Autonomic Fault Tolerance Using HAProxy in Cloud Environment

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Vishonika Kaushal
(Roll No. 800931024)

Under the supervision of:
Mrs. Anju Bala
(Assistant Professor)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Autonomic Fault Tolerance Using HAProxy in Cloud Enviornment"**, in partial fulfilment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mrs. Anju Bala* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Vishonika Kaushal
(Vishonika Kaushal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Anju Bala
(Mrs. Anju Bala)
Assistant Professor
Computer Science and Engg. Deptt.
Thapar University
Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

Many people have shared their time and expertise to help me accomplish my goal. First, I would like to sincerely thank my guide, Mrs. Anju Bala, Assistant Professor, Thapar University, Patiala for her constant support and guidance. Her instant responses to my countless inquiries have been invaluable and motivational. It was a great opportunity to work under her supervision.

Many thanks to Dr. Inderveer Chana, Assistant Professor, Thapar University for her moral support and research environment she had facilitated for this work.

Finally I wish to thank my family and friends for their immense love and encouragement throughout my life without which it would not have been possible to complete this work.

Last but not the least I would like to thank God who have always been with me in my good and bad times.

Vishonika Kaushal
(800931024)

Abstract

Cloud computing, with its great potentials in low cost and on-demand services, is a promising computing platform for both commercial and non-commercial computation clients. In this thesis various fault tolerance techniques have been discussed to improve performance and availability of server applications.

Cloud computing is a relatively new way of referring to the use of shared computing resources, and it is an alternative to having local servers handle applications. The cloud computing end users usually have no idea where the servers are physically located. Cloud computing environments comprise of high level of communication and server failures in contrast to conventional data centers. Therefore new tools and techniques are required to build fast, reliable and autonomic fault tolerant system.

When several instances of an application running on several virtual machines, a problem that arises is implementation of an autonomic fault tolerance technique to handle a server failure to reassure system reliability and availability. If any of the servers break down, system should automatically redirect user requests to the backup server. The solution provided in this thesis make use of a software tool known as HAProxy that offers high availability and can handle server failures in the framework of virtual machines. This thesis has proposed the cloud virtualized system architecture by using HAProxy. The prototype system is implemented in Linux by using three web server applications that may comprise of faults. The approach provides autonomic and transparent fault tolerance capability to cloud applications. The experimental results show that HAProxy can make server applications to recover from these faults in a few milliseconds by increasing system availability and reliability.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives the introduction of the thesis work. It gives brief introduction of Cloud Computing, Fault Tolerance, and overview of work done.

1.1 Background

Cloud computing has evolved through a number of phases which include grid and utility computing, Software as a Service (SaaS) and cloud computing. The roots of cloud computing can be traced to the early 1990s and the concept of grid computing, through which many computing devices were networked together to work on a single problem, usually scientific in nature and requiring exceptionally high levels of parallel computation. Grid computing led to utility computing, which attempted to provide metered computing services as though they were a utility. Utility computing led to Software as a Service (SaaS), which allows users to access commercially available software online instead of using it locally, charging service fees instead of selling licensed applications [6].

Next, the term cloud computing originated that transforms once-expensive capital assets like disk storage and processing cycles into a readily available, affordable commodity. Figure1.1 shows the evolution of the Cloud.

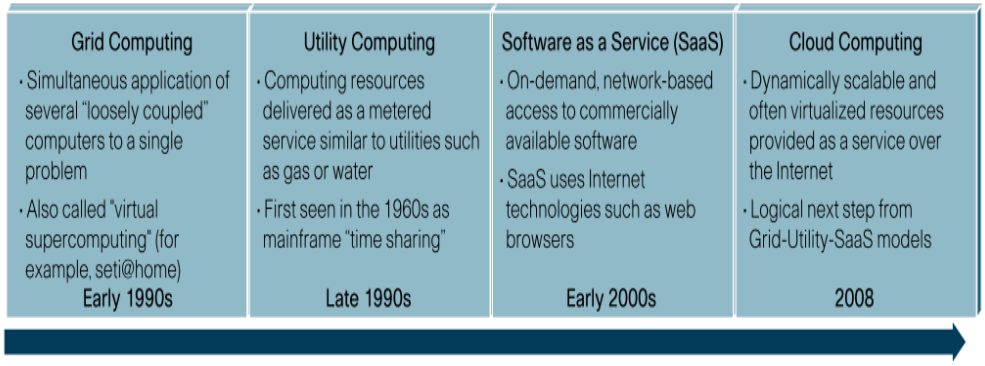

Figure 1.1 Evolution of Cloud Computing [6]

The term cloud computing is sometimes used to refer to a new paradigm; some authors even speak of a new technology that offers IT resources and services over the Internet. The technology analysts at Gartner see cloud computing as a so-called "emerging technology" [1] on its way to the hype. When looking at the number of

searches for the word pair “cloud computing” undertaken with the Google search engine one can get a feeling of the high interest on the topic.

1.2 Introduction to Cloud Computing

Cloud computing is an IT deployment model, based on virtualization, where resources, in terms of infrastructure, applications and data are deployed via the internet as a distributed service by one or several service providers. These services are scalable on demand and can be priced on a pay-per-use basis [2].

Cloud computing describes highly scalable computing resources provided as an external service via the internet. It can be using a storage cloud to manage application, business, and personal data. The cloud is simply a metaphor for the internet, based on the symbol used to represent the worldwide network in computer network diagrams [8]. Cloud computing is a recent trend in IT that moves computing and data away from desktop and portable PCs into large data centres. It refers to applications delivered as services over the Internet as well as to the actual cloud infrastructure — namely, the hardware and systems software in data centres that provide these services. Consumers purchase such services in the form of infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), or software-as-a-service (SaaS) and sell value added services (such as utility services) to users. Cloud computing is Internet-based computing, whereby shared resources, software and information are provided to computers and other devices on-demand, like the electricity grid. Cloud computing customers do not own the physical infrastructure, instead avoiding capital expenditure by renting usage from a third-party provider. They consume resources as a service and pay only for resources that they use [4].

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet. Cloud computing is attractive to business owners as it eliminates the requirement for users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand [5].

1.3 Introduction to Fault Tolerance

Fault Tolerance is one of the key issues of cloud computing. Cloud computing has been increasingly popular in the last era, and is being commonly used in various kinds of applications, many of which are not tolerant to system failures and results in loss of service. Fault tolerance is concerned with all the techniques necessary to enable a

system to tolerate software faults remaining in the system after its development. These software faults may or may not manifest themselves during systems operations, but when they do, software fault tolerant techniques should provide the necessary mechanisms of the software system to prevent system failure occurrences.

A system is fault tolerant, if service failure can be avoided when faults are present in the system. Fault tolerance makes a system capable to operate correctly in a faulty condition, protect against accidental or malicious destruction of information, protect against generating erroneous output and guarantees that confidential information cannot be divulged.

1.4 Organisation of Thesis

The chapters in this thesis are organized as follows –

Chapter 2 – This chapter describes in detail the literature survey, what is cloud computing, fault tolerance and implementing fault tolerance in Cloud Computing.

Chapter 3 – This chapter describes the problem statement of the thesis. It gives the gap analysis and requirement analysis.

Chapter 4 – This chapter describes the solution of problem, design of solution, system architecture, Tools Alternatives for fault tolerant system and core implementation of solution.

Chapter 5 – This chapter gives the Experimental results –Web application snapshots, database snapshots, HAProxy snapshots, Snapshots of HAProxy statistics.

Chapter 6 – This chapter describes the conclusion, contributions of work done and future research work possible.

Chapter 2

Literature Review

This chapter describes in detail the literature survey, what is cloud computing, fault tolerance and implementing fault tolerance in Cloud Computing.

2.1 Cloud Computing

Cloud computing is expected to be the platform for next generation computing, in which users carry thin clients such as smart phones while storing most of their data in the cloud and submitting computing tasks to the cloud. A web browser serves as the interface between clients and the cloud. Operating system in web browsers allows the users to manage their data and computation tasks [5].

One of the main drivers for the interest in cloud computing is cost and reliability. As personal computers and their OS and software are becoming more and more complex, the installation, configuration, update, and removal of such computer systems require a significant amount of intervention time from the users or system managers. Instead, outsourcing the computation tasks eliminates most of such concerns. The cloud of computer grids provides such services on fee-based, which can be more cost-efficient for the users than purchasing, maintaining, and upgrading powerful servers. Furthermore, resource sharing improves overall resource usage [5].

Overall, cloud computing brings the following three new aspects in computing resource management: infinite computing resources available on demand for the perspective of the end users; zero up-front commitment from the cloud users; and short-term usage of any high-end computing resources [3] [4].

Related Terms and Technologies

- **Grid Computing**

Grid computing is a form of distributed computing based on “a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities”.

- **Utility Computing**

Utility computing is a service provisioning model in which consumers use services on a pay-per-use basis. Utility computing is also similar to IaaS implementations of cloud computing. However, the main difference is that utility computing is simply a

"resources for rent" model as opposed to the much broader approach defined by cloud computing for designing, building, deploying, and running applications in the cloud.

- **On-Demand Computing**

On-Demand Computing is simply another term used to refer the on-demand characteristics of cloud computing and utility computing. Some industry players in this growing market are Google, HP, IBM, Rackable Systems, Sun, and Verari Systems.

2.1.1 What does it comprise?

Cloud computing can be visualised as a pyramid consisting of three sections:

- **Cloud Application**

This is the apex of the cloud pyramid, where applications are run and interacted with via a web browser, hosted desktop or remote client. A hallmark of commercial cloud computing applications is that users never need to purchase expensive software licenses themselves. Instead, the cost is incorporated into the subscription fee. A cloud application eliminates the need to install and run the application on the customer's own computer, thus removing the burden of software maintenance, ongoing operation and support.

- **Cloud Platform**

The cloud platform is a middle layer of the cloud pyramid, which provides a computing platform or framework as a service. A cloud computing platform dynamically provisions, configures, reconfigures and de-provisions servers as needed to cope with increases or decreases in demand. This in reality is a distributed computing model, where many services pull together to deliver an application or infrastructure request.

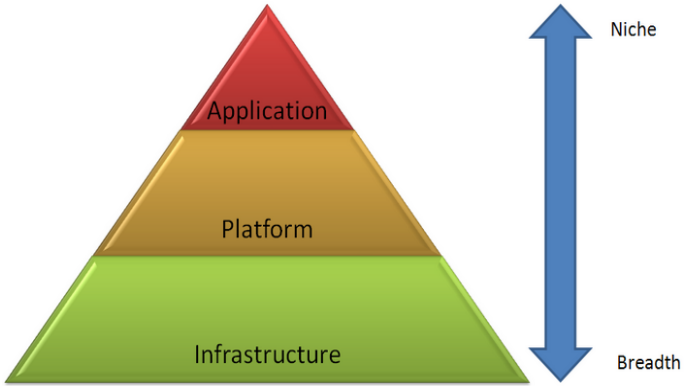

Figure 2.1 Cloud Pyramid [11]

- **Cloud Infrastructure**

The foundation of the cloud pyramid is the delivery of IT infrastructure through virtualisation. Virtualisation allows the splitting of a single physical piece of hardware into independent, self governed environments, which can be scaled in terms of CPU, RAM, Disk and other elements. The infrastructure includes servers, networks and other hardware appliances delivered as either Infrastructure “Web Services”, “farms” or "cloud centres". These are then interlinked with others for resilience and additional capacity.

2.1.2 Deployment Models of Cloud Computing

Clouds can be classified into following computing implementations:

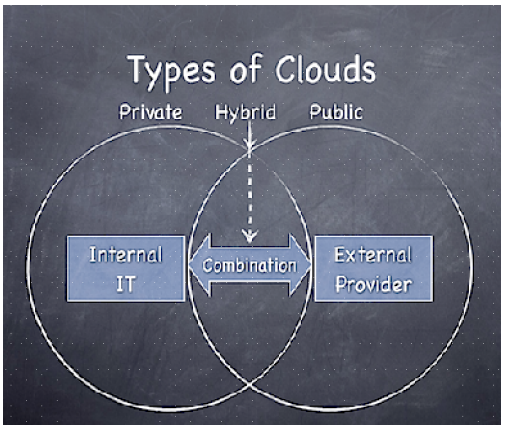

Figure 2.2 Public, Private, and Hybrid Cloud Deployment

- **Public clouds**

In public clouds, resources are offered as a service, usually over an internet connection, for a monthly or a pay-per-usage fee. Users can scale on-demand and do not need to purchase hardware. Cloud providers manage the infrastructure and pool resources into capacity required by consumers as shown in Figure2.3.

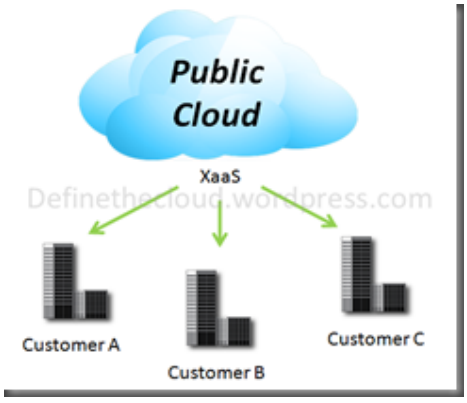

Figure 2.3 Public Cloud

- **Private clouds**

Private clouds are typically deployed inside a firewall and managed by the user organization. In this case, the user organization owns the software and hardware running in the cloud, manages the cloud, and provides virtualized cloud resources. These resources are typically not shared outside the organization, and full control is retained by the organization as shown in Figure2.4.

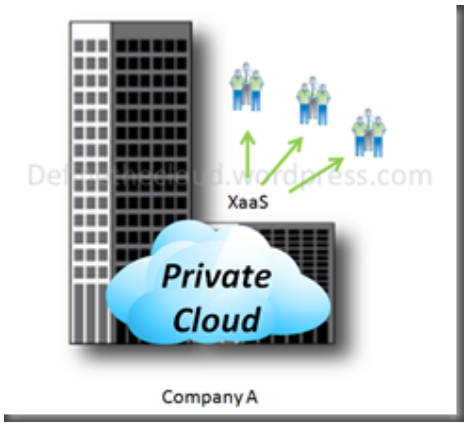

Figure 2.4 Private Cloud

- **Hybrid Cloud**

The cloud infrastructure consists of a number of clouds of any type, but the clouds have the ability through their interfaces to allow data and/or applications to be moved from one cloud to another. This can be a combination of private and public clouds that support the requirement to retain some data in an organization, as shown in Figure2.5 and also the need to offer services in the cloud.

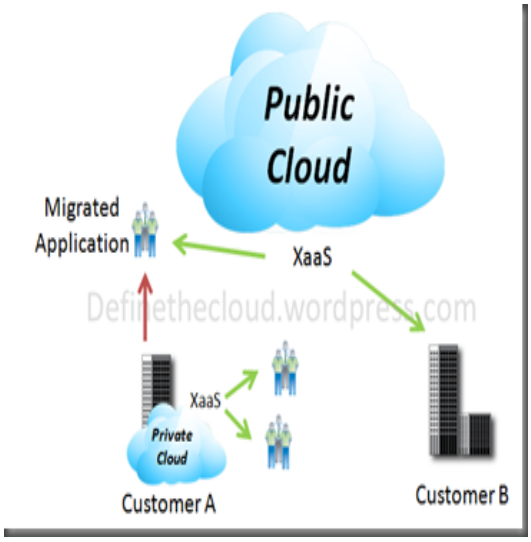

Figure 2.5 Hybrid Cloud

- **Community Cloud**

The cloud infrastructure is shared among a number of organizations with similar interests and requirements. This may help limit the capital expenditure costs for its establishment as the costs are shared among the organizations. The operation may be in-house or with a third party on the premises as shown in Figure2.6.

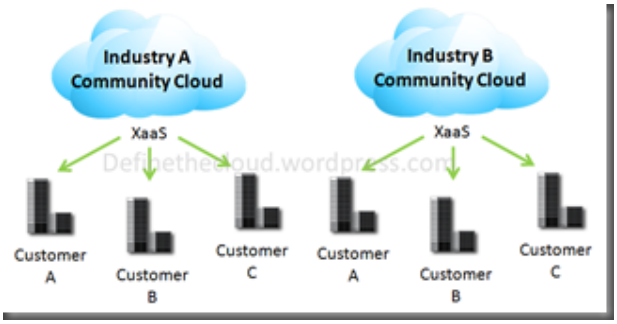

Figure 2.6 Community Cloud

2.1.3 Cloud Computing Service Models

Based on capabilities, there are three types of cloud computing implementations:

- **SaaS (Software as a Service)**

SaaS is the most widely known and widely used form of cloud computing. It provides all the functions of a sophisticated traditional application to many customers and often thousands of users, but through a Web browser, not a "locally-installed" application. Little or no code is running on the Users local computer and the applications are usually tailored to fulfil specific functions. SaaS eliminates customer worries about application servers, storage, application development and related, common concerns of IT. Highest-profile examples are Salesforce.com, Google's Gmail and Apps, instant messaging from AOL, Yahoo and Google, and VoIP from Vonage and Skype [8].

- **PaaS (Platform as a Service)**

PaaS delivers virtualized servers on which customers can run existing applications or develop new ones without having to worry about maintaining the operating systems, server hardware, load balancing or computing capacity. These vendors provide APIs or development platforms to create and run applications in the cloud like using internet. Managed Service providers with application services provided to IT departments to monitor systems and downstream applications such as virus scanning for e-mail are frequently included in this category. Well known providers would include Microsoft's Azure, Salesforce's Force.com, Google Maps, ADP Payroll processing, and US Postal Service offerings [8].

- **IaaS (Infrastructure as a Service)**

IaaS delivers utility computing capability, typically as raw virtual servers, on demand that customers configure and manage. Here cloud computing provides grids or clusters or virtualized servers, networks, storage and systems software, usually (but not always) in a multitenant architecture. IaaS is designed to augment or replace the functions of an entire data centre. This saves cost (time and expense) of capital equipment deployment but does not reduce cost of configuration, integration or management and these tasks must be performed remotely. Vendors would include Amazon.com (Elastic Compute Cloud [EC2] and Simple Storage), IBM and other traditional IT vendors [8].

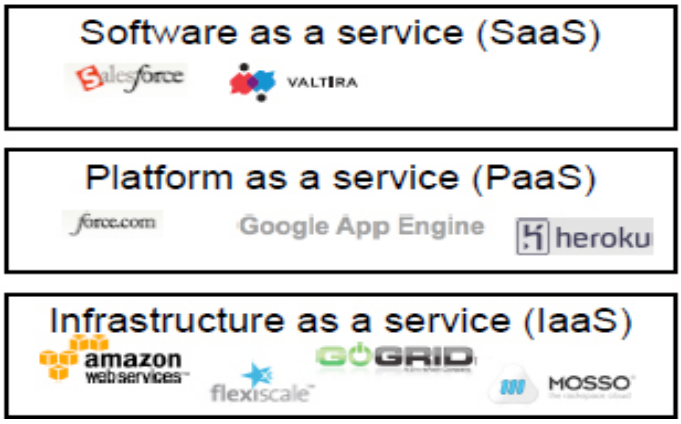

Figure 2.7 The 3 layers of Cloud Computing: SaaS, PaaS, and IaaS [7]

2.1.4 Layered Architecture Of Cloud

Clouds can be built on top of many existing protocols such as Web Services (WSDL, SOAP), and some advanced Web 2.0 technologies such as REST, RSS, AJAX, etc. In fact, behind the cover, it is possible for Clouds to be implemented over existing Grid technologies leveraging more than a decade of community efforts in standardization, security, resource management, and virtualization support [9]. Figure 2.8 shows the layered design of service-oriented Cloud computing architecture. Physical Cloud resources along with core middleware capabilities form the basis for delivering IaaS. The user-level middleware aims at providing PaaS capabilities. The top layer focuses on application services (SaaS) by making use of services provided by the lower layer services. PaaS/SaaS services are often developed and provided by 3rd party service providers, who are different from IaaS providers [10].

- **User-Level Middleware**

This layer includes the software frameworks such as Web 2.0 Interfaces (Ajax, IBM Workplace) that help developers in creating rich, cost effecting User-interfaces for browser-based applications .The layer also provide the programming environments and composition tools that ease the creation, deployment, and execution of applications in Clouds.

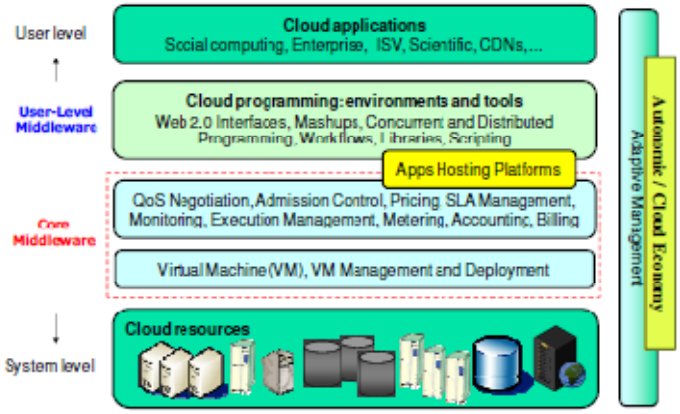

Figure 2.8 Layered Cloud Computing Architecture [15]

- **Core Middleware**

This layer implements the platform level services that provide runtime environment enabling Cloud computing capabilities to application services built using User-Level Middleware. Core services at this layer includes Dynamic SLA Management, Accounting, Billing, Execution monitoring and management, and Pricing. The well-known examples of services operating at this layer are Amazon EC2, Google App Engine, and Aneka [12].

- **System Level**

The computing power in Cloud computing environments is supplied by a collection of data centres, which are typically installed with hundreds to thousands of servers [14]. At the System Level layer there exist massive physical resources (storage servers and application servers) that power the data centres. These servers are transparently managed by the higher level virtualization [13] services and toolkits that allow sharing of their capacity among virtual instances of servers. These VMs are isolated from each other, which aid in achieving fault tolerant behaviour and isolated security context.

2.1.5 Active platforms

Many industrial companies presented their understanding of Cloud computing and also successfully applied it with various technologies. Here are some specific Cloud industry instances and research units.

- **Amazon Elastic Compute Cloud (EC2**)

Amazon Elastic Compute Cloud (EC2) [37] provides a virtual computing environment that enables a user to run Linux-based applications. The user can either create a new Amazon Machine Image (AMI) containing the applications, libraries, data and associated configuration settings, or select from a library of globally available AMIs. The user then needs to upload the created or selected AMIs to Amazon Simple Storage Service (S3), before he can start, stop, and monitor instances of the uploaded AMIs. Amazon EC2 charges the user for the time when the instance is alive, while Amazon S3 [38] charges for any data transfer

- **Google App Engine**

Google App Engine [39] is a platform for building and hosting web applications on infrastructure operated by Google. It allows a user to run web applications written using the Python programming language. Other than supporting the Python standard library, Google App Engine also supports Application Programming Interfaces (APIs) for the data store, Google Accounts, URL fetch, image manipulation, and email services. Google App Engine also provides a web-based Administration Console for the user to easily manage his running web applications. Currently, Google App Engine is free to use with up to 500MB of storage and about 5 million page views per month.

- **Microsoft Azure**

Microsoft Azure [40] aims to provide an integrated development, hosting, and control Cloud computing environment so that software developers can easily create, host, manage, and scale both Web and non-web applications through Microsoft data centers. To achieve this aim, Microsoft Azure supports a comprehensive collection of proprietary development tools and protocols which consists of Live Services, Microsoft .NET Services, Microsoft SQL Services, Microsoft SharePoint Services, and Microsoft Dynamics CRM Services. Microsoft Azure also supports Web APIs such as SOAP and REST to allow software developers to interface between Microsoft or non-Microsoft tools and technologies.

- **Sun network.com (Sun Grid)**

Sun network.com (Sun Grid) [41] enables the user to run Solaris OS, Java, C, C++, and FORTRAN based applications. First, the user has to build and debug his applications and runtime scripts in a local development environment that is configured to be similar to that on the Sun Grid. Then, he needs to create a bundled zip archive (containing all the related scripts, libraries, executable binaries and input data) and upload it to Sun Grid. Finally, he can execute and monitor the application using the Sun Grid web portal or API. After the completion of the application, the user will need to download the execution results to his local development environment for viewing.

- **Rackspace Cloud – (cloud servers, cloud files)**

Rackspace Cloud [16] is the cloud hosting division of Rackspace, an industry leader that currently manages over 40,000 servers and devices for customers all over the world. Rackspace Cloud Server is a compute service that provides server capacity in the cloud. Cloud Servers come in different flavours of memory, disk space, and CPU, and can be provisioned in minutes. There are no contracts or commitments. Interactions with Rackspace Cloud Servers can occur via the Rackspace Cloud Control Panel (GUI) or programmatically via the Rackspace Cloud Servers API. Rackspace Cloud Files is an affordable, redundant, scalable, and dynamic storage service offering. The core storage system is designed to provide a safe, secure, automatically re-sizing and network-accessible way to store data. Cloud Files provides a simple yet powerful way to publish and distribute content behind the industry-leading Akamai Content Distribution Network. Cloud Files users get access to this network automatically without having to worry about contracts, additional costs, or technical hurdles. Cloud Files allows users to store/retrieve files and CDN-enable content via a simple ReST (Representational State Transfer) web service interface. There are also language-specific APIs that utilize the ReST API but make it much easier for developers to integrate into their applications.

2.1.6 Benefits of using Cloud

Despite its possible security and privacy risks, Cloud Computing has following benefits that organizations are certain to want to take advantage of. In very brief table form they are as follows [6]:

Table 2.1 Benefits of using Cloud

Benefit	Comment
Cost Savings	Organizations can reduce or eliminate capital expenditures and decrease ongoing operating expenditures by paying only for the services they use and, potentially, by reducing or redeploying their staffs.
Ease of Implementation	Without the need to purchase hardware, software licenses, or implementation services, an organization can deploy cloud computing rapidly.
Flexibility	Cloud computing offers more flexibility (often called "elasticity") in matching resources to business functions than past computing methods. It can also increase staff mobility by enabling access to business information and applications from a wider range of locations and/or devices.
Scalability	Organizations using cloud computing need not scramble to secure additional, higher-caliber hardware and software when user loads increase, but can instead add and subtract capacity as the network load dictates.
Sustainability Sustainability	The poor energy efficiency of most data centers, due to substandard design or inefficient asset utilization, is now understood to be environmentally and economically unsustainable. Cloud service providers, by using economies of scale and their capacity to manage computing assets more efficiently, can consume far less energy and other resources than traditional data center operators.
Focusing on Core Competencies	Arguably, the ability to run data centers and to develop and manage software applications is not necessarily a core competency of most organizations. Cloud computing can make it much easier to reduce or shed these functions, allowing organizations to concentrate on critical issues such as (in government) the development of policy and the design and delivery of public services.

Redeployment of IT Staff	By reducing or eliminating constant server updates and other computing issues, and by cutting expenditures of time and money on application development, organizations can focus on higher-value tasks.
Access to Top-End IT Capabilities	Particularly for smaller organizations, cloud computing can allow access to higher-caliber hardware, software, and staff than they can attract and/or afford themselves.
Agility	Cloud computing makes users can rapidly and inexpensively re-provision technological infrastructure resources.

2.2 Fault Tolerance

Fault tolerance, reliability, and availability are becoming major design issues nowadays in massively parallel distributed computing systems. Examples of systems in which fault tolerance is needed include mission-critical, computation-intensive, transactions (such as banking), and mobile/wireless computing systems/networks. High performance, measured in terms of speed and computing power, is essentially used as major design objective for such systems. It is however conceivable that great loss of crucial transactions can take place due to a small system/component error. The emergence of new paradigms, such as mobile/wireless computing, requires the introduction of new techniques for fault tolerance. It is therefore prudent that the issue of fault tolerance become among the set of design objectives of current and future computing systems [17]. Fault tolerance makes a system capable to operate correctly in a faulty condition, protect against accidental or malicious destruction of information, protect against generating erroneous output and guarantees that confidential information cannot be divulged.

Fault Tolerance is one of the key issues of cloud computing. Fault tolerance is concerned with all the techniques necessary to enable a system to tolerate software faults remaining in the system after its development. These software faults may or may not manifest themselves during systems operations, but when they do, software fault tolerant techniques should provide the necessary mechanisms of the software system to prevent system failure occurrences. With increasing heterogeneity and complexity of computational clouds, and due to the inherent unreliable nature of large-scale cloud infrastructure, fault tolerance techniques have become a major

concern. Fault tolerance techniques are designed to allow a system to tolerate software faults that remain in the system after its development. Fault tolerance techniques are employed during the procurement, or development, of the software. When a fault occurs, these techniques provide mechanisms to the software system to prevent system failure from occurring [18]. It should also be incorporated in autonomic cloud computing environment.

2.2.1 Concepts and Terminologies

- **Dependability**

Being fault tolerant is strongly related to what are called dependable systems. Dependability is a term that covers a number of useful requirements for distributed systems including the following:

- **Availability**

Availability is defined as the property that a system is ready to be used immediately. In general, it refers to the probability that the system is operating correctly at any given moment and is available to perform its functions on behalf of its users.

- **Reliability**

Reliability refers to the property that a system can run continuously without failure. A highly reliable system is one that will most likely continue to work without interruption during a relatively long period of time.

- **Safety**

Safety refers to the situation that when a system temporarily fails to operate correctly, nothing catastrophic happens.

- **Maintainability**

Maintainability refers to how easy a failed system can be repaired. A highly maintainable system may also show a high degree of availability, especially if failures can be detected and repaired automatically.

- **Fault**

Fault can be defined as incorrect state of hardware or software resulting from physical defects, design flaw or operator error.

Faults can be classified into one of three categories:

- Transient faults
- Intermittent faults
- Permanent faults

Transient faults occur once and then disappear. For example, a network message transmission times out but works fine when attempted a second time. Intermittent faults are the most annoying of component faults. This fault is characterized by a fault occurring, then vanishing again, then occurring. An example of this kind of fault is a loose connection. Permanent faults are persistent: it continues to exist until the faulty component is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out hardware.

- **Error**

Error is a part of a system state that may lead to a failure or it can be the manifestation of a fault.

- **Failure**

When a system or module is designed, its behaviour is specified. When observed behaviour differs from the specified behaviour it is called as failure.

Failures can be classified as:

- Crash failure
- Transient failure
- Byzantine failure
- Temporal failure

A process undergoes crash failure, when it permanently ceases to execute its actions. This is an irreversible change. The agent inducing transient failure may be temporarily active, but it can make a lasting effect on the global state. When a process behaves arbitrarily, it means that Byzantine failure has occurred. Real time systems require actions to be completed within a specific amount of time. When the deadline is not met, a temporal failure occurs.

2.2.2 Fault Tolerance Policies

Fault tolerance (FT) policies can typically be listed into two sets: reactive fault tolerance policies and proactive fault tolerance policies. While reactive fault tolerance policies reduces the effect of failures on application execution when the failure effectively occurs; proactive fault tolerance policies keeps applications alive by avoiding failures through preventative measures. The principle of proactive action is to avoid recovery from faults, errors and failures by predicting them and proactively replace the suspected components by other correctly working components providing the same function.

Table 2.2 summarizes the difference between reactive and proactive fault tolerance [24].

Table 2.2 Reactive Vs Proactive Fault Tolerance

Reactive Fault Tolerance	Proactive Fault Tolerance
The principle of reactive action is to minimize the impact of failures on application execution when the failure effectively occurs.	The principle of proactive action is to avoid recovery from faults, errors and failures by predicting them and proactively replace the suspected components by other correctly working components providing the same function.
Reactive Fault Tolerance keeps parallel applications alive through recovery from experienced failures.	Proactive FT keeps applications alive by avoiding failures through preventative measures.
Employed mechanisms in reactive FT react to failures.	Employed mechanisms in proactive FT anticipate failures.
The rate of failures in larger systems is bound to increase, that cause reactive schemes to result in increased I/O bandwidth requirements, which can become a bottleneck.[25]	The rate of failures in larger systems is bound to increase, but proactive FT supports larger systems.[25]
Examples: Checkpoint/Restart, Replay and Retry.	Example: Preemptive migration, Software Rejuvenation.

2.2.3 Fault Tolerance Techniques

To enhance fault-tolerance, in cloud computing environment various techniques are being introduced. The brief summary of techniques is as follows:

- **Checkpointing/Restart**

In checkpointing when a task fails, it is allowed to be restarted from the recently checkpointed state rather than from the beginning. Checkpoints can be created at fixed intervals or at particular points during the computation determined by some optimizing rule. It is an efficient fault tolerance technique for long running applications [19].

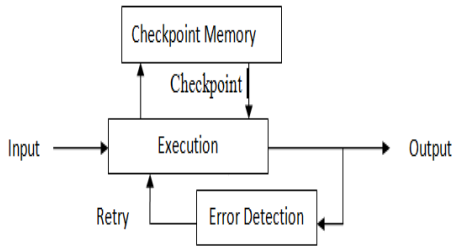

Figure 2.9 Logical representation of checkpoint/ restart [19]

It has an advantage of being independent of the damage caused by a fault. It is mostly applicable to unanticipated faults. It can be used at multiple levels in a system, and conceptually simple to implement. It provides ability to save and restore state for critical applications. One of the limitations of this technique is that it is time costly.

Local checkpoints

A process may take a local checkpoint any time during the execution. The local checkpoints of different processes are not coordinated to form a global consistent checkpoint [20].

Forced checkpoints

Each process examines the information and occasionally is forced to take a checkpoint according to the protocol.

Useless checkpoints

A useless checkpoint of a process is one that will never be part of a global consistent state [21]. Useless checkpoints are not desirable because they do not contribute to the recovery of the system from failures, but they consume resources and cause performance overhead Checkpoint intervals: A checkpoint interval is the sequence of events between two consecutive checkpoints in the execution of a process.

- **Replication**

The basic idea behind replication is to have replicas of a task run on different resources, so that as long as not all replicated tasks crash, the task execution would succeed. Software replication focuses mostly on processor or network faults, and does not address the whole spectrum of possible faults (design, timing). Replication algorithms are simpler to implement [19].

It has the advantage that it is less intrusive with respect to execution time. It provides the ground work for the shortest recovery delays. It scales much better. On the design perspective, software replication is relatively generic and transparent to the

application domain. The limitation can be the failover time, the time it takes to elect a new primary in case of failure, may be unacceptably high.

Replication in Distributed Parallel Processing

Existing approaches to managing intermediate data are at two ends of a spectrum – they are either store-local or DFS. Store-local means storing intermediate data locally at the outputting node and having it read remotely. DFS – uses a distributed file system that replicates intermediate data [20]. Intermediate data are short-lived distributed data for parallel dataflow programs that are critical for completion of the job and for good run-time performance. One of the disadvantages of replicating enormous amount of intermediate data can be network interference and increased job completion time.

Some more effective replication policies for reducing network interference and job completion time are discussed further.

Replication Using Spare Bandwidth [21]

It is a replication mechanism that leverages spare bandwidth and yet achieves higher and more controllable network utilization [21]. The approach currently employed here is a master/slave architecture, where the master is essentially a controller that coordinates the actions of slaves. The master periodically receives feedback from slaves regarding its current data transfer rate. Then, it calculates the spare bandwidth and notifies the slaves of the values [21]. In particular, this approach allows effective use of spare bandwidth.

Deadline-Based Replication [21]

While the above approach should effectively utilize the spare bandwidth, there is uncertainty about when data is finally replication [21]. A stage-based replication deadline where, a replication deadline of N means, that intermediate data generated during a stage has to complete replication within the next N stages. This deadline-based approach can reduce the cost of cascaded re-execution [20] because it bounds the maximum number of stages that any re-execution can cascade over (N) [21].

Cost Model Replication:

Cost model replication means dynamically deciding whether to replicate intermediate data or not, by comparing the cost of replication to the cost of cascaded re-execution. The thumb-rule is to minimize the cost – at the end of each stage, if the cost of replication is cheaper than the cost of cascaded re-execution, replication is performed.

Asynchronous Replication

Asynchronous replication basically is used in dataflow programming framework such as HDFS, DFS. A dataflow programming framework typically runs in a datacenter-style environment, where machines are organized in racks and connected with a hierarchical topology. Intermediate data generated by dataflow programming frameworks is written once by a single writer, and read once by a single reader [20]. Asynchronous replication, allows writers to proceed without waiting for replication to complete. In comparison, HDFS uses synchronous replication, where writers are blocked until replication finishes. Asynchronous replication cannot provide the same level of consistency guarantee as synchronous replication [20].

Rack-Level Replication

Typically, a dataflow programming framework runs in a datacenter-style environment, where machines are organized in racks, and the network topology is hierarchical, e.g., 2 levels with top-of-the rack switches and a core switch [21]. Based on this observation, a rack-level replication is employed, where replicas of an intermediate data block are always placed among the machines in the same rack. The advantage of this approach is that it lowers data transfer through the bandwidth-scarce core switch. This reduces the chance of interfering with foreground dataflow computations. The disadvantage is that it cannot tolerate rack failures, because if a whole rack fails, all replicas get lost.

- **Software Rejuvenation**

Software rejuvenation is a technique of proactive fault tolerance that designs the system for periodic reboots. Micro- rejuvenation is a technique of small reboots done frequently to extend the time without a failure.

Software Rejuvenation is an act of preventing unexpected error termination by terminating the program before it suffers an error. It restarts the system with clean state. Software rejuvenation refers to the rejuvenation of the environment in which the software is executing. A software system is known to suffer from outages due to transient errors and the state of software system degrades as time goes [22].

- **Proactive Fault Tolerance using Pre-emptive Migration**

Pre-emptive Migration relies on a feedback-loop control mechanism. Application health is constantly monitored and analyzed. It is reallocated to improve its health and avoid failures. This technique is a closed- loop control similar to dynamic load

balancing. There is no 100% coverage of failures that can be anticipated, such as random bit flips [23].

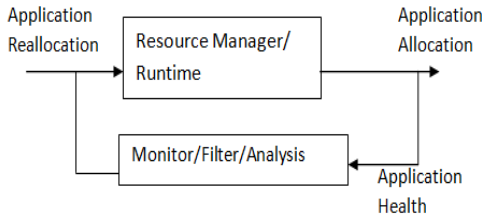

Figure 2.10 Feedback-loop control of Proactive Fault tolerance [23]

- **Proactive Recovery**

A Byzantine-faulty replica may appear to behave properly even when broken; therefore recovery must be proactive to prevent an attacker from compromising the service by corrupting 1/3 of the replicas without being detected. This algorithm recovers replicas periodically independent of any failure detection mechanism. However a recovering replica may not be faulty and recovery must not cause it to become faulty, since otherwise the number of faulty replicas could exceed the bound required to provide safety. In fact, there is a need to allow the replica to continue participating in the request processing protocol while it is recovering, since this is sometimes required for it to complete the recovery.

2.3 Implementing Fault Tolerance in Cloud Computing

Although many mature technologies from other domains can be used as components in cloud computing, but there are still many unresolved and open problems due to its unique characteristics which are different from distributed computing, cluster computing, grid computing, utility computing and service computing. Fault-tolerant network systems are designed to provide reliable and continuous services for wired and wireless distributed networks despite the failures of some of their components. Although there are many algorithms available in other domains that cannot be directly applied to cloud computing. Most existing schemes cannot automatically adjust their parameters for dynamic conditions which are common in cloud computing environments [42].

Current clouds infrastructures do not provide the full potential of autonomic fault tolerant services. Modern cloud infrastructures live in an open world, characterized by continuous changes in the environment and in the requirements they have to meet. Continuous changes occur autonomously and unpredictably, and they are out of

control of the cloud provider. Therefore, advanced solutions have to be developed able to dynamically adapt the cloud infrastructure, while providing continuous service and performance guarantees. A number of autonomic computing solutions have been developed such that resources are dynamically allocated among running applications [26].

2.3.1 Benefits of Implementing fault tolerance in Cloud Computing

The main benefits of implementing fault tolerance in cloud computing include failure recovery, lower cost, improved performance metrics.

- **Failure Recovery**

Implementing fault tolerance in cloud would prompt response and recovery to any anomalous event that could be triggered due to misconfiguration, fault, performance problems, and cyber attack or exploitation.

- **Performance Improvement**

The most common problem that is seen in cloud virtualized environment is server breakdown that degrades system availability. Fault tolerance will lead to reduce time to deploy and configure new resources and services, reduce cost to failures, reduce cost by eliminating manual intensive activities, and improve performance metrics such as availability and reliability.

- **Cost Savings**

Fault tolerant services will lead to cost reduction by minimizing the number of failures and managing runtime faults. These may also reduce number of monitoring and management tools required to manage performance, fault, security and configuration.

2.3.2 Challenges of Implementing FT in Cloud

Providing fault tolerance requires careful consideration and analysis not only because of their complexity and inter-dependability but also for the following reasons.

- Fault tolerance should take into consideration the heterogeneous and complex nature of cloud environment. Providing autonomic fault tolerance for a virtualized infrastructure requires: understanding the relative position of the component within the physical infrastructure, the user and infrastructure properties, the management of the components affecting others, and the dependencies between the components.
- Cloud infrastructure consists of different types of hardware/software technologies, which are provided by multiple, and most likely competing vendors [27]. The fault

tolerance implementation requires complex communications at different stages across various cloud entities. This in turn means different technologies from competing vendors' needs to be integrated for establishing a reliable system. Providing such services using components from the same vendor are very complex to setup, error prone, and raise fault tolerant challenges in comparison with traditional systems [27].

- Cloud infrastructure is not hosted at a single data center that is located at a specific location; it is rather the opposite, as most likely it is distributed across distant data centers [27]. This factor has a major impact on decisions being made by fault tolerance mechanisms for several reasons; for example, the distance and the communication medium between distant data centers will have an impact on data transfer speed. Autonomic fault tolerance must react to these factors for improved synchronization.
- Cloud-of-cloud is a term that is used to refer to the collaboration of multiple cloud providers to support dependable cloud infrastructures [27]; i.e. cloud providers collaborate to help each other in enhancing mechanisms as in the case of higher reliability, availability, resilience and dependability.

Chapter 3

Problem Statement

This chapter discusses about the gap analysis, requirement analysis and need for Autonomic FT in Cloud Environment.

3.1 Gap Analysis

There exist a large class of applications which could benefit immensely with support for autonomic properties and behaviour. For example, many web server applications have irregular and highly variable resource requirements which are very difficult to predict in advance. As a consequence of irregular execution characteristics, dynamic resource requirements are not only difficult to predict, but are themselves complex, and complex to manage, being prone to faults in hardware, software specification and software implementation [28].

However, only performance and energy trade-off have been considered so far with a lower emphasis on the system dependability/availability which has been demonstrated to be the weakest link in the chain for early cloud providers. Therefore, an autonomic fault tolerant system should be implemented to fill this literature gap devising a backup server allocation during run time in cloud virtualized environment, to make system capable of identifying system performance and energy trade-offs. Thus providing an application availability guarantees for cloud end-users.

Another problem in cloud environment exits is the increasing criticality of data; all enterprises should have a disaster recovery plan in place for key applications. But many do not have a recovery plan, that primarily due to cost and complexity issues. Replication is an interesting alternative that addresses both of these issues. Data replication is highly adaptive to failures of any kind so as to maintain high data availability. A potential solution could be to maintain eventual data consistency among replicas. Upon replication, a second node contains the replicated data. Replication and cloud computing can also be considered as an alternative to local backup.

Gaps

- Cloud service architectures and vendors that provide Fault tolerance in virtualized environment are still immature and are best suited for small web applications.

- There is a need to implement autonomic fault tolerance by using different performance parameters in cloud environment.
- Autonomic fault tolerance for real time applications in cloud are not as efficient and reliable.
- For the cloud provider, data consistency should allow better availability, lower latency, and other benefits.

3.2 Requirement Analysis

Cloud Computing requirements for Autonomic Fault Tolerance

When several instances of an application running on several virtual machines, a problem that arises is how to employ an autonomic fault tolerance to handle a server failure to reassure system reliability and availability. If any of the servers break down, system should automatically redirect user requests to the backup server. The application availability can be maintained and any change made in the database of backup server is replicated to the failed server.

Table 3.1 shows the experimental platforms used in cloud virtualized system for implementing fault tolerance and data replication among servers.

Table 3.1 Platform Configurations

Type	Specifications
Processor	Intel(R) Core(TM)2 Duo
CPU Speed	2.20GHz
Memory	4.00 GB
Platform	32-bit Operating System
Operating System	Ubuntu 10.04

Table 3.2 shows the software packages used in cloud virtualized system for implementing fault tolerance and data replication among servers.

Table 3.2 Software Versions

Software Packages	Versions
VMware Workstation	6.5
Apache Tomcat	6.0.32
JDK	1.6.0_20
HAProxy	1.3.15.2
XAMPP for Linux	1.7.4
MySQL GUI	SQLyog

Chapter 4

Solution of Problem

4.1 Design of Solution

Following section gives overall guidance on the implementation of fault tolerance using HAProxy on cloud environment. Implementation includes a Virtual Machine server(VMware) and Linux operating system that is evaluated with two virtual machines (VMs) hosting as web servers, named as server 1 and server 2, both running the same application instances. Apache Tomcat 6.0.32 is configured on server 1 and server 2. HAProxy software version 1.3.15.2 is configured on third virtual machine.

A simple database application written in Java is installed on the web servers. Xampp for Linux Windows XP (SP2) is used to install MySQL. Data in MySQL is replicated using replication technique for local backup. Data consistency is also maintained through MySQL replication.

4.1.1 System Architecture

The design of the cloud virtualized system architecture using HAProxy is proposed in this thesis. Figure 4.1 shows the cloud virtualized system architecture. The server virtualized system consists of hosting server installing VMware and two hosted VMs (server 1 and server 2) on which an Ubuntu 10.04 OS and database application are running.

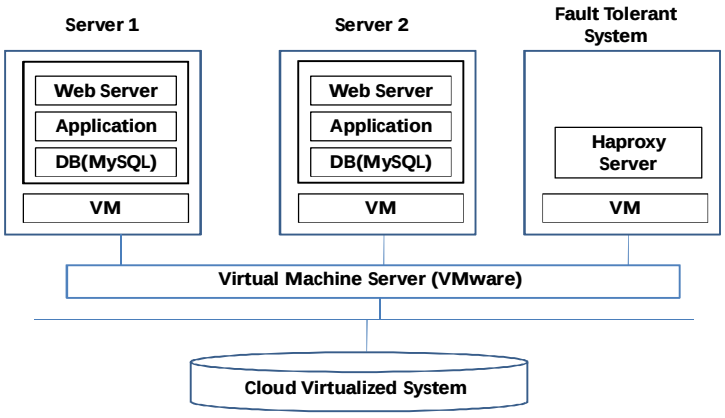

Figure 4.1 Cloud Virtualized System Architecture

Server 2 is a backup sever in case of failure. Third virtual machine with HAProxy configured on it, is used for fault handling. The availability of the servers is continuously monitored by HAProxy Statistics tool on a fault tolerant server. HAProxy is running on web server to handle requests from web. It is configured to listen to port 80#. When one of the servers goes down unexpectedly, connection will automatically be redirected to the other server. Figure 4.2 shows an application flow starts from sending request to web server, then the request is forwarded to HAProxy server. After receiving request HAProxy forwards that request to one of the application servers.

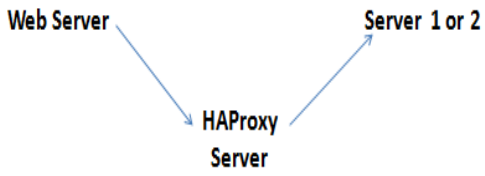

Figure 4.2 System Architecture

The end result is an automated solution which fails over to secondary servers and keeps administrators notified about the health of the servers.

4.1.2 VMware Workstation

VMware workstation is desktop software that allows running multiple x86-compatible desktop and server operating systems simultaneously on a single PC, in fully networked, portable virtual machines with no rebooting or hard drive partitioning required [43].

Product Benefits:

Workstation is used in the software development, quality assurance, training, sales, and IT fields. Easy to develop and test multiple operating systems and applications on a single PC. It is useful for testing multitier configurations. Workstation streamlines software development and testing. It also facilitates testing using multiple snapshots and debugging support. Workstation enhances productivity of IT professionals by configuring and testing desktops and servers as virtual machines before deploying them to production.

Host System Requirements

The virtual machines running under Workstation perform better if they have faster processors and more memory. The two terms that are used in here are host and guest.

Host is the physical computer on which Workstation software is installed. It is also known as host computer and its operating system is called host operating system. Guest is the operating system running inside a virtual machine.

Host system requirements include standard x86-compatible or x86-64-compatible for PC hardware. Compatible processors include Intel X and AMD. Multiprocessor systems are also supported. The minimum memory required for each guest OS and for applications on the host and guest is 512MB (2GB is recommended). The maximum amount of memory for each virtual machine is 8GB. 16-bit and 32-bit display adapter is recommended for the users. Guest OS can reside on physical disk partitions or in virtual disk files. VMware recommends at least free disk space for each guest OS and the application software used with it. It also supports ISO disk images. Any Ethernet controller that supports host OS can be used for local area networking. Figure 4.3 shows the VMware Workstation GUI.

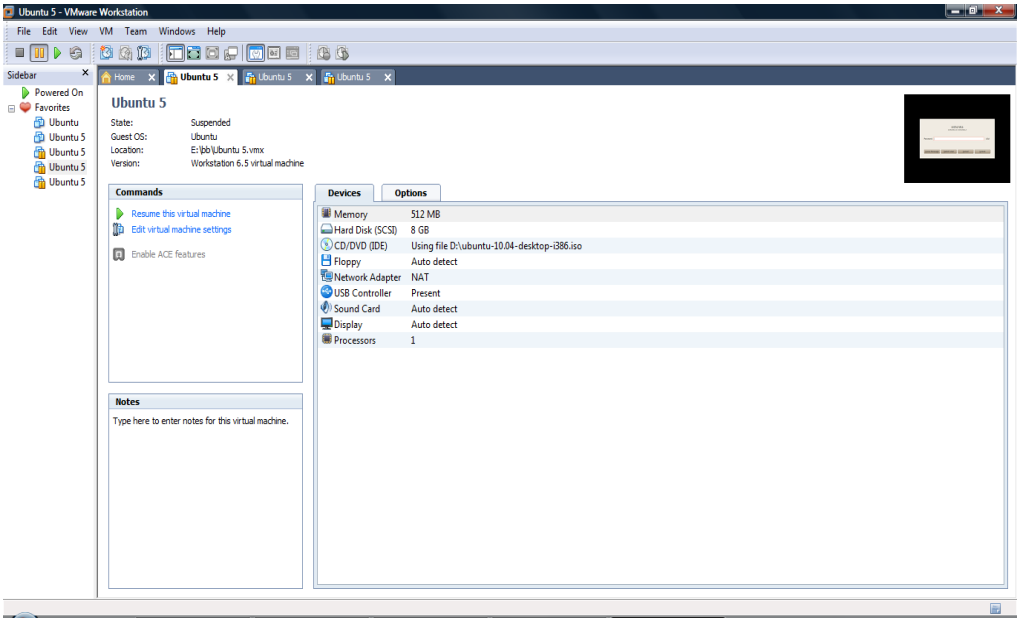

Figure 4.3 VMware Workstation

Virtual Machine Specifications

Virtual machine specifications include processor specifications that are same as that on host computer. Chip set supported for virtual machine includes Intel 440BX-based motherboard, NS338 SIO and 82093AA IOAPIC. VGA and SVGA graphics are supported.

4.1.3 Ubuntu

Ubuntu is a computer operating system based on the Debian GNU/Linux distribution and distributed as free and open source software. It is named after the Southern African philosophy of Ubuntu ("humanity towards others"). Ubuntu is designed primarily for desktop use although netbook and server editions exist as well. Ubuntu software center gives instant access to thousands of applications that are needed to customize the computer.

In this thesis Ubuntu 10.04 has been installed on all the three virtual machines. The ISO image of Ubuntu-10.04-desktop-i386 is downloaded and installed on the three VMs. It is fast, secure, easy to install and easy to use operating system.

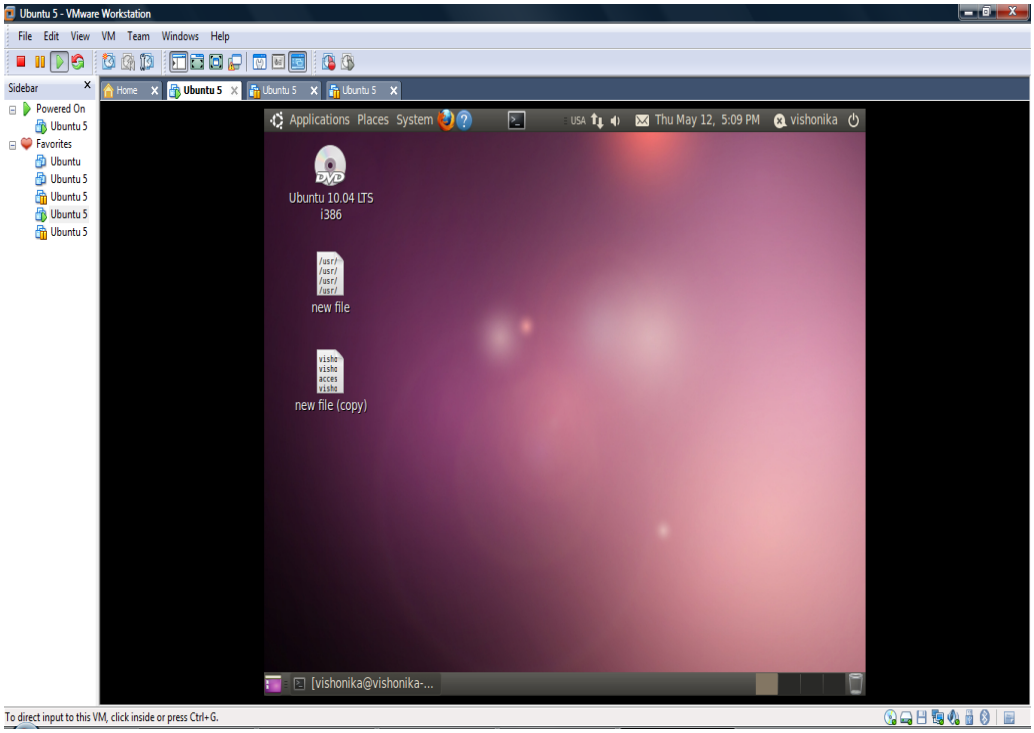

Figure 4.4 Ubuntu OS on VMware

System Requirements

Ubuntu 10.04 LTS Server Edition supports two major architectures: Intel x86 and AMD64 [33]. The minimum memory requirement for Ubuntu 10.04 LTS is 256 MB of memory. The table below lists recommended hardware specifications.

Table 4.1 Ubuntu Desktop Edition Requirements

Install Type	Processor	RAM	Hard Disk Space	Graphics Card and Monitor
Desktop	1 GHz x86	1 GB	8 GB	1024 by 768

4.1.4 Apache Tomcat

Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2. Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations. Apache Tomcat, Tomcat, Apache, the Apache feather, and the Apache Tomcat project logo are trademarks of the Apache Software Foundation [34].

In this thesis Apache Tomcat 6.0.32 has been downloaded from apache mirrors and installed on server 1 and server 2 for enabling access to web application.

4.1.5 Xampp

XAMPP is a small and light Apache distribution containing the most common web development technologies in a single package. Its contents, small size, and portability make it the ideal tool for students developing and testing applications in PHP and MySQL. XAMPP is available as a free download in two specific packages: full and lite. While the full package download provides a wide array of development tools-Apache HTTP Server, PHP, MySQL, phpMyAdmin, Openssl, and SQLite [35]. In this thesis MySQL tool has been used to create database for server application on server 1 and server 2.

4.1.6 MySQL Replication

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous-slaves need not be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dialup service [36]. In this thesis master IP address is 192.168.81.158 and slave IP address is 192.168.81.155.

The target uses for replication in MySQL include:

- Scale-out solutions – spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.

• Data security- because data is replicated to the slave, it is possible to run backup services on the slave without corrupting the corresponding master data.

• Analytics-live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.

4.1.7 SQLyog

SQLyog is an easy to use, compact and very fast graphical tool to manage MySQL database anywhere in the world. SQLyog is a tool that allows managing MySQL database. SQLyog works on the Windows platform starting from Windows XP to Windows 7. In this thesis, server 1 and server 2 databases are managed using SQLyog tool. SQLyog will connect to MySQL servers (server 1 and server 2) running on Ubuntu OS. Figure 4.5 shows the SQLyog Interface.

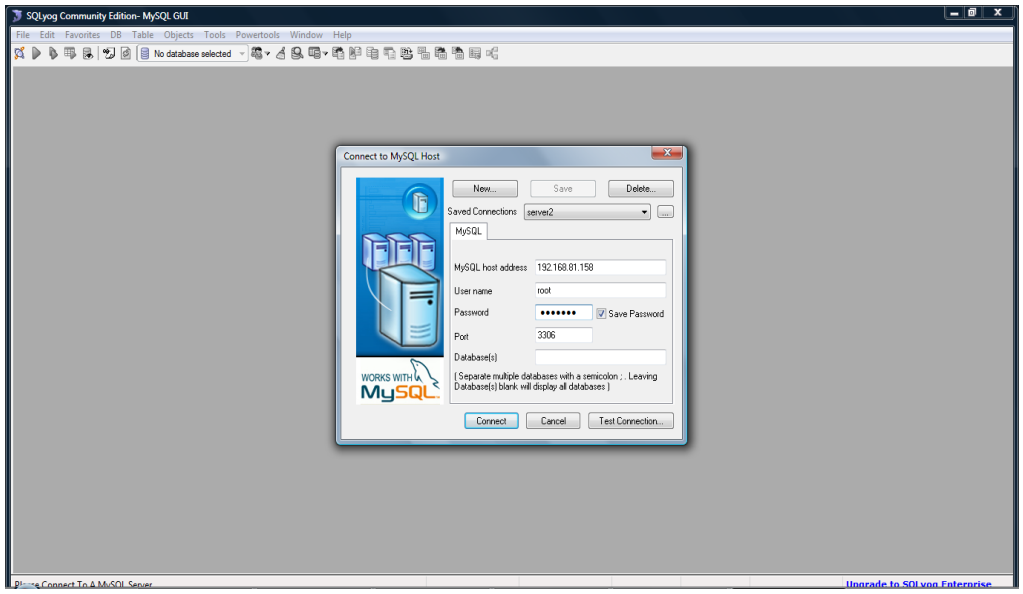

Figure 4.5 SQLyog Interface

4.1.8 HAProxy

HAProxy stands for High Availability Proxy and is used by companies such as RightScale for load balancing and server fail over in the cloud. Companies do not want their website to go down, or worse, for users to notice the site is down. Due to this larger websites will run on multiple servers to provide some level of high availability. In HAProxy there is typically a load balancer to distribute the load among a pool of web servers. It also works as a fault tolerant system. Whenever a server goes down it is taken out of the pool until it is once again ready to handle requests. HAProxy has the ability to perform this task by doing periodic health checks

on all the servers in a cluster. Even if one of the application servers is not working, users will still have the availability to the application. HAProxy will properly handle the request from users by redirecting them to the second server, giving the impression that all is well.

HAProxy is a free, very fast and reliable solution offering high availability, failover and load balancing mechanism, and proxying for TCP and HTTP based applications. When HAProxy is running in HTTP mode, both the request and the response are fully analyzed and indexed, thus it becomes possible to build matching criteria on almost anything found in the contents. However, it is important to understand how HTTP requests and responses are formed, and how HAProxy decomposes them. It will then become easier to write correct rules and to debug existing configurations.

HAProxy is installed on all "Front End" Server Templates. A "Front End" (FE) is a server with a static IP (typically a Elastic IP) and is registered with DNS. A normal setup has two "Front Ends." A FE server can also have an application server installed on it or it can be stand alone. The FE's purpose is to direct users to available application servers. It can also detect server failures (hardware or software), and provides client transparent fault tolerance. HAProxy can be easily migrated to another server in the presence of system failures.

HAProxy currently does not support the HTTP keep-alive mode, but knows how to transform it to the close mode. Right now, HAProxy only supports the first mode (HTTP close) if it needs to process the request. This means that for each request, there will be one TCP connection. If keep-alive or pipelining are required, HAProxy will still support them, but will only see the first request and the first response of each transaction. While this is generally problematic with regards to logs, content switching or filtering, it most often causes no problem for persistence with cookie insertion.

HAProxy takes care of all these complex combinations when indexing headers, checking values and counting them, so there is no reason to worry about the way they could be written, but it is important not to accuse an application of being buggy if it does unusual, valid things. It needs very little resource. Its event-driven architecture allows it to easily handle thousands of simultaneous connections on hundreds of instances without risking the system's stability.

HAProxy Configuration

HAProxy's configuration process involves three major sources of parameters:

- the arguments from the command-line, which always take precedence,
- the "global" section, which sets process-wide parameters,
- the proxies sections which can take form of "defaults", "listen", "frontend" and "backend".

First parameter is an optional list of arguments which may be needed by some algorithms. Here, the load balancing algorithm of a backend is set to roundrobin. The algorithm may only be set once for each backend [12]. Secondly process wide OS-specific parameters are used. They are generally set once for all and do not need to be changed. Our HAProxy configuration file supports keywords for process management and security and performance tuning. Keywords such as ‘stats’ and ‘ulimit-n’ are used for process management and security and for later ‘maxconn’ are used. Then for third parameter, "defaults" section sets default parameters for all other sections following its declaration. Those default parameters are reset by the next "defaults" section. A "frontend" section of HAProxy accepts client connections. A "backend" section of server 1 and server 2 connect to proxy for forwarding incoming connections A "listen" section defines a complete proxy with its frontend and backend parts combined in one section. It is generally useful for TCP-only traffic [29]. The following is the configuration file that has been created for this system.

```
#haproxy.cfg
defaults
contimeout      5000
clitimeout      50000
srvtimeout      50000
maxconn 250000
stats enable
stats auth root:ibaddat
balance roundrobin
cookie SERVERID insert indirect
frontend www 192.168.81.157:80
mode http
acl dyn_content url_sub cgi-bin
use_backend dyn_server if dyn_content
default_backend stat_server
```

```
backend dyn_server
balance roundrobin
mode http
server dserver1 192.168.81.158:8080 cookie A check
server dserver2 192.168.81.155:8080 cookie B check

backend stat_server
balance roundrobin
mode http
server sserver1 192.168.81.158:8080 cookie A check
server sserver2 192.168.81.155:8080 cookie B check
```

Experimental results in next chapter shows that HAProxy can assist server applications to recover from server failures in just a few milliseconds with minimum performance overhead.

4.1.9 Tool Alternatives for Fault Tolerant System

Autonomic fault tolerant system for cloud environment has different configurations and deployments. Assure and SHelp are the tools already used for autonomic fault tolerance in cloud environment.

- **Assure**

Assure [30] system is proposed to better address faults by introducing rescue point and error virtualization techniques. Rescue points are locations in the existing application code used as possible checkpoints for handling programmer-anticipated failures and the error virtualization is a technique used to force a heuristic-based error return in a function. Once an appropriate rescue point is used to successfully bypass a fault in the offline triage production, it is deployed in the online production by inserting codes at the appropriate rescue point to checkpoint the application state once the function is called. Thus the server application can recover quickly from the same fault in the future. However, one potential problem is, if the appropriate rescue point is in a loop or the main procedure, the appropriate rescue point can be called very frequently during normal executions, which may cause high overhead in both space and time, and sometimes even make the server applications not function properly. Although ASSURE can alleviate the problem by strengthening patch test phase, it may face an even worse situation in which ASSURE may be unable to find an

appropriate rescue point because selecting a upper level function as the appropriate rescue point may cause the application to exit directly when responding to the fault [31].

- **SHelp**

SHelp [31] is a lightweight runtime system that can survive software failures in the framework of virtual machines. SHelp extends ASSURE to a virtualized computing environment with two distinctive features. First, SHelp adopts a two-level storage hierarchy for rescue point management. It deploys a global rescue point database in a privileged domain (called Dom0 in Xen [32]) that can be accessed by all virtual machines, and a rescue point cache in each guest operating system (called DomU) to store a small number of rescue points relevant only to the applications running in the virtual machine. By introducing this two-level storage hierarchy for rescue points in the virtual machine environment, SHelp can dramatically reduce the redundancy because there is no need to build the same rescue point database in every virtual machine, make it very convenient for multiple application instances running in different virtual machines to contribute and share the fault-related information, and thus enable more effective recoveries from the failures. The second distinctive feature of SHelp is the introduction of weight values to rescue points. When an appropriate rescue point is chosen, its associated weight value is incremented. After a fault is detected, the system will first select the appropriate rescue point with the largest weight value [31].

4.1.10 Core Implementation

This section discusses how an application was developed and Tomcat, MySQL were implemented.

First the Ubuntu platform was set up in a virtual environment. Then application was developed in Java and HTML. Then Tomcat was configured on server 1 and server 2. Tomcat requires setting the Java path and following commands to run on in it.

#Java path is added to file /etc/profile as follows:

export JAVA_HOME=/java/jdk1.6.0_18/jre/bin/java

export PATH=$PATH:/java/jdk1.6.0_18/jre/bin

#Tomcat on server 1 and server 2 starts with this command:

root@vishonika-desktop:cd /tomcat/apache-tomcat-6.0.32/bin

root@vishonika-desktop:/tomcat/apache-tomcat-6.0.32/bin# ./startup.sh

#To check the successful startup of Tomcat, we can view the catalina.out log file, which includes the running processes. Last few lines will look like this:

```
INFO: Deploying configuration descriptor manager.xml
19  May,  2011  8:51:32  PM  org.apache.catalina.startup.HostConfig
deployWAR
INFO: Deploying web application archive Login.war
19  May,  2011  8:51:32  PM  org.apache.catalina.startup.HostConfig
deployDirectory
INFO: Deploying web application directory docs
19  May,  2011  8:51:32  PM  org.apache.catalina.startup.HostConfig
deployDirectory
INFO: Deploying web application directory ROOT
19  May,  2011  8:51:32  PM  org.apache.catalina.startup.HostConfig
deployDirectory
INFO: Deploying web application directory examples
19 May, 2011 8:51:33 PM org.apache.coyote.http11.Http11AprProtocol
start
INFO: Starting Coyote HTTP/1.1 on http-8080
19 May, 2011 8:51:33 PM org.apache.coyote.ajp.AjpAprProtocol start
INFO: Starting Coyote AJP/1.3 on ajp-8009
19 May, 2011 8:51:33 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2057 ms
```

#Tomcat on server 1 and server 2 can be stopped with this command:

root@vishonika-desktop:cd /tomcat/apache-tomcat-6.0.32/bin

root@vishonika-desktop:/tomcat/apache-tomcat-6.0.32/bin# ./shutdown.sh

#MySQL starts with this command:

root@vishonika-desktop:cd /opt/lamp# ./lampp startmysql

#MySQL console starts with this command:

root@vishonika-desktop:/opt/lampp/bin# ./mysql -u root –p

Screen will show the following output:

root@vishonika-desktop:/opt/lampp/bin# ./mysql -u root –p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 3

Server version: 5.1.37 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

#Databases can be shown with this command:

mysql> show databases;

#Screen will show the following output:

```
+--------------------------+
| Database                 |
+--------------------------+
| information_schema |
| cdcol                    |
| mysql                    |
| phpmyadmin           |
| test                     |
| vishu                    |
+--------------------------+
6 rows in set (0.00 sec)

mysql> exit
```

#HAProxy starts with this command:

root@vishu-desktop:/ cd /opt/haproxy# nohup ./haproxy -f ./haproxy.cfg -p /var/run/haproxy.pid &

Chapter 5
Experimental Results

This chapter focuses on implementation of Autonomic Fault Tolerance in Virtualized Cloud Environment using HAProxy and MySQL Replication for server application. This chapter shows the experimental results of this approach.

5.1 Implementation and working of system

Fault tolerant system has been developed using HAProxy and MySQL. HAProxy is used to handle server failures in Fault Tolerant Cloud Environment. HAProxy provides a web interface for statistics known as HAProxy statistics. Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Application can be accessed on any of the web server. Its basic functionalities are shown as snapshots below. Figure 5.1 shows the User Login Web Interface.

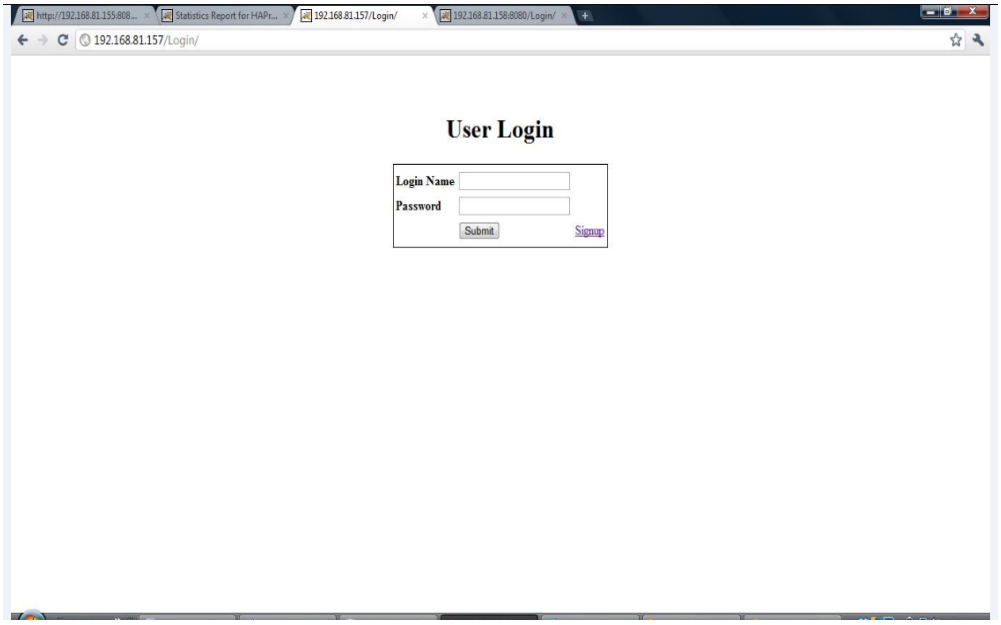

Figure 5.1 User Login Web Interface

Figure 5.2 - 5.4 shows the procedure of user login. Figure 5.2 shows the signup for user login. Data is entered after clicking on the signup button. Figure 5.3 shows the login details entered by the user. On clicking submit button from figure 5.3, details are successfully entered in both Master and Slave databases. Figure 5.4 shows the success of submitting the details.

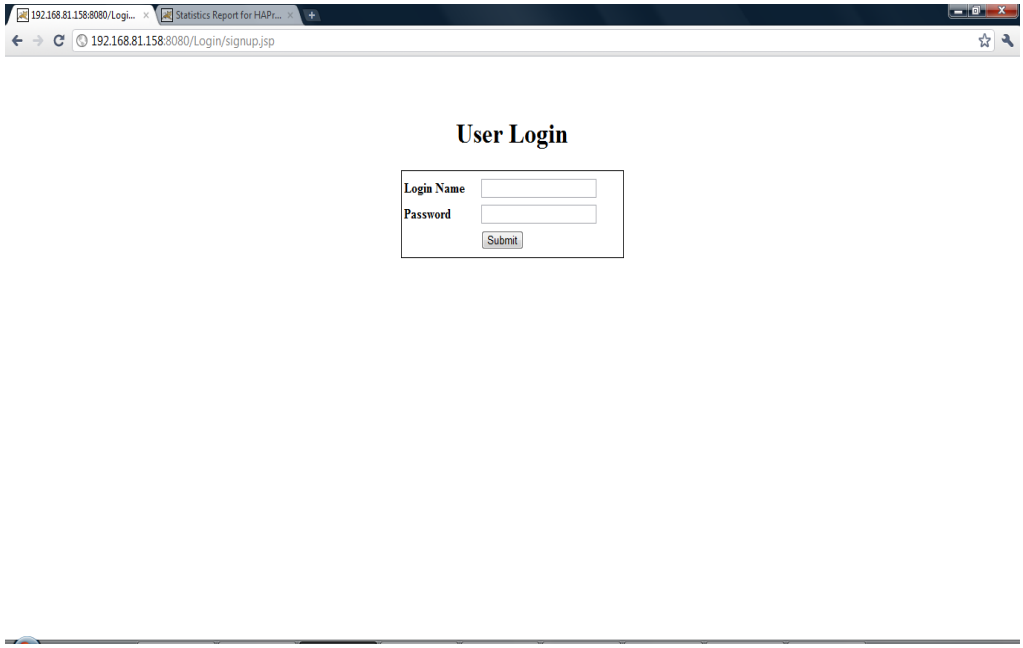

Figure 5.2 Signup for User Login

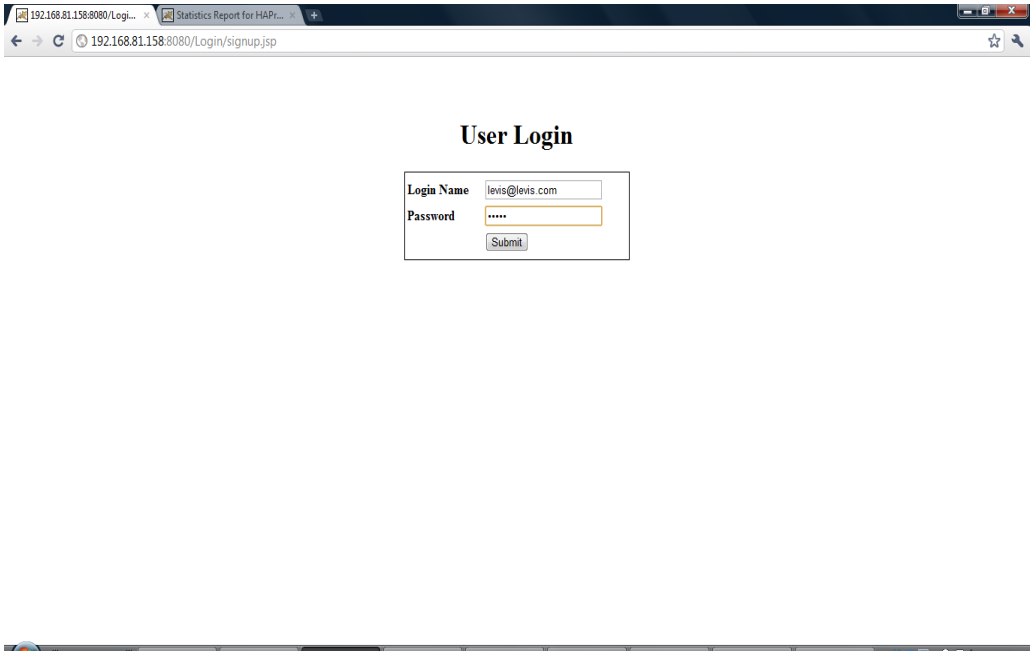

Figure 5.3 Login details entered by user

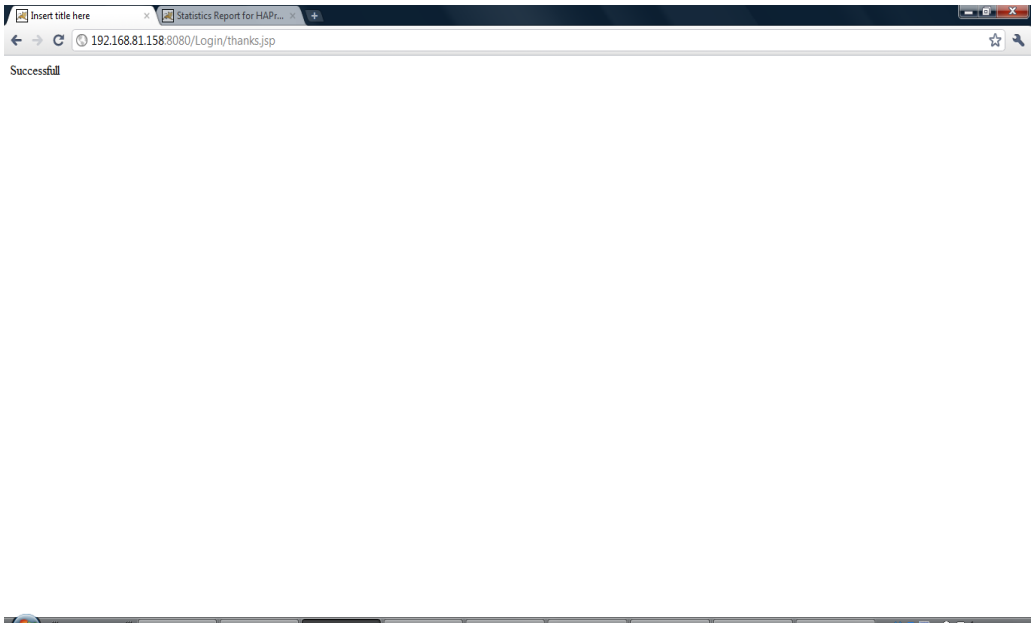

Figure 5.4 Server Message regarding login

Figure 5.5 – 5.8 shows the database entries in both master and slave tables using SQLyog. User access the server 1 (master) with IP address 192.168.81.158 as shown in Figure 5.1 - 5.4. Figure 5.5 shows the connection establishment with server 1. Figure 5.6 shows the database table of server 1. Figure 5.7 shows the connection establishment with server 2 slave (slave). Figure 5.8 shows the replicated database table of server 2.

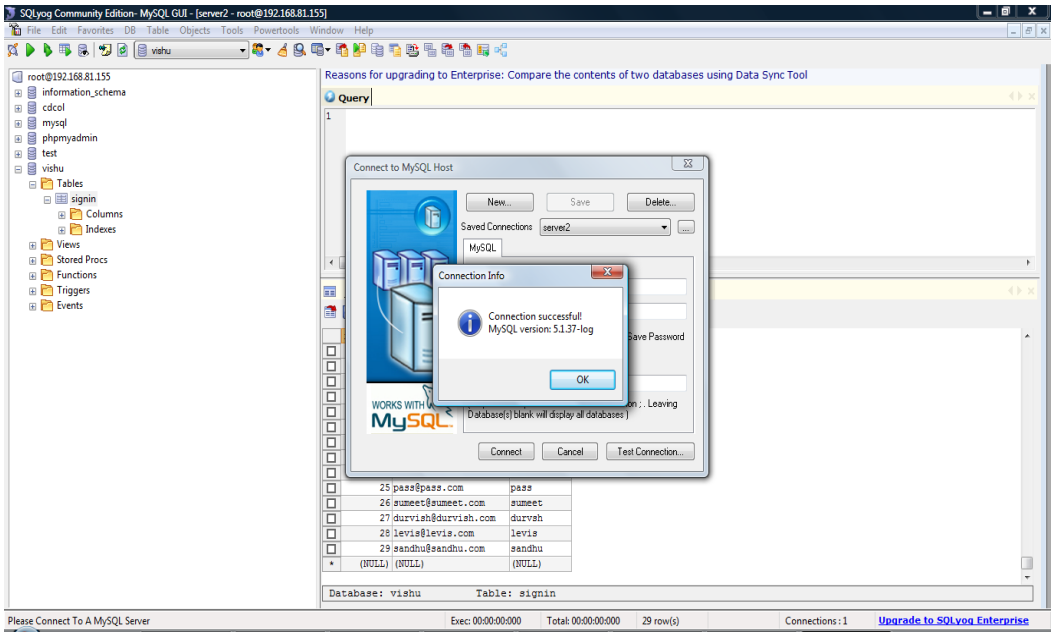

Figure 5.5 SQLyog connected with server 1

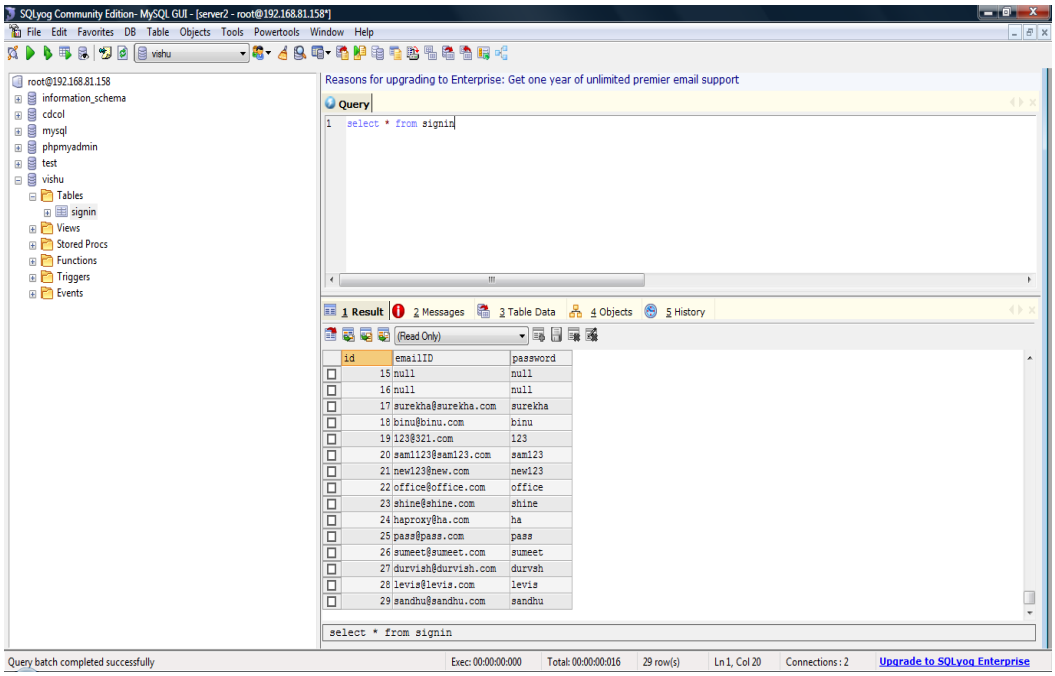

Figure 5.6 Database entries of sever 1

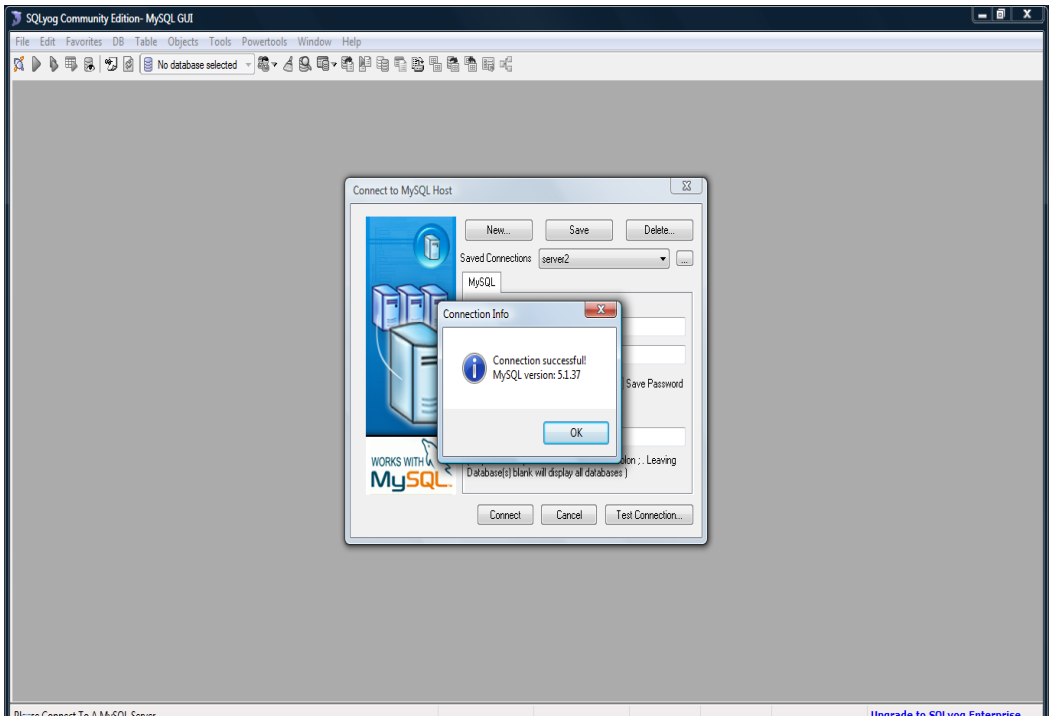

Figure 5.7 SQLyog connected with server 2

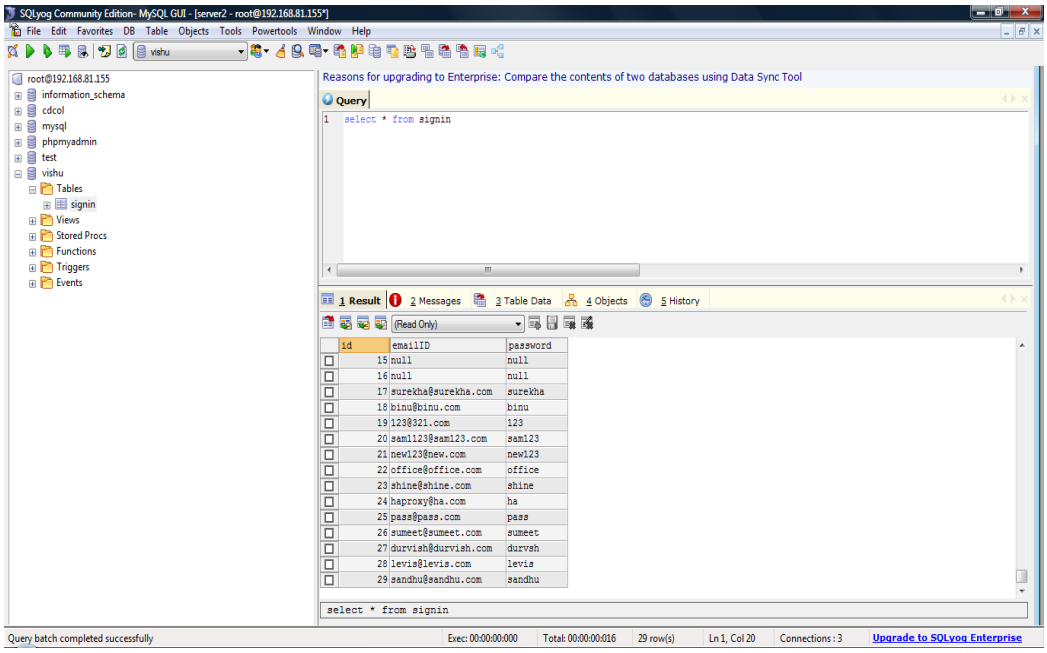

Figure 5.8 Database entries of server 2

Figure 5.9 – 5.12 shows the HAProxy statistics of server 1 and server 2. Figure 5.9 shows the authentication required to access HAProxy statistics. Figure 5.10 shows the stats when both the servers are up. Figure 5.11 shows the stats when server 1 goes down and server 2 is still up. Figure 5.12 shows the stats when server 2 goes down and server 1 is still up.

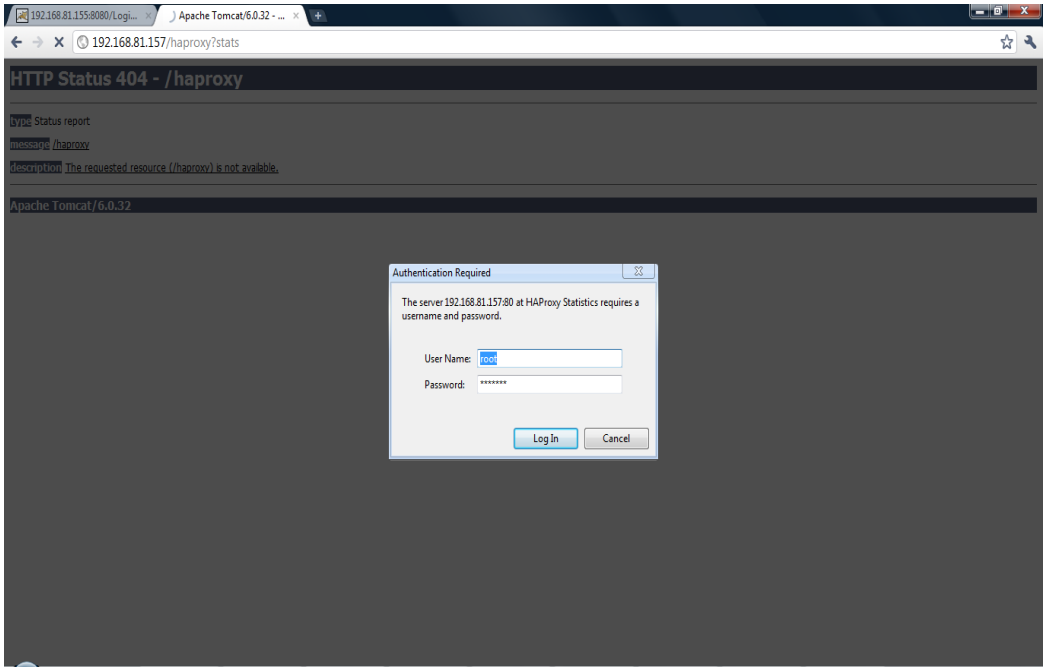

Figure 5.9 HAProxy Authentication

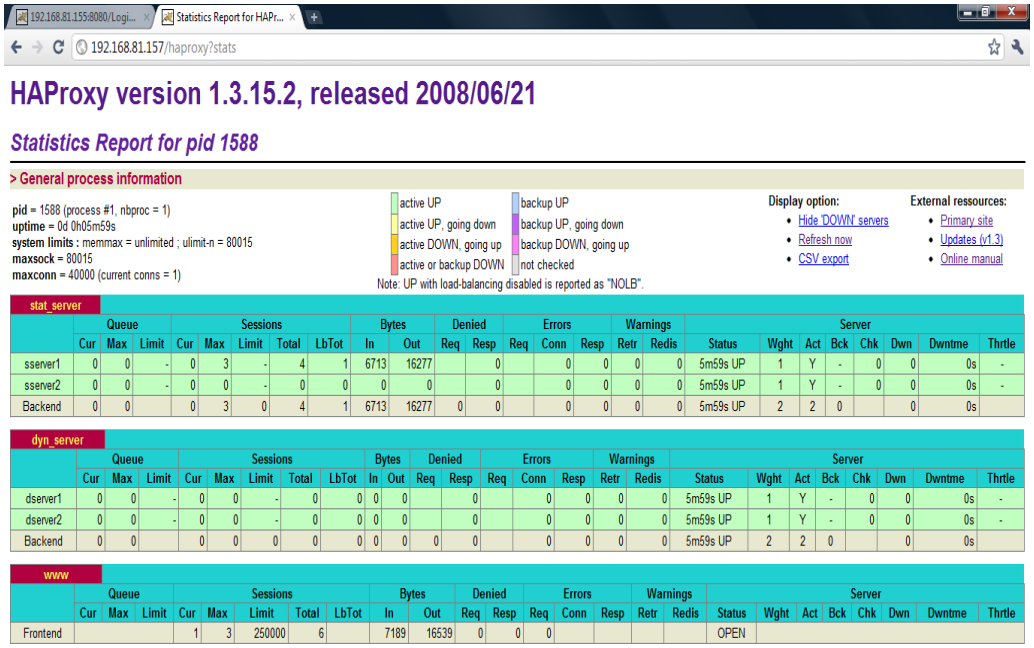

Figure 5.10 HAProxy statistics

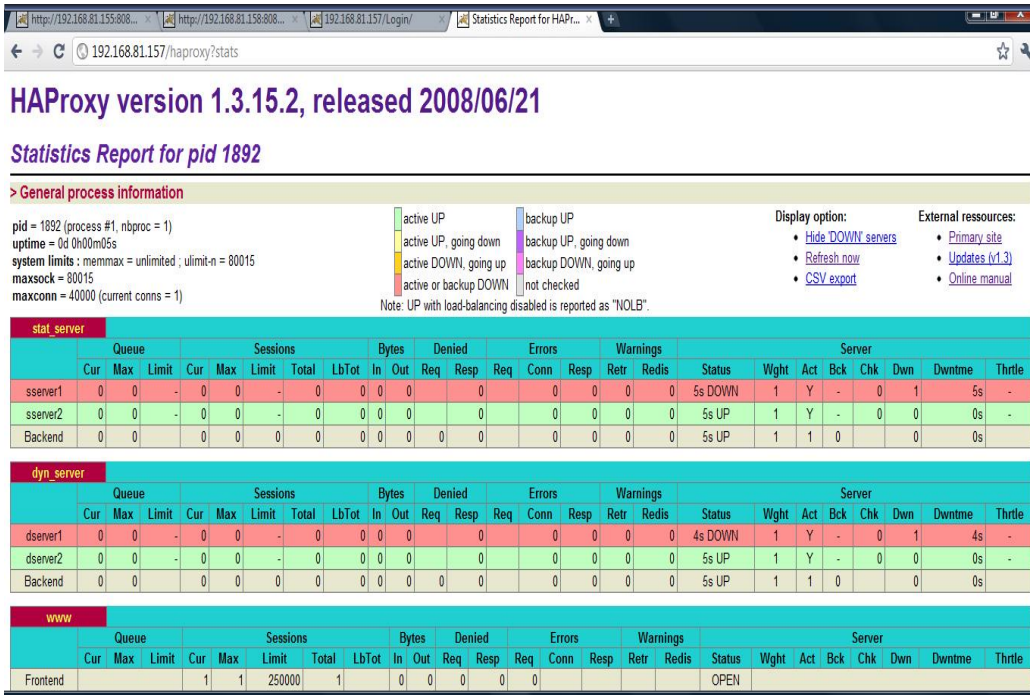

Figure 5.11 HAProxy statistics when server 1 is down

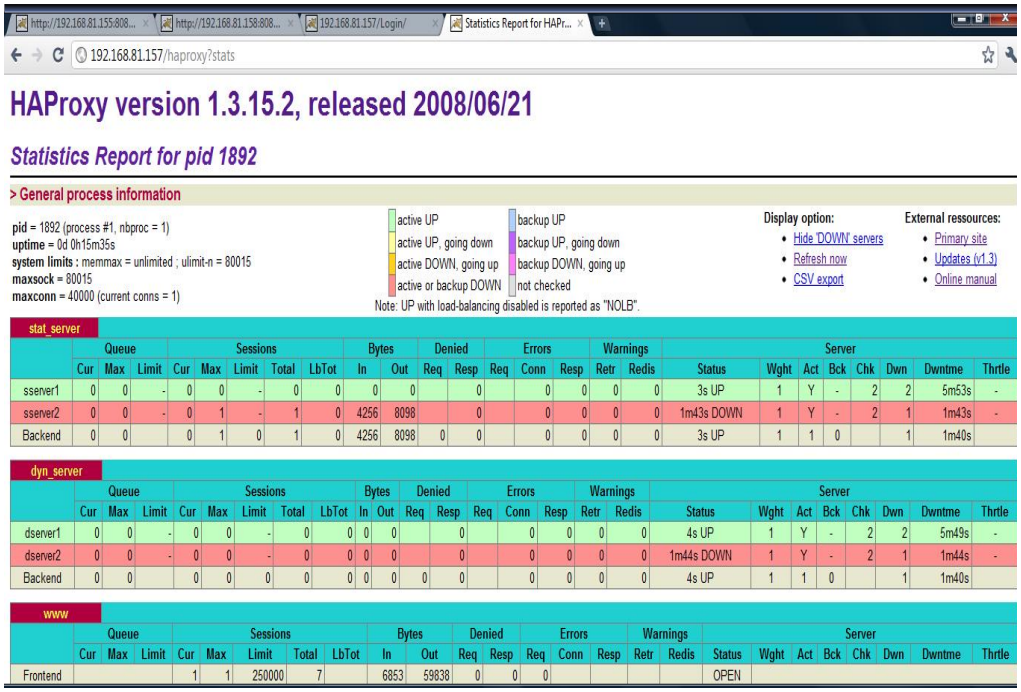

Figure 5.12 HAProxy statistics when server 2 is down

Chapter 6

Conclusion and Future Scope

This chapter discusses the conclusions of work presented in this thesis. The chapter ends with a discussion of the future direction which thesis work can take.

6.1 Conclusion

In this thesis cloud virtualized system architecture based on HAProxy has been proposed. A highly reliable system is presented that can deal with various software faults for server applications in a cloud virtualized environment. A similar database application is deployed on multiple virtual machines. The database servers are replicated for better performance and data consistency. These applications are evaluated and experimental results are obtained, that validate the system fault tolerance.

6.2 Thesis Contributions

- Proposed Cloud Virtualized System architecture.
- Development of replication on MySQL database servers.
- Implementation of autonomic fault tolerant system in cloud environment.

6.3 Future Research

- This work shows implementation of the single application on both the servers. Multiple applications can be included for future extensions.
- Real time data can be analyzed and monitored using HAProxy.
- Data Replication can be used to handle errors in databases.

References

[1] Fenn, Jackie, Nikos Drakos, Whit Andrews and et al.,"Hype Cycle for Emerging Technologies", Gartner, 2008.

[2] Markus Bohm, Stefanie Leimeister, Christoph Riedl, Helmut Krcmar," Cloud Computing and Computing Evolution, (TUM)", Germany.

[3] M. Armbrust, A. Fox, and et al., "Above the clouds: A berkeley view of cloud computing", UC Berkeley, Tech. Rep. UCB/EECS-2009-28, February 2009.

[4] K. Birman, G. Chockler, and R. van Renesse, "Toward a cloud computing research agenda", SIGACT News, vol. 40, no. 2, pp. 68–80, 2009.

[5] Jing Deng, Scott C.-H. Huang, Yunghsiang S. Han, and Julia H. Deng," Fault-Tolerant and Reliable Computation in Cloud Computing, Intelligent Automation", Inc., Rockville, MD, USA, 2010.

[6] Russell Craig, Jeff Frazier, Norm Jacknis, Seanan Murphy, Carolyn Purcell, Patrick Spencer, JD Stanley," Cloud Computing in the Public Sector: Public Manager's Guide to Evaluating and Adopting Cloud Computing", Cisco (IBSG), November 2009.

[7] Cloud platforms available at http://interactivesnack.wordpress.com/2009/06/.

[8] "Introduction to Cloud computing", white paper, Think Grid.

[9] Ian Foster, Yong Zhao, Ioan Raicu, Shiyong Lu," Cloud Computing and Grid Computing 360-Degree Compared", 2008.

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic," Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generation Computer Systems", 25(6): 599-616, Elsevier Science, Amsterdam, The Netherlands, June 2009.

[11] Cloud Pyramid available at," http://blog.gogrid.com/2008/06/24/the-cloud-pyramid/".

[12] X. Chu et al.," Aneka: Next-generation enterprise grid platform for e-science and e-business applications", Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, 2007.

[13] J. E. Smith and R. Nair," Virtual Machines: Versatile platforms for systems and processes", Morgan Kauffmann, 2005.

[14] R. Buyya and M. Murshed," GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience", 14(13-15), Wiley Press, Nov.-Dec., 2002.

[15] http://cloud-simulation-frameworks.wikispaces.asu.edu/

[16] "Cloud servers Development Guide", Rackspace Hosting, API v1.0 (April 15, 2011).

[17] Mostafa Abd-El-Barr," Design and Analysis of Reliable and Fault-Tolerant Computer Systems", 2006.

[18] Zaipeng Xie, Hongyu Sun and Kewal Saluja, "A Survey of Software Fault Tolerance Techniques".

[19] Golam Moktader Nayeem, Mohammad Jahangir Alam," Analysis of Different Software Fault Tolerance Techniques",2006.

[20] Steven Y. Ko, Imranul Hoque, Brian Cho and Indranil Gupta, "Making Cloud Intermediate Data Fault-Tolerant" June 2010.

[21] Steven Y. Ko, Imranul Hoque, Brian Cho and Indranil Gupta, "On Availability of Intermediate Data in Cloud Computations" , 2010.

[22] Manish Pokharel and Jong Sou Park, "Increasing System Fault Tolerance with Software Rejuvenation in E-government System", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010.

[23] Antonina Litvinova, Christian Engelmann and Stephen L. Scott," A Proactive Fault Tolerance Framework For High Performance Computing", 2009.

[24] Geoffroy Vallee, Kulathep Charoenpornwattana, Christian Engelmann, Anand Tikotekar, Stephen L. Scott," A Framework for Proactive Fault Tolerance".

[25] A.B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott.," Proactive fault tolerance for HPC with Xen virtualization", ICS '07: Proceedings of the 21st annual international conference on Supercomputing, pages 23–32, New York, NY, USA,2007. ACM Press.

[26] Bernardetta Addis, Danilo Ardagna, Barbara Panicucci and Li Zhang, " Autonomic Management of Cloud Service Centers with Availability Guarantees", IEEE third International Conference on Cloud Computing, 2010.

[27] Imad M. Abbadi, "Self-Managed Services Conceptual Model in Trustworthy Clouds' Infrastructure", 2010.

[28] Yaakoub El Khamra , Yaakoub El Khamra ,"Developing Autonomic Distributed Scientific applications: A Case Study From History Matching Using Ensemble Kalman-Filters", GMAC'09, June 15, 2009.

[29] http://haproxy.1wt.eu/download/1.3/doc/configuration.txt.

[30] S. Sidiroglou, O. Laadan, C. Perez, N. Viennot, J. Nieh, and A. D. Keromytis, "ASSURE: Automatic Software Self-healing Using REscue points", Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09), ACM Press, Washington, DC, USA, pp.37-48, March 7-11, 2009.

[31] Gang Chen, Hai Jin, Deqing Zou, Bing Bing Zhou, Weizhong Qiang, Gang Hu, "SHelp: Automatic Self-healing for Multiple Application Instances in a Virtual Machine Environment", IEEE International Conference on Cluster Computing, 2010.

[32] Golam Moktader Nayeem, Mohammad Jahangir Alam, "Analysis of Different Software Fault Tolerance Techniques", 2006.

[33] Ubuntu Server Guide, 2010 is available at "https://help.ubuntu.com/10.04/serverguide/C/serverguide.pdf".

[34] http://tomcat.apache.org/

[35] Dalibor D. Dvorski," Installing, configuring and developing with Xampp", March 2007.

[36] MySQL Replication, Oracle, 2011

[37] Amazon Elastic Compute Cloud (EC2). http://www.amazon.com/ec2/

[38] Amazon Simple Storage Service (S3). http://www.amazon.com/s3/

[39] Google App Engine. http://appengine.google.com

[40] Microsoft Azure. http://www.microsoft.com/azure/

[41] Sun network.com (Sun Grid). http://www.network.com

[42] Dr. Yi Pan," Fault Masking in the Cloud and Cloud Self-Monitoring and Autonomic Control".

[43] VMware user's manual is available at "http://www.vmware.com/pdf/ws32_manual.pdf".

Paper Published/Communicated

1. Vishonika Kaushal, Anju Bala, “Autonomic Fault Tolerance Using HAProxy in a Virtual Machine Environment”, International Journal of Computer Science Issues Volume 8, Issue 4, July 2011. (Communicated).