data.

R_{SLA} Risk of unavailability/downtime in the cloud service (see 3.2.5.5-Service level agreements are inadequate). The maximum downtime the provider guarantees. If the compensation for breaching the SLA is less than the estimated cost for a breach: add 0.9% downtime to this risk factor[‡].

C_H Cost of self-hosted server (cost / server / hour).

C_C Cost of equal cloud hosted server (cost / server / hour). These costs should include an average of all costs associated with each alternative. Include the sum of rent, hardware cost(s), server maintenance, electricity, cooling, traffic transfer, storage etc.

N Number of self-hosted servers required for peak usage.

U Average utilization of self-hosted servers.

I Financial impacts of one hour downtime.

[*] The reason for the use of the adjective "suggested" is to point out the untested nature of this model.

[†] Given without any scientific background and must carefully selected on a case to case basis. A starting point for this factor is to set it the same as R_{SLA} or higher. This factor is only of interest if the application has implemented automatic scaling routines in the cloud; if not the risk of usage surges would not be eliminated.

[‡] This factor of 0.9% is taken from the author worst known, downtime of a cloud provider (Amazon, S3 service, down for 8 hours) (Armbrust, et al. 2010)

Equation 1 – Cost factor

$$CostFactor = \frac{N * C_H + (R_S + R_D) * I}{80\%^* * N * U * C_C + R_{SLA} * I}$$

Note: **The denominator should not be lower than the cost of the minimally required number of servers + the EMV or the risk. For example in a 3 tier system the minimum number of servers is 3 and in that case, a cost lower than the smallest instance multiplied by 3 would be impossible to achieve.**

A cost factor equal to 1 would indicate that the two options are equally expensive while a cost factor less than 1 indicates that the self-hosted environment is cheaper and the opposite for a cost factor greater than 1.

4.4. Analyzing scalability

In many cases it can be useful to assure customers that a SaaS solutions scales to the customer's needs or analyzing if a SaaS provider can scale as claimed. The easiest way to prove scalability is the use of case studies of previous successful scaling, in other words based upon the actual experience of another customer or a stress test. When such cases are not available or they are deemed insufficient, the author suggests there are a few aspects that a SaaS provider can use to prove that s/he has a plan for and thus a plan for scaling. These points can be divided into two categories: technological and managerial. To receive a good evaluation, a SaaS provider needs the ability to do well (i.e., be evaluated well) in both.

4.4.1. Technological aspect

To prove a technical scaling capability the provider must, in some way, show[†]:

- proof of the ability to split each tier over several servers
  - either by a DB abstraction layer with a database with logically decoupled data and/or
  - a solution for session handling when many servers are involved
- the ability to use an arbitrary number of load balancers or an load balancing algorithm that can be scaled to the desired level, and
- **that adding more servers will increase capacity**

If these points can be proven to be true, then there is a good foundation for scaling. Unless all managerial aspects also fall in place, the SaaS provider may not be able to scale in practice

4.4.2. Managerial aspect

To prove that scaling is possible from a managerial point of view, the provider should show or ensure that it has:

- Sufficient finances to invest in infrastructure and the necessary personnel
- Sourcing agreements to establish the required infrastructure in time. The timeframe from order to delivery and implementation could become a significant issue if not properly managed.
- Processes for training and recruitment of R&D and support personnel.

[*] Even in the cloud it is unreasonable to assume use of 100% of the resources available.
[†] Deducted from pervious chapters

Feindt et al (Feindt, Jeffcoate and Chappell 2002) cites, in their work, a report (from London Business school) where one of the success factors for a rapid growing small and medium sized enterprise is “Close contact with customers and a commitment to quality of product and/or service” and later cites another work (case study from European Innovation Monitoring Systems) that stats that one of three success factors is “Mobilising resources: securing necessary financial, human and technological resources to enable growth”. Though the aspects above are not literally mentioned as a key success factor in their work, they can easily be deducted from other statements, thus supporting the arguments of the aspects.

Chapter 5 - Suggested framework for scaling

This chapter summarizes the previous chapters and suggests a model to assist in deciding optimal ways to scale a SaaS solution. There is no single solution to scaling, but certain aspects about the service to be scaled can be used to eliminate options. In the end there will always be a need to examine the advantages and disadvantages of different alternatives.

5.1. Theoretical scaling

To achieve or predict future scaling and performance bottlenecks it is possible to use queuing theory. If the service is already deployed it is possible to use logs to get input variables to the MVA algorithm. Adjusting these input parameters in a systematic fashion will show how small changes will affect the overall system performance. This information can be used to target actions to where they are most needed.

5.2. Scaling web servers, tier 1 and 2

Scaling tier 1 and 2 servers is most frequently a matter of finding a load balancer that meets the needs of the current business. Including caching and offloading of heavy file downloads to lightweight web servers are supplemental actions that can be taken. Identifying commonly used files or database queries can be done through a log analysis and targeted actions can be developed to implement caching or offloading if these have not been a part of the original development process of the application. Both actions increase the speed of the servers, but the problems of caching such as the use of stale data must be kept in mind.

5.3. Scaling databases

When scaling databases there is one golden question to ask before all others: "Do I expect near infinite scaling capabilities of my solution". If the answer to the questions is yes, there are only two solutions to choose from.

- Use an out of the box cluster solution where scaling is managed by an already implemented system. The advantage of using a cluster solution is that it usually comes with a built in degree of fault tolerance, but may lack functionality or performance[*] compared to a standalone server.
- Use sharding to divide data logically across servers and query only the server with the specific data. Consider the cost of backing up data on several servers and the cost of implementing the lookup algorithm to find the correct server.

If a scaling solution does not have to be unlimited, then replication can be used on systems that have limited database updates. Replication can be used in conjunction with sharding but, requires more adaptation of the application.

5.4. Scaling hardware

When scaling hardware there are today four main alternatives:

1. Running dedicated servers
2. Running virtual servers (For example, running a private cloud)
3. Running in a public cloud.
4. A combination of any of the above.

The first option is mostly viewed as a solution with limited future, except in very special circumstances. Virtualization offers unprecedented flexibility to maximize the utilization of resources. Today there are very few reasons not to run applications in a virtual environment.

[*] The performance could just as well be better in a cluster.

The decision to use a private cloud or not depends on the size of the current server park or if the private cloud is to be used to ease or enable a (partial) transition to a public cloud. With a small number of servers, the usage of the private cloud solution is limited unless the providers want to gain experience in working with a cloud.

5.5. Schematic chart for infrastructure scaling

This section suggests a simplified framework to aid in the decision if a cloud solution is suitable for a business application. Figure 7 displays this framework in the form of a flowchart.

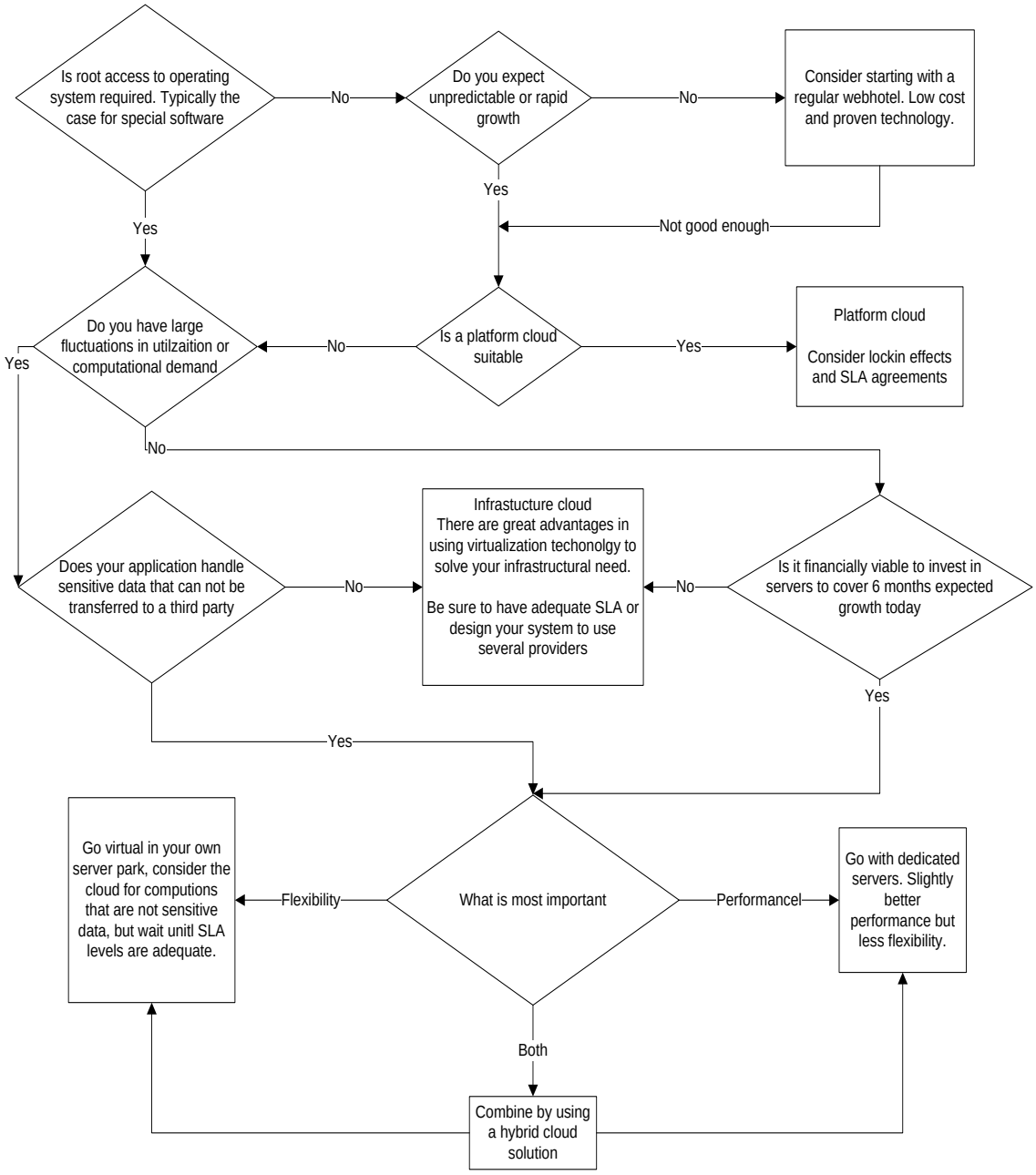


Figure 7 - Suggested framework for deciding upon an infrastructure solution

Chapter 6 - Case study – Quinyx FlexForce

This chapter will use the knowledge from previous chapters to analyze the FlexForce servers provided by Quinyx FlexForce AB (QFAB).

6.1. Quinyx FlexForce AB – FlexForce scheduling and communication service

FlexForce is QFABs scheduling product that provides their customer with a completely hosted service for managing a complex scheduling solution. The service focuses on high availability; quick and efficient communication through email and SMS; real time editing; and cost efficiency.

Each FlexForce customer has a local list of users where some users are administrators that manage and classify scheduling needs and others are regular users that check the end results and give feedback on current or planned schedules.

The system pushes changes instantaneously to all users[*] through Adobe's Live Cycle services that act as a newsfeed for changes in the database which in turn inform clients to update their workspace. Since the application uses Adobe Flex to generate a frontend, the need for a dedicated web server is smaller than for a classic web application. Services from the application server are provided directly to the client interface. The client interface in turn takes care of the presentation of all of the data. Service calls are made in the background and almost always results in one or more SQL database lookups. The server setup is described in Figure 8.

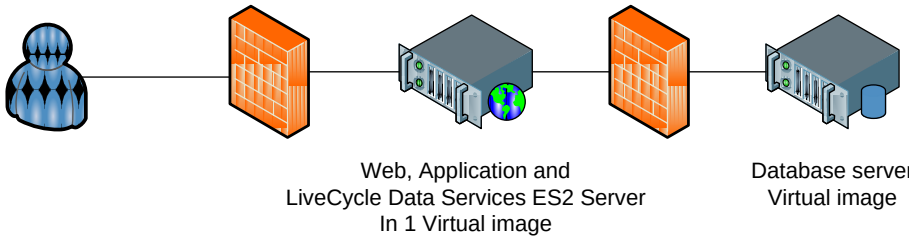


Figure 8 - FlexForce infrastructure layout, 3rd party integrations excluded

6.2. Design structure.

FlexForce is a multi tenant system where database tables are shared among customers. However, except for a few tables, used for shared bug tracking and feature requests, every row in the database can be associated with a single customer that has sole access to the data.

The current design does not make use of an abstract database layer to access data, but does make use of a shared database initiation script[†] and other techniques used provide similar functionality. The users log in using their email addresses and password. The email is uniquely bound to a single FlexForce customer.

[*] Delays up 3 seconds have been recorded.
[†] The implementation of a DB abstraction layer is in the R&D pipeline.

6.3. Log analysis

When analyzing Quinyx FlexForce* there are a few facts worth noting, namely:

- 4% of all requests are used to download the user inteface (UI) file
- Downloading the UI accounts for over 64% of the bandwidth usage.
- The weekly and daily usage is predictable and change in a predictable manner.
- 64% of all requests served are user generated service calls.
- 9% of all requests are third party integrations and account for 9% of bandwidth used
- The lowest weekly load is during 2am and 6 am on Saturdays (See Figure 9 and Figure 10).
- The service peaks at about 3000 concurrent users.
- Each visiting user performs roughly 25 requests to the application server. With the current statistics it is not possible to separate administrators from regular users. QFAB's estimatations suggest that administrators perform 5 times as many requests as a regular user.
- 2-5 % of the users are administrators.
- Each user stays on an average 20 minutes,
- Each request to tier 1 produces 7-15 database queries (with an average of 11 queries per request).

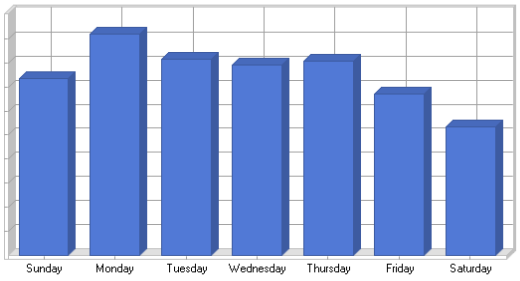


Figure 9 - Weekly usage

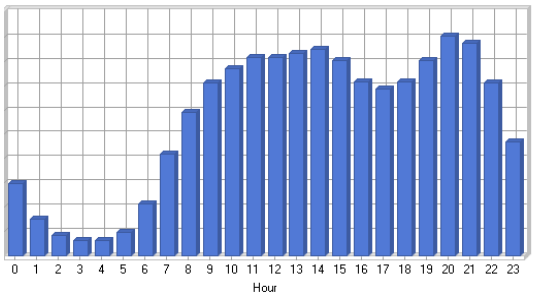


Figure 10 - Hourly usage

6.4. Known bottlenecks

QFAB has from experience recognized that the MySQL server as a bottleneck, but solved this issue by upgrading the hardware. After the upgrade, the SQL server has never spiked over 10% CPU utilization and the MySQL slow query log has very few entries.

6.5. Applying suggested framework.

Key issues when scaling FlexForce are:

- The client application allows for load balancing at the client side if server status can obtained.
- There are small number of updates of the database compared to number of reads in the database.
- Data in the database can be decoupled based upon customer ID.
- Personal and sensitive information is stored in the database and this data is subject to discretion.
- The application is classified as business critical by many of the FlexForce customers and SLAs are defined accordingly.

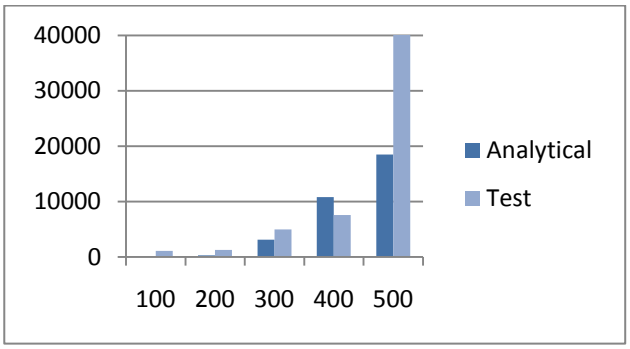
* Raw data is gathered from server logs and can due to client discretion not be made public.

- QFAB has sufficient infrastructure and financial strength to handle the expected growth for the next years no matter what scaling alternative is selected.
- Root access is needed for the Adobe Life Cycle server application.

From these key issues we can draw a few conclusions about scaling:

- Tier 1+2 scaling can be achieved by using either using load balancers in the server environment or by adapting the client code to select a server with the least load (and providing the clients with information about the server(s) load(s)).
- Using a light weight web server for the initial client download enables CPU cycles and bandwidth to be saved.
- Tier 3 scaling can today be achieved by vertical scaling, replication, and sharding (horizontal scaling).
- Both cloud and self hosted solutions are viable options if an adequate safety (relaiability, transparancy and compenstation) level can be achieved in the cloud*.

We start by applying the queuing theory presented earlier (in section 3.2) by using using parameters derived from logs and with the desired maximum number of concurrent users set to represent a stress test. The analysis is based upon information that QFAB had in spring 2010. test the theoretical analysis using actual logged data in order to check whether the theoretical model can be used for analyzing FF. As we can see in Figure 11 the analytical estimate and the actual test do **not** correlate as well as hoped. This is probably due to the simplification of the algorithm. The in depth algorithm requires knowledge about system utilization during operation which was not available to the author. However, we can observe that both the estimate and the actual test results were of the same order of magnitude and that a logarithmic correlation exists. Tweaking the input parameters to fit the stress test is not in our interest at this time, but we assume that the analytical aspect is sufficiently accurate to give scaling advice. Hence we will use the model and adjust our input parameters to better understand which parameters will have the most effect on scaling.



Parameters used in analysis 1.
- Think time : **18sec**
- Max concurrent sessions :**500**
- Tier 1 service time :**150ms**
- Tier 2 service time : **7ms**
- Average visit ratio to T2: **11**

Figure 11 - The plot on the left illustrates FlexForce response time as a function of concurrent sessions.

* This requirement also refers to being able to prove security to a customer.

We adjust the parameters to the analytical model according to Table 2and analyze the theoretical effects*. The results are found in Figure 12 and Figure 13

Table 2 - Input parameters for the analytical model

		From	To	Step size
Tier 1	service time	100ms	400ms	100ms
	number of servers	1	4	1
Tier 2	service time	4ms	12ms	4ms
	number of servers	1	4	1
	visit ratio	10	30	10
Static	Think time	75 sec	-	-
	Max sessions	300	-	-

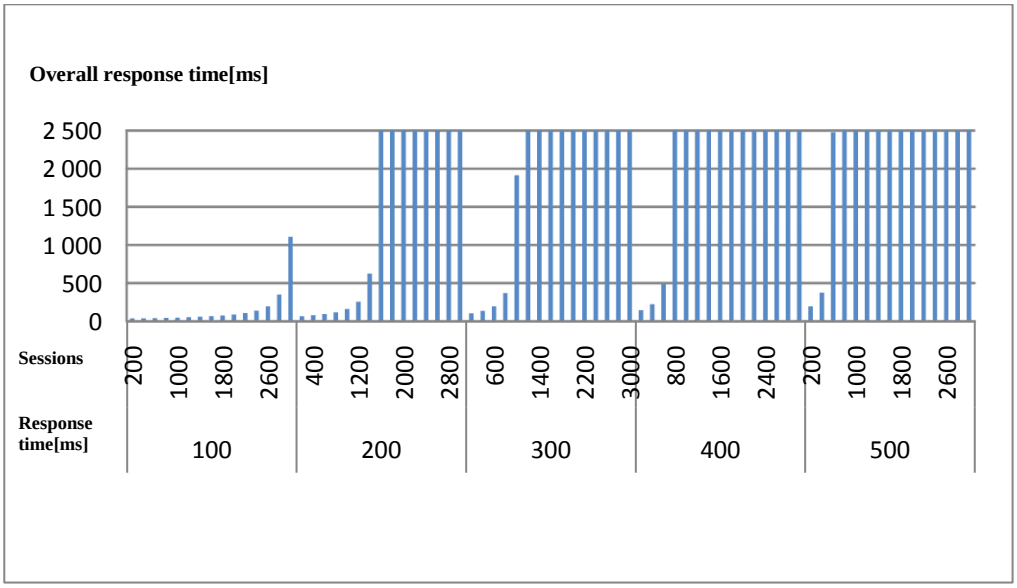


Figure 12 - Analytical effects: Tier 1 response time[ms] as a function of concurrent sessions and service time[ms]. Constant values are Tier 1 servers (4), Tier 2 service time (4ms), Tier 2 servers (4) and Tier 2 service ratio (20).

* The calculations were made with a java application that can be obtained from appendix C.

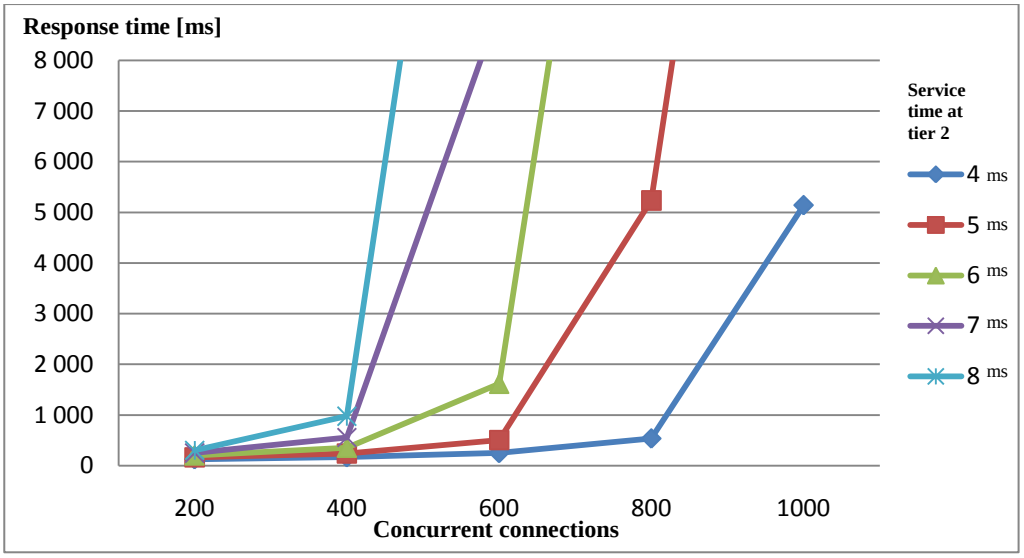


Figure 13 - Analytical effects: Response time[ms] as function of concurrent connections (x-axis) and tier 2 service time[ms] (data series). Constant values are Tier 1 service time (100ms), Tier 1 servers (1), Tier 2 servers (1), Tier 2 visit ratio (20) and Tier

The analytical model indicates that due to the large number of requests made to the database for each application request, system performance is very sensitive to an increase in the response time at tier 2. In Figure 13 we can see that at 800 concurrent connections, an increase in the service time at Tier 2, from 4ms to 5ms, leads to approximately 10 times longer response time! Long time spent processing in tier 1 makes the throughput in the application server less than of the database. As seen in Figure 12, problems occur quickly when congestion takes place in tier 1. These findings are summarized in Table 3

Table 3 - Analytical findings

	Primary risk	Mitigation
Tier 1, Application server	Too many concurrent sessions	Increase capacity before 2000 concurrent sessions
Tier 2, Database server	Too high response time	Increase capacity before the average query takes longer than 5ms including connection time.

A preferred solution for the FlexForce application would be to use the flexibility of the cloud, but never to be fully dependent on one cloud provider and never to lose control of data. Using a cloud solution for the application server(s) and a remote database (to get around issues with secure data deletion and third party trust) is not currently viable due to the effect a small increase in database response times has on the overall performance.

The Amazon AWS service provides vertical scaling up to 64GB of RAM and 26 computational cores. With this underlying hardware a scaling up approach for the database is expected to be a viable option for at least 2 years, thus giving FlexForce time to adapt to sharding techniques. The problems raised in section 3.2.5 have to be solved in order for this to be a viable option. The solution could be use of a virtual machine with a lot of RAM to store the whole database in the RAM, in order to be independent of the on disk access times which except for performance benefits would also remove much of the uncertainty of secure data deletion. To keep data secure in case of failure a replicated slave, hosted in another data center, could be used to store the data to a physical disk.

This option also has the advantage of geographical failover in case the Amazon service fails* since data is spread across 2 regions. Figure 14 illustrates this scenario.

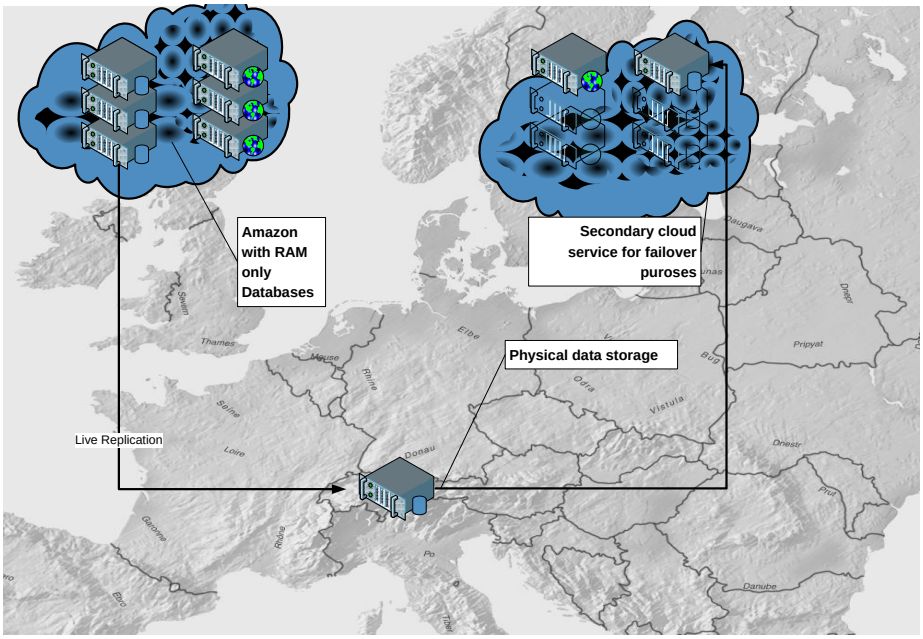


Figure 14 - Cloud failover schema

To test this approach, a simple test was conducted. The details and results of this test can be found in appendix B. As expected, the slave replication has minimal performance impact on the master, therefore the solution is a viable option. The test also indicates that the most utilized resource in the database is the CPU and not the disk. Hence scaling the database performance would require more and faster CPUs. Amazon can offers many "computational units", but they tend to be slower than most of their competitors. Slower computation may have a negative effect on overall performance as slow processes may lock up resources for other processes thus slowing the overall process. A better solution in this case would be to use a client side processes for the complex computation instead of a servers side computation or switch operations such as these to a batch queue.

It is possible for QFAB to scale their application in many ways. They can use a cloud service while maintaining control over sensitive data. Using a cloud would solve most of their problems for the coming year(s), but since their usage is predictable and does not to fluctuate, a privately owned server park setup is also a viable solution.

Applying the economical calculations in section 4.3 with the following values:

R_s	0.5% - The risk of extreme surge is deemed unlikely
R_D	[Blanked out] (Estimated by the CTO of UCMS)
R_{SLA}	0.5 + 0.9 (Amazon SLA & inadequate compensation)
C_H	0,4$
C_C	0,54$ (2 Large + 2 Extra large + 200GB traffic + 1 private server for DB backup, translated into hourly costs per server)
N	*4*
U	*10%*
I	[Blanked out] (Estimated by the CEO of QFAB)

* DNS based geographical failover comes with a maximum downtime equal to the Time to life setting in the DNS entry. This time is usually between 5-15 minutes and considered by many a poor solution but it does prove a worst case scenario for a cheap failover system.

The result is a cost factor of roughly 0.7 which indicates that the completely self hosted environment is cheaper. The is due to the high costs associated with a failure and that the risk of failure is, by our model, deemed to be higher in the cloud.

We summarize the case analysis by concluding that FlexForce can be scaled. The structure of the database allows for sharding (the most complex and effective scaling technique of databases) which is good since we, in the theoretical queuing theory analysis, noted that the database access was the most sensitive link in the application chain. Scaling can be done both with cloud technology and with self hosted servers. The decision to use the cloud as an option is not today economically favorable, but could be with increased capacity needs and/or if the cloud providers provide better SLAs. For now I recommend that QFAB continue to use their current infrastructure, but to keep an eye on SLAs for different cloud providers. The flexibility of the cloud could prove very attractive in the future.

Chapter 7 - Conclusions and Future Work

7.1. Conclusions

Even though there is no out of the box solution for scaling, there are methods to scale almost every server type. Some design considerations will ease the process of scaling and should be implemented as soon as scaling might become an issue. This thesis has presented the most common practices for scaling a 3-tier web application and has shown different issues to be examined when scaling SaaS applications.

We have shown that the scaling principles presented in this thesis can be applied to the FlexForce application. Moreover we have not found any reason as to why these lessons could not be used with other applications.

Virtualization has changed the way server capacity can be consolidated and utilized. Except in a very few cases virtualization is a great tool to allow for dynamic scaling. To what extent the cloud providers will manage to attract business critical applications is a matter of adapting their SLAs and infrastructure transparency to meet market expectations as well as convincing the community that there are adequate safety measurements in place. There is an already ongoing process in this direction, but it is far from complete. If businesses decide to move applications into a public cloud today their main concerns are security related issues, including uptime assurance and privacy of data. Legal aspects will have a part in the decisions of whether or not to use the cloud which may benefit local cloud providers over global actors.

7.2. Future Work

As more cloud providers enter the market, a comparison of performance compared to cost metrics, as well as an independent security analyses will be needed for corporations to make informed decisions. With increased use of cloud computing it is likely we will see more geographically distributed systems, even for smaller businesses, to increase redundancy and performance. This increase will host new scaling and performance issues that need further research.

This also thesis presented an analytical model for evaluating the economical impacts of a cloud solution that will require further research and fine tuning.

Appendices

Appendix A	Sniffing a cloud, a network security analysis of City Cloud
Appendix B	Remote database for enhanced security.
Appendix C	Java source code used to derive average waiting times in FlexForce case study.

Literature and references

Amazon Web Services. "Amazon Web Services." *Overview of Security Processes.* Amazon webservice. November 2009. http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf (accessed April 20, 2010).

Apache Software Foundation. *Apache - HTTPD server project.* 2009. http://httpd.apache.org/ (accessed April 23, 2010).

Armbrust, Michael, et al. *A Bearkley view on cloud computing (pp 10).* UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2010.

Avila, Joseph A., Nathaniel J Mass, and Mark P. Turchan. "McKinsey Quarterly." *Is your growth strategy your worst enemy.* McKinsey&Company. May 1995. https://www.mckinseyquarterly.com/Is_your_growth_strategy_your_worst_enemy_92 (accessed May 01, 2010).

Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi,"*An analytical model for multi-tier internet services and its applications"*, In proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '05), ACM, New York, NY, USA, June 2005, ISBN 1-59593-022-1, pages 291-302, http://doi.acm.org/10.1145/1064212.1064252 (accessed May 10, 2010).

CA.com. *Unleashing the Power of Virtualization 2010.* CA. Feburary 2010. http://www.ca.com/se/survey2010 (accessed May 01, 2010).

Chamas Enterprises Inc. *Apache::ASP.* http://www.apache-asp.org/ (accessed April 22, 2010).

Coarfa, Cristian, Peter Drusche, and Dan S. Wallach. "Association for Computer Machinery." *Performance analysis of TLS Web servers.* Rice University. Feburary 2002. (accessed April 29, 2010).

Djurberg, Joakom Arstrad. "Dåliga på molnet." *Computer Sweden*, 2010, 23 april ed.

Dormando. *Memcached.* 2009. http://memcached.org/ (accessed April 16, 2010).

Durkee, Dave. "Why Cloud Computing Will Never Be Free." *Communications of the ACM*, May 5th, 2010: pp62-69.

Enomaly.com. *Enomaly.com.* Feb 2010. http://src.enomaly.com/ (accessed 05 20, 2010).

Eucalyptus Systems. *Eucalyptus White Paper Aug 2009.* August 2009. http://www.eucalyptus.com/themes/eucalyptus/pdf/EucalyptusWhitepaperAug2009.pdf (accessed Mars 28, 2010).

Sylvie Feindt, Judith Jeffcoate, and Caroline Chappell, "*Identifying Success Factors for Rapid Growth in SME E-commerce*", Journal Small Business Economics, Springer Netherlands, Volume 19, Number 1, August, 2002, pages 51-62, ISSN 0921-898X (Print) 1573-0913 (Online), DOI 10.1023/A:1016165825476

Fred P. Brooks, Jr. "The Mythical Man-Month: Essays on Software Engineering." In *The Mythical Man-Month: Essays on Software Engineering*, 25. Anniversary Edition (2nd Edition), Addison-Wesley Professional, August 1995 (first released 1975).

Google Trends. accessed June 2, 2010. http://www.google.com/trends?q=cloud+computing.

Vipul Goyal and Rohit Tripathy, "*An Efficient Solution to the ARP Cache Poisoning Problem*", In proceedings of Information Security and Privacy, 10th Australasian Conference (ACISP 2005), Colin Boyd and Juan Manuel González Nieto (Eds.), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 3574, July 2005, pages 40-51, ISSN 0302-9743 (Print) 1611-3349 (Online), http://dx.doi.org/10.1007/11506157_4 (accessed April 25, 2010).

Gustavsson, Björn, and Caroline Agholme. *Cloud computing.* ICT easyfairs, Kista. Feburary 10, 2010

Gutmann, Peter. University of Auckland, Department of Computer Science. January 10, 2003. http://www.usenix.org/publications/library/proceedings/sec96/full_papers/gutmann/ (accessed April 27, 2010).

Halderman, J. Alex, et al. "LestWe Remember: Cold Boot Attacks on Encryption Keys." Princeton University, Electronic Frontier Foundation and Wind River Systems. Feburary 21, 2008. http://citp.princeton.edu/pub/coldboot.pdf (accessed April 20, 2010).

Hellström, Madelende. "Forskarens varning: Lite inte på molnet." *Computer Sweden.*, 2010, 16 Mars ed.

HyperSQL. *Features.* December 10, 2009. http://hsqldb.org/web/hsqlFeatures.html (accessed Mars 04, 2010).

jetNEXUS. *Hardware Load Balancer.* http://www.hardwareloadbalancer.com/index.htm (accessed May 20, 2010).

Keith Adams and Ole Agesen, "*A comparison of software and hardware techniques for x86 virtualization*", In Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (ASPLOS-XII), ACM, New York, NY, USA, March 2006, ISBN 1-59593-451-0, pages 2-13, http://doi.acm.org/10.1145/1168857.1168860

Kerwin, Doug. *sql-server-performance.com.* May 10, 2003. http://www.sql-server-performance.com/articles/clustering/massive_scalability_p1.aspx (accessed June 2, 2010).

LigHTTPD. Lighttpd.net. March 5, 2007. http://www.lighttpd.net/benchmark/ (accessed May 8, 2010).

Mahemoff, Michael. *Ajax Design Patterns.* O'Reilly Media, Inc., 2006.

MDB2. *Pear.php.net.* 2009. http://pear.php.net/package/MDB2 (accessed June 2, 2010).

Mell, Peter, and Tim Grance. "National institue of Standards and Technology." *Definition of Cloud Computing, v15.* September 10, 2009. http://csrc.nist.gov/groups/SNS/cloud-computing/ (accessed June 2, 2010).

Microsoft Corperation. *Microsoft Technet.* 2010. http://technet.microsoft.com/en-us/library/cc740265%28WS.10%29.aspx (accessed May 10, 2010).

MySQL. *MySQL Cluster.* 2010. https://www.mysql.com/products/database/cluster/faq.html (accessed June 2, 2010).

Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha, "*Analyzing the energy consumption of security protocols"*, In Proceedings of the 2003 international symposium on Low power electronics and design (ISLPED '03), ACM, New York, NY, USA, August 2003, pages 30-35, ISBN 1-58113-682-X, http://doi.acm.org/10.1145/871506.871518 (accessed April 29, 2010)

Netcraft.com. Netcraft. 04 2010. http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html (accessed April 20, 2010).

Olsson, Jan, and Per-Hugo Skärvad. *Företagsekonomi 100 (part 3).* Malmö: Liber, 2007.

Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell, *Modeling virtual machine performance: challenges and approaches*, SIGMETRICS Performance Evaluation Review, ACM, New York, NY, USA, volume 37, number 3, December 2009, ISSN 0163-5999, pages 55-60, http://doi.acm.org/10.1145/1710115.1710126 (accessed May 1, 2010).

Osipov, Konstantin, and Peter Gulutzan. *MySQL Conference and Expo.* April 22, 2009. http://www.mysqlconf.com/mysql2009/public/schedule/detail/6814 (accessed May 20, 2010).

Riboe, Jens. *Vad är Cloud computing.* EasyFairs, ICT fair, Kista. 17th of Feb 2010.
SalesForce.com. *Sales Force.* http://www.salesforce.com/crm/sales-force-automation/ (accessed June 2, 2010).
Scholl, Matthew, et al. "U.S. Department of Health & Human Services." *Guidelines for Media Sanitization.* National Institute of Standards and Technology (NIST). September 2006. http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/nist80066.pdf (accessed April 27, 2010).
Shabat, Gil. *Novologies.* Mars 25th, 2009. http://www.novologies.com/post/2009/03/25/Database-Scale-Out-Strategies-for-Web-Apps.aspx (accessed June 2, 2010).
The European Parliament And The Council, Of The European Union. "European Commission." *Fredom, Secureity and justice - Data protection.* October 24, 1995. http://ec.europa.eu/justice_home/fsj/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf (accessed Mars 20, 2010).
The Project Manangement Hut. *PM Hut.* 2010. http://www.pmhut.com/benefits-of-using-the-raci-arci-matrix-in-project-management (accessed April 27, 2010).
UCMS Group. *UCMS Group.* http://www.quinyx.com/#sub=1;cat=1 (accessed May 20, 2010).
Yaozu Dong, Jinquan Dai, Zhiteng Huang, Haibing Guan, Kevin Tian, and Yunhong Jiang, "*Towards high-quality I/O virtualization*", In Proceedings of SYSTOR 2009 (SYSTOR '09), The Israeli Experimental Systems Conference, ACM, New York, NY, USA, 2009, ISBN 978-1-60558-623-6, pages 1-8, http://doi.acm.org/10.1145/1534530.1534547
Zaitsev, Peter. MysqlPerformanceBlog.com. July 7, 2006. http://www.mysqlperformanceblog.com/2006/07/07/thoughts-on-mysql-replication/ (accessed June 2, 2010).

APPENDIX A: SNIFFING A CLOUD

NETWORK SNIFFING IN A CLOUD ENVIRONMENT

A report on how co-tenants' network traffic can be intercepted in a poorly configured cloud environment.

ABSTRACT

This report shows how a faulty or naive configuration of a public cloud can be used to intercept traffic from other co-tenants in a cloud environment. This report shows how HTTP traffic can be intercepted and used to perform an attack, but the reader should note that intercepting unencrypted traffic of any kind can be equally fatal to privacy. The reason for choosing HTTP for the analysis was simply to easily find a vulnerable host as quick as possible.

The cloud provider in this report, City Cloud (http://www.citycloud.se/), is a Swedish provider of Cloud infrastructure services. City Cloud has been informed of this potential threat before the publication of this report. No logs were kept and no unauthorized logins have been made. Except for initial sniffing used for discovery of hosts, all analyzed traffic has been generated by the author.

VARIABLES USED

In this report I will use the following variables to mask out IP addresses for discretion.

IP_VM	The public IPv4 address of a virtual host residing in the cloud.
IP_REMOTE	The public IPv4 address of a computer sending HTTP requests
IP_X#	Random sniffing target number #

TEST ENVIRONMENT.

All of the tests were conducted in City Cloud's virtual environment with the addition of a single remote host outside the cloud that generates HTTP requests. There were two configurations of machines: configuration 1 for the machine image to be executed in the cloud and configuration 2 for the remote host outside the cloud, these are shown in tables 1 and 2. A logical view of the network environment is shown in Figure 1.

Table 1 - image in the clooud

Hardware	
Machine template used	: Windows 2008 DC With MSSQL Std (19.53GB)
Machine hardware template used	: "Small" = 512Mb RAM, 1 CPU @ ~2.4Ghz.
Software used	
Wireshark 32bit 1.2.7 with included WinPCap driver(4.1.1).	
Software configuration	
Wireshark is set to listen to traffic on the active network card with the public IPv4 address.	

Table 2 - remote host (outside the cloud) sending HTTP requests

Hardware used
AMD x64 quad core at 2.8Ghz
8GB DDR2 RAM
Connection:
ADSL, 24Mbit downlink, 2Mbit uplink.
Software used
Windows 7
Mozilla Firefox/3.6.3

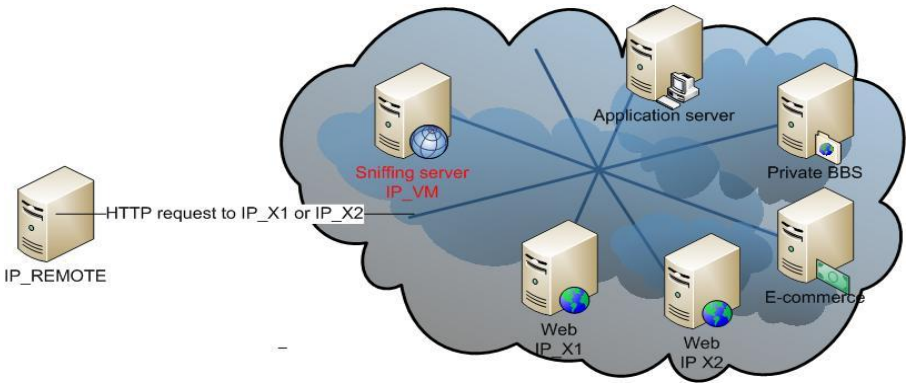


Figure 1 - Logical network diagram of the overall test environment

TEST SCENARIO

Step 1: Find other hosts

Run Wireshark with the filter “http” on. This will generate a list of available HTTP hosts on the network that are visible to your host and that have been access by someone (i.e., these machines received a HTTP request) after you started sniffing.

Note some of the IP addresses discovered (see the column labeled ”Destination”).

Step 2: Start logging

Stop sniffing and start a new recording of network activity with the filter “http && ip.src == IP_REMOTE”. As noted above, this will capture the traffic that your remote hosts sends that can be heard at the host where you are running Wireshark.

Step 3: Browse the hosts

From your remote computer enter one of the IP addresses found in step 1 in the webbrowser of your remote host. If this host responds, then you have found a web server that you can browse. Find a login form or search form and try to login with any random strings (e.g. test:test)

Repeat this step for each IP address found in step 1.

Step 4: Analyze intercepted packets

In the previous step, Wireshark captured all your communication with the different web servers including the login attempts.

TEST RESULTS

Test 1

Login attempt on host IP_X1 with credentials “test:test”

Package dump: (some data has been masked out for discretion).

```
TElEb2y@w;Q"[{cwPu _,"PA:5?POST /←MASKED OUT→ HTTP/1.1
Host: ←IP_X1→/
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; sv-SE; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: sv-se,sv;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http:// ←IP_X1→/
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
username=test&password=test&…←MASKED OUT→
```

As seen above the password and username provided to a neighboring host are visible to the sniffing host.

Test 2

The second webserver provided more content.

Package log: (some data has been masked out for discretion).

```
18008 1779.699179 ←IP_REMOTE→ ←IP_X2→ HTTP GET /modules/system/system-menus.css?y HTTP/1.1
18009 1779.706278 ←IP_REMOTE→ ←IP_X2→ HTTP GET /modules/user/user.css?y HTTP/1.1
….
18015 1779.728181 ←IP_REMOTE→ ←IP_X2→ HTTP GET /sites/all/modules/cck/theme/content-module.css?y HTTP/1.1
…
18068 1779.911508 ←IP_REMOTE→ ←IP_X2→ HTTP GET /misc/jquery.js?y HTTP/1.1
…
18074 1779.943840 ←IP_REMOTE→ ←IP_X2→ HTTP GET /misc/drupal.js?y HTTP/1.1
```

The data posted from the login form is once again completely visible.

```
aElE{#@wLQ"[{P8nRP@WPOST /?q=user/login HTTP/1.1
Host: ←IP_X2→
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; sv-SE; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: sv-se,sv;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http:// ←IP_X2→/?q=user/login
Cookie: SESS5c1155aeed184a5c90e22bb859b63e60=a8a421e05e5374bc99ea33c47a3c5e12; has_js=1;
__utma=19638512.309671263.1271871441.1271871441.1271976451.2; __utmc=19638512;
__utmz=19638512.1271871441.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); Drupal_l10n_client=0;
__utmb=19638512.2.10.1271976451
Content-Type: application/x-www-form-urlencoded
Content-Length: 100
name=test&pass=test&form_build_id=form-
20d548a213ca30c82a4bc71318860ba3&form_id=user_login&op=Log+in
```

Worth noting is that since each intercepted GET and POST form contains the session identifier it is easy to do a session hijacking of an already logged in user and gain all s/he's privileges. It is not necessary to log the actual login attempt, any logged page request by a logged in administrator would grant a hacker full access.

CONCLUSION

Though this test does not show to what extent traffic is available to co-tenants of the "City Cloud" service it does proves that some traffic is available to more than the designated host. This is a security breach since a malicious uses could use unencrypted data to gain access to systems within the cloud. To circumvent this problem all users of City Cloud are advised to only use **encrypted** communication at least until City Cloud has resolved the issue. The author has not yet tested other cloud providers for similar weaknesses. Amazon AWS claims that their cloud service is free from this problem, but this has not yet been verified by the author.

APPENDIX B: REMOTE DB

UTILIZING REPLICATION TO ENHANCE SECURITY IN A CLOUD ENVIRONEMNT

A report on how Quinyx FlexForce' application can be scaled in the cloud, but still maintaining control over sensitive data.

REPORT SUMMARY

This report will analyze the performance implications of using a hybrid cloud solution for scaling SaaS applications while maintaining control over sensitive data. The report analyzes how FlexForce, a SaaS application, acts when changing the database settings and using a remote database for backup purposes.

The report will consider 3 cases and briefly look at the performance effects.

MOTIVATION

Moving a SaaS application into the cloud increases the potential for scalability but, comes with some legal and security related issues that need to be addressed. We will focus on one issue, namely the unwillingness to transfer a complete data set to a cloud due to the potential security risks involved with secure data deletion from a physical disk. We setup a test environment where data is never physically stored in the cloud and compare the performance compared to other configurations.

CASE SUMMARY

Case 1 - The reference case

The database is stored locally on a physical disk.

Case 2 – Use of RAM disk

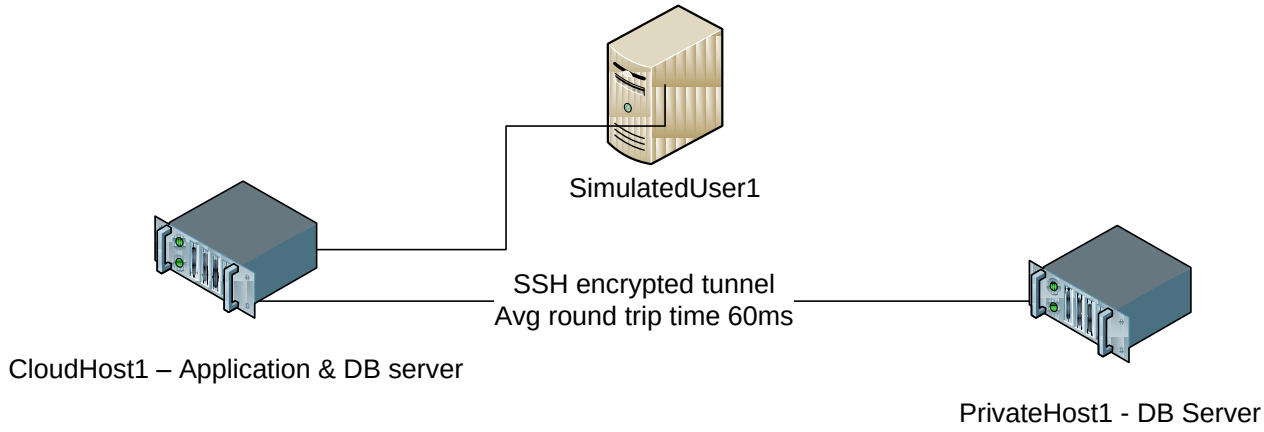
We utilize the cheap alternatives provided by cloud computing to store a database fully in the RAM. This option should increase performance and remove problems with secure data removal by never storing data on a physical disk. This is ensured using a RAM-FS that is never paged to disk.

Case 4 – RAM disk with replication

This case addresses the reliability issues of case 2, where a power failure would cause a complete loss of the whole database. Replicating the database to a slave (in another datacenter) with a physical disk 2, gives several advantages: 1.Physical storage of data in a controlled environment, 2.geographical failover having the data in 2 geographically separated locations offers the ability restore data to another server park in case of failure of the first server park.

Note: *None of the cases will simulate a real environment with actual user loads. The tests were developed to maximize the load on the server to compensate for this.*

TEST ENVIRONMENT



CloudHost1

Amazon virtual host, eu-west-1a region
m1.large (7.5 GB memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units), 64-bit platform)
Debian "Lenny"
Database stored on a RAM FS mount point.

PrivateHost1

8GB DDR2, 1066MHz RAM, AMD Phenom II X4 925 Processor, 2800 MHz
Gentoo 2010, x64, Linux kernel 2.6
24/2 Mbit/second ADSL connection

SimulatedUser1

3 GB DDR2 RAM,
Intel Core 2 Duo CPU T8100 @ 2.10GHz
Windows 7 Professional 6.1.7600 Build 7600
24/2 Mbit/second ADSL connection

Round trip times between the different servers are indicate in the table below.

	Cloud host 1	**Private host 1**	**Simulated user 1**
Cloud host 1	-	58ms	58ms
Private host 1	58ms	-	-
Simulated user 1	58ms	-	-

Note: *The test environment is far from optimal since both the PrivateHost1 and SimulatedUser1 shares the same connection and that this is not a typical server connection (i.e., typically we would expect this server to be connected by a much higher speed link that was symmetric in uplink and downlink data rates). Also the Amazon EC2 compute units correspond to 1.2 GHz-1.4GHz of computational power which is well below the average speed of a modern server.*

MYSQL SERVER CONFIGURATION.

All servers have MySQL 5.1.3X installed.
All servers have the following settings set

```
[mysqld]
skip-external-locking

key_buffer              = 16M
max_allowed_packet      = 16M
thread_stack            = 128K
thread_cache_size       = 8

query_cache_limit       = 1M
query_cache_size        = 16M


expire_logs_days        = 10
max_binlog_size         = 100M
skip-bdb

sloq_query_log = 1
long_query_time = 0
```

The only differences between test configurations are the master/slave specific settings in each case.

MEASUREMENTS

All measurements are derived from a MySQL "slow query log". Turning on the slow query log has performance implications that will affect the results, but since we are interested in differences between solutions and not actual times we deem this performance penalty acceptable. However, this approach only holds a precision of 1 second.

Two operations are measured. Operation number one selects a few objects from the database and inserts 1024 objects. This is a simple insert operation that will update indexes and check foreign key constraints. The second operation selects data, analyzes it and performs an insert operation - this process is repeated 1024 times. The SQL SELECTs are simple (singe element, searches using indexes) but there are 15 times as many as SELECTs as the number of updates. The simulated computation in the application server is fairly simple. Most of the load is due to the MySQL server executing selects and updating the database.

Results

Each of the above operations were repeated three times - see table below

Case	Operation 1 (seconds)				Operation 2 (seconds)			
	Run1	Run2	Run 3	Average	Run1	Run2	Run 3	Average
1	8	7	8	7,6	419	420	398	412
2	8	7	7	7,3	435	390	406	410
3	9	8	8	8,3	397	408	410	405

In case 3 the time from action to a complete replication was less than 10 seconds.

CONCLUSIONS

We observed that the standard deviation in these measurements was too large to be able to draw any conclusions about a change in performance. However, if we were to assume that there is no noticeable difference in performance between each case, than we can draw the following conclusions:

- In our test the limiting resource is CPU cycles and not the disk speed. In a production environment where more than 1 operation occurs simultaneously it is likely that using a RAM disk will have a larger effect.
- The replication process had minimal impact on the master server's (CloudHost1) performance and could be achieved even with a limited connection.

APPENDIX C: SOURCE CODE

QUEUING THEORY

The source code used to do queue calculations in the case analysis. Based on Bhuvan Urgaonkar et al.'s research (2005).

```java
/**
 * Java application used to do queuing calculations.
 * Based on Bhuvan Urgaonkar et al's research (2005)
 * @author Mikael Rapp, 2010
 */
import java.util.Hashtable;
public class Main {

    static int S1Servers, S2Servers, S1Time, S2Time;
    static float S1Ratio, S2Ratio;

    public static void main(String[] args) {
        S1Servers= S2Servers= S1Time= S2Time = 0;
        S1Ratio = S2Ratio = 1;

        for(int i = 10; i <= 30; i = i + 10){
            //S2 ratio, 10, 20, 30
            S2Ratio= i;
            for(int j = 1; j <= 4; j++){
                //S2 servers, 1, 2, 3, 4
                S2Servers= j;
                for(int k = 4; k <= 14;  k++){
                    //S2 time, 4->14
                    S2Time = k;
                    for(int l = 1; l <= 4; l++){
                        //S1 servers, 1,2, 3, 4
                        S1Servers = l;
                        for(int m = 100; m <= 500; m = m + 100){
                            //S1 time, 100->500
                            S1Time = m;
                            Calc(3000/*Max concurrent sessions*/,75 /*User think time*/);
                        }
                    }
                }
            }
        }
    }

    //Will calculate the average queing time and print it.
    private static void Calc(int maxSessions, int thinkTime) {
        Hashtable Sq = new Hashtable();
        Hashtable Vq = new Hashtable();
        Hashtable Rq = new Hashtable();
        Hashtable Dq = new Hashtable();
        Hashtable Lq = new Hashtable();

        //Just to be on the safe side we clear them all.
        Sq.clear();        Vq.clear();
        Rq.clear();        Dq.clear();
        Lq.clear();

        //Will hold our thoughput later on
        float tao=0;

        //Fill the hashtables with their values.
        Sq.put(1, S1Time);
        Vq.put(1, S1Ratio / ((Number)S1Servers).floatValue() ); //Force it to be a float calc
```

```
        Sq.put(2, S2Time);
        Vq.put(2, S2Ratio / ((Number)S2Servers).floatValue() ); //Force it to be a float calc

        Rq.put(0,thinkTime);
        Dq.put(0,thinkTime);
        Lq.put(0,0);


        try{
            for(int i = 1; i <= 2; i++){
                Lq.put(i,0);
                Dq.put(i,
                        ((Number)Sq.get(i)).floatValue()
                        *
                        ((Number)Vq.get(i)).floatValue()
                    );
            }

            //Add a customer one by one.
            for(int i = 1; i <= maxSessions; i++){
                /* avg. delay at each que.*/
                for(int j = 1; j <= 2/*numb of tiers */;j++){
                    Rq.put(j,
                            //Ri = Di(1+Li)
                            ((Number)Dq.get(j)).floatValue() *(1+ ((Number)Lq.get(j)).
floatValue())
                    );
                }

                //Calculate tao (the thorughput)
                tao =  i/(summa(0, /*numb of tiers */2, Rq) );

                //Loop through the 2 tiers
                for(int j = 1; j <= 2;j++){
                    //Update the queue length. (Littles law)
                    Lq.put(j, tao* ((Number)Rq.get(j)).floatValue() );
                }

                Lq.put(0, tao * ((Number)Rq.get(0)).floatValue());

                //Only print every 200 results. Save CPU time and computational time later
                if((i %200) == 0)
                     System.out.println(  i + " " + //Time
                          S2Ratio  + " "    +//T2 visits rq
                          S2Time  + " "     +//T2 time
                          S2Servers + " "   +//T2 servers
                          S1Servers  + " "  +//T1 servers
                          S1Time // T1 time+
                         +" " + summa(1,2, Rq)
                    );
            }
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
```

```
    //Sums up elements [start] to [stop] in the hashtable and returns result.
    static float summa(int start, int stop, Hashtable list){
        float sum = new Float(0);
        for(int i = start; i<= stop; i++){
            sum = sum + ((Number)list.get(i)).floatValue();
        }
        return sum;
    }

}
```

TRITA-ICT-EX-2010:107

www.kth.se