Abstract

Cloud Computing is an important technology that is revolutionizing the field of computers. However, there are no good benchmarking tools for assessing the performance of cloud systems. The requirements for a cloud system benchmark are different from those of traditional systems in a data center. Therefore, traditional benchmarks cannot be directly applied. Examples of these differences include the use of sharding in database technology, and the distribution of servers over multiple clouds in the case of hybrid clouds.

In this project, we propose a new benchmark for cloud systems, including hybrid cloud systems. This will enable us to assess the performance and hence more informed decisions can be taken while deciding on the cloud architecture setup required for internet services. The benchmark models an online trade scenario.We demonstrate how the performance of servers to respond to the requests vary with the way load is distributed in the backend servers. Our benchmark application is a modification of an existing benchmark - the Apache Daytrader. This is originally an application used to benchmark standalone application servers. This has been extended and set up in a hybrid cloud architecture. We prove the efficiency of our benchmark in estimating the performance of the server in a trade workload.

Acknowledgement

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and college. I would like to extend my sincere thanks to all of them.

We are highly indebted to Mr. Dinkar Sitaram and Mr. Phalachandra for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. We also thank Mr. Chandrashekar for providing us resources to work with in the CCBD lab at PES Institute of Technology

We are eternally grateful to Mr. Nitin V. Pujari, HOD CS Department to give us this opportunity to undertake this project.

Table of contents

I Introduction

The objective of the project is to design a relevant benchmarking application for the cloud. This is done by using an existing application - Apache Daytrader which was developed by IBM to benchmark standalone application servers based on a stock market scenario.This tool cannot be directly used to since the requirements for cloud system benchmark are different from the traditional application servers.

We deal with various scenarios of cloud architectures to compare and contrast the benchmarking results .By using our benchmark , the customer can take more informed decisions while choosing the type of architecture required for his application.We are going to test our benchmark tool on private and public clouds.Also in each scenario , we will be changing various parameters like the number of users in DayTrader , the number of application servers , WAN speed etc. By obtaining results for all these cases , we will get a sound benchmarking tool for cloud system .

The previous research in this area doesn’t involve any reference to the type of load used. By mimicking a stock market scenario, we are benchmarking the cloud for a real - world loading of the cloud.

II Problem Definition

Use and Setup the existing daytrader application to benchmark hybrid clouds in various scenarios and varying several parameters like the no of users, no of backend servers, read/write ratio, clustering and WAN delay.

There is no free and open benchmark to test cloud architecture for a stock trading application. This benchmark will help design real life scenarios and test the reaction of the cloud in these setups.

III Literature Survey

These were the steps followed by us before starting the project:

1. Understand the code design of DayTrader Application to gain an insight on how it can be used for benchmarking a particular type of load, that of a stock market to be more precise.
2. The use of a web loader in the present scenario.
3. The importance of using a load balancer/ reverse proxy to make the application cloud scalable and how to add this component in the present condition.
4. The different components of openstack since that is the cloud we would be making use of.
5. Understand the way in which MySQL database should be setup with Apache geronimo . Currently Daytrader is set up with embedded derby . Had to decouple the application from this and set up Mysql instead.

Since we have planned to follow an iterative model,which is detailed further in the document, so each each step might have to revisited in future.

IV Project Requirement definition

4.1 Functional Requirements

4.1.1 Introduction

- The project requires us to design a benchmarking tool .Since we are using the DayTrader application , this has to be made a cloud scalable application.
- The server used by DayTrader application is Apache Geronimo server .In order to simulate the HTTP requests , JMeter loader is used .These requests are delegate to server using Nginx load balancer .The delegation is done based on the id of the user which gets randomly generated by Loader.
- The backend database used to populate users is MYSql Db.
- The other main requirement is, to use this benchmarking tool to decide the cloud architecture required for the particular application.

4.1.2 Inputs

The Input for this project is :
The HTTP requests which are simulated by JMeter .These request ping the Trade Scenario servlet and produces random transaction ids and random actions ie either buy or sell stocks for the user. The number of requests and the number of iterations can be specified.

4.1.3 Processing

- The Http Requests from Jmeter have randomly generated ids of users.
- These are passed to the NGinx proxy web server .
- This server proxies the requests to the backend servers based on id in the http get request parameter.
- The request which reaches the backend server makes a transaction and updates the corresponding database .
- The Database which is configured is MYSQL database.

- The throughput of processing the requests can be visualised graphically in JMeter UI. Also , the logs of Jmeter can be used to understand the output.

4.1.4 Outputs

- The graphical results produced by JMeter for the requests made. From these , the performance of the servers can be deciphered. This can be tested on various types of servers on clouds hence can give the benchmarking statistics . This will enable one to make informed decisions while selecting the architecture of cloud required for the application .
- The logs produced by Jmeter can be imported with xsl in MS Excel to decipher the results better.
- The logs produced by Nginx can be used to decipher the load ratio with which the backend servers are pinged.

4.1.5 Error Handling

- Log files of JMeter to see how the requests are created .
- Log files of Nginx to see to which backend server the request is forwarded.
- Log files of DayTrader application to trace the transaction made.

4.2 Use Cases

4.2.1 Selection of the appropriate cloud environment in a hybrid architecture.

4.2.2 The amount of load which can be applied to a given cloud environment.

4.2.3 The extent of possible expansion of an application across or within the cloud.

4.3 Non-Functional Requirements

The non functional requirements of this project are :

- The size of RAM and the power of the virtual machines which hosts the

backend and proxy servers and the JMeter client
- The number of backend servers used .
- The quality of metric produced by the daytrader application.
- The ability of backend servers to handle requests based on transaction id.
- The WAN speed used .
- The number of users populated in the database.

4.3.1 Performance

- The performance of the virtual machines is an important measure to understand the capability of the servers and hence benchmark them.
- The performance of the DayTrader application which determines the accuracy of the benchmarking metrics provided .

4.3.2 Reliability

- The reliability of the outcome of the project which is the benchmarking metrics for different architecture clouds is mainly dependant on DayTrader application .This is not very accurate and hence must be improved . This is a part of second specific requirement.

4.3.3 Availability

- The various components involved in this project namely DayTrader application , Nginx, OpenStack environment are all open source and hence are easily available .The source code is accessible and hence the performance can be improved .

4.3.4 Security

- The security of the openstack cloud instance , web server , the DayTrader application and that of the database are all based on login - password mechanism.
- The security factor for this project is also applicable to the security of transactions made by the customers using the DayTrader application.

4.3.5 Portability

- The benchmarking metric should gauge all the kinds of servers in

different cloud architectures . These should also be reliable and accurate .

4.4 Logical Database Requirements

- Currently Derby Database is used for embedded database and MySQL for testing in hybrid environment. Derby database is built-in with the Apache geronimo server. It is used for storing the stock quotes and the user profiles are maintained. The daytrader application talks to this database for all its actions .
- This database can be replicated in a master slave manner

V System requirements specification

5.1 External Interface Requirements

5.1.1 User Interfaces

The interfaces utilised in this project are :

1. The openstack cloud instance and load command terminal where the Geronimo server is installed.
2. The geronimo server UI where the DayTrader application is installed.
3. The DayTrader application UI where the database is populated and other run time configurations are set.
4. The command terminal of NGinx where the logic for proxy is entered.
5. The JMeter UI where the requests are sent and the throughput graph is visualised

5.1.2 Hardware Interfaces

The progress of the project needed the support in terms of hardware requirements. The following were predominantly made use of:

- A stable cloud setup making use of virtual machines.
- Network configurations :
    - speed of connection between servers
    - round trip time between servers
    - Network card used
- OS configurations
    - Memory capacity : 4GB RAM, 500GB Hard disk
    - CPU speed : 2.27GHz
    - cores :
    - Energy inputs :
- A router to provide stable and secure communication between servers

Virtual machines deployed in the cloud environment.These have 4GB RAM and 500GB hard disk space. A minimum of four virtual machines are used . In the first two , the application instances are added .These act as backend servers. The third has NGinx proxy web browser running . The fourth has the client JMeter loader which loads the cloud with requests.

5.1.3 Software Interfaces

1. The Openstack cloud scalable OS instances.
2. The Geronimo J2EE application web server .
3. The Apache DayTrader benchmarking application.
4. NGinx web proxy server.
5. Apache JMeter Loader.

5.1.4 Communications Interfaces

1. Weekly presentation of the progress to the project guides.
2. Internal communication among the members in cloud lab and google hangouts

VI System design

6.1 Design Overview

The overview of the design is similar to that of how a cloud works, for detail refer to the diagram 6.1.
The design of the application DayTrader is based on MVC architecture,for more detailed flow-diagram see figure 6.2.

6.2 System Architectural Design

6.2.1 Architectural Design

The different scenarios tested on the cloud were:
→ Varying number of users within a cloud keeping the number of application servers constant
→ Varying number of users across cloud keeping the number of application servers constant.

6.2.2 Chosen system Architecture

The architecture of each of the systems involved have been detailed here:

- Nginx - Load Balancer
    1. The load balancer takes all the requests from clients.
    2. It then distributes the load across the cloud servers.
    3. The logic used for distributing load is weighted round robin.
    4. After getting the response from the server, it echoes it back to client.

- DayTrader Application
    1. It follows MVC architecture.
    2. The presentation layer consists of java servlets and JSPs which provide the UI for the application.
    3. The backend consists of JDBC.
    4. There are also message driven beans which act as controller.

- Derby Database
    1. A Derby database contains dictionary objects such as tables, columns, indexes, and jar files. A Derby database can also store its own configuration information.
    2. A Derby database is stored in files that live in a directory of the same name as the database. Database directories typically live in *system* directories.
    3. In this project , the derby database is already embedded with the Apache geronimo server.
    4. It can be populated using the DayTrader functionalities.

- MySQL Database
    1. Should be configured in Apache Geronimo to use MySQL Database by providing specific daytrader-mysql plugins and the appropriate drivers for the application server to communicate with the database.

6.3. Description Of Components

6.3.1 Jmeter

The Apache JMeter desktop Java application designed to load test functional behavior and measure performance.

- Apache JMeter may be used to test performance both on static and dynamic resources It can be used to simulate a heavy load on a server to make a graphical analysis of performance.
- In this project , it is used to send Http requests to the DayTrader application installed on the geronimo server.

6.3.2 Nginx

- This is a web proxy server which can be used to distribute the requests load to the backend servers based on some algorithm.
- In our project , weighted round robin algorithm is used to distribute the requests .
- Personalised logs can be got by defining logs in the default file of NGInx.

6.3.3 Geronimo Server

- Apache Geronimo is an open source server runtime to create Java/OSGi server runtimes that meet the needs of enterprise developers and system administrators.
- In this project, it acts as the container of the DayTrader application .

6.3.4 DayTrader Application

- DayTrader is benchmark application built around the paradigm of an online stock trading system.
- The application allows users to login, view their portfolio, lookup stock quotes, and buy or sell stock shares.
- With the aid of a Web-based load driver Apache JMeter, the real-world workload provided by DayTrader can be used to measure and compare the performance of Java Platform, Enterprise Edition (Java EE) application servers offered by a variety of vendor.
- In this project , DayTrader application is converted into a cloud deployable application so that it can benchmark the servers on the cloud.

6.3.5 Derby Database

- Apache Derby, an Apache DB subproject, is an open source relational database implemented entirely in Java .
- When DayTrader is deployed as a cloud application , the databases across the instances must be such that they have information of the clients based on the transaction id .
- There is provision to replicate derby database mimicking master slave relation . This might not be compatible for the project which might lead to the transition from derby database to mysql database.

6.3.6 MySQL Database

- MySQL Database is used to decouple database and the application server unlike Derby database where the database is embedded within the application server
- Using this master-slave model can be constructed
- Using this database can be pushed to a different cloud environment compared to the application server hence emulating communication across hybrid cloud architectures

VII Pseudo code

7.1 NGINX Load Balancer

NGINX web server is used for load balancing. Following is the code used to delegate the requests :

```
location /daytrader{
        if ($arg_uid ~ "8\d\d|9\d\d|10\d\d|11\d\d|12\d\d|13/d/d"){
            rewrite ^ http://10.20.1.6:8080/daytrader/scenario;
        }

        if ($arg_uid ~ "14\d\d|15\d\d|16\d\d|17\d\d"){
            rewrite ^ http://10.10.4.205:8080/daytrader/scenario;
        }


        if ($arg_uid ~ "18\d\d|19\d\d|2\d\d\d"){
            rewrite ^ http://10.20.0.134/daytrader/scenario;
        }


        if ($arg_uid ~ "[\d]|[\d\d]|[\d\d\d]"){
            rewrite ^ http://10.20.1.3:8080/daytrader/scenario ;
        }

        access_log sites-available/logs/mylog.log addr_combined ;
    }
```

7.2 Apache Daytrader

1. We need to generate the user id’s from a required ID in each database server. For this we need to build separate ear files to deploy in each application server. Generating required user-id requires changing the start value of the for loop in the file where the user are created.

2. Make the required changes in the plan.xml and pom.xml files corresponding to MYSQL in order to point to the right database IP address.

7.3 Apache Geronimo

The following code should be added to apache geronimo to support MySQL database

1. In the Advanced settings , add the following artifact :

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
</dependency>
```

2. Run the following command in the Geronimo repository folder in order to install the plugin driver .

```
mvn install:install-file
"-Dfile=mysql\mysql-connector-java\5.1.10\mysql-connector-java-5.1.10.jar
"
"-DgroupId=mysql"
"-DartifactId=mysql-connector-java"
"-Dversion=5.1.10"
"-Dpackaging=jar"
```

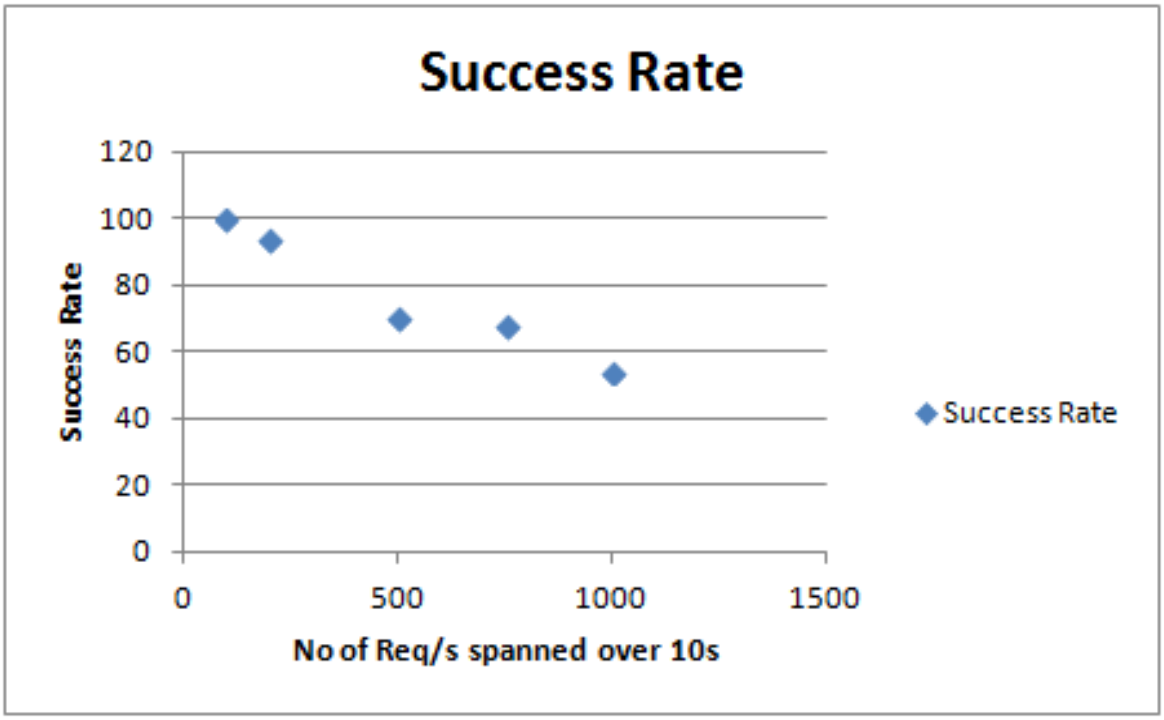
7.4 Database

Run the corresponding DDL file from DayTrader code in order to set up the schema.Populate this database using the user interface of DayTrader application.

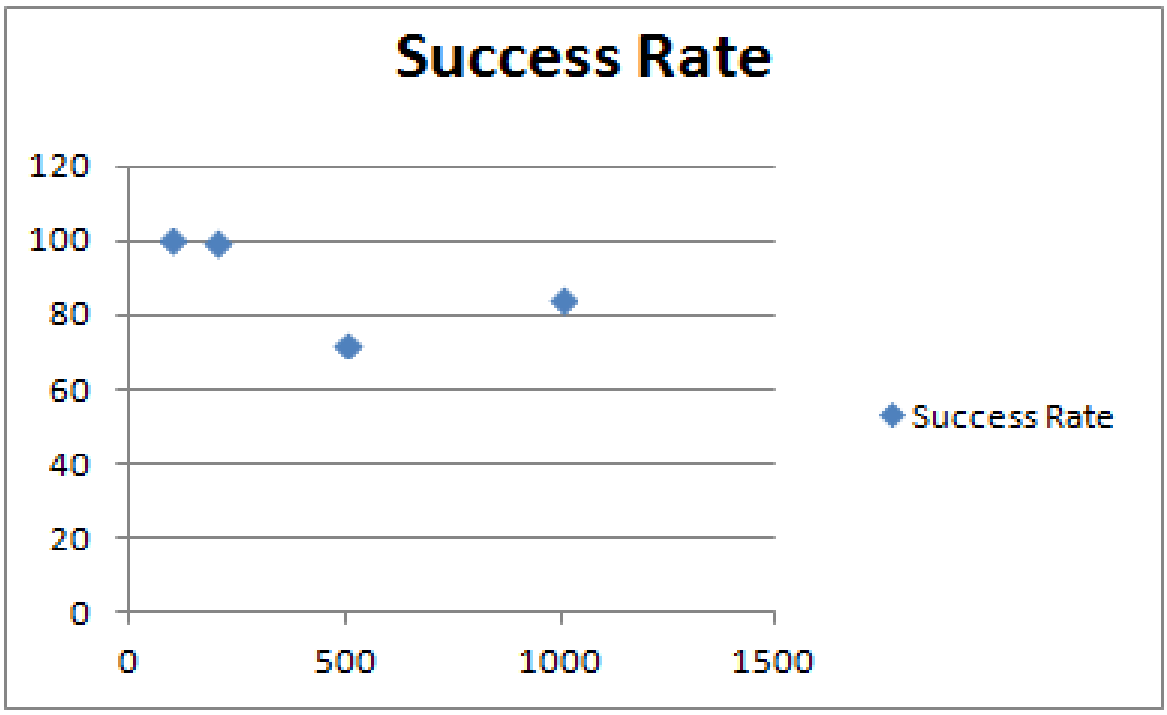
VIII Results discussion

8.1 Within Same Cloud :

8.1.1 One Server :

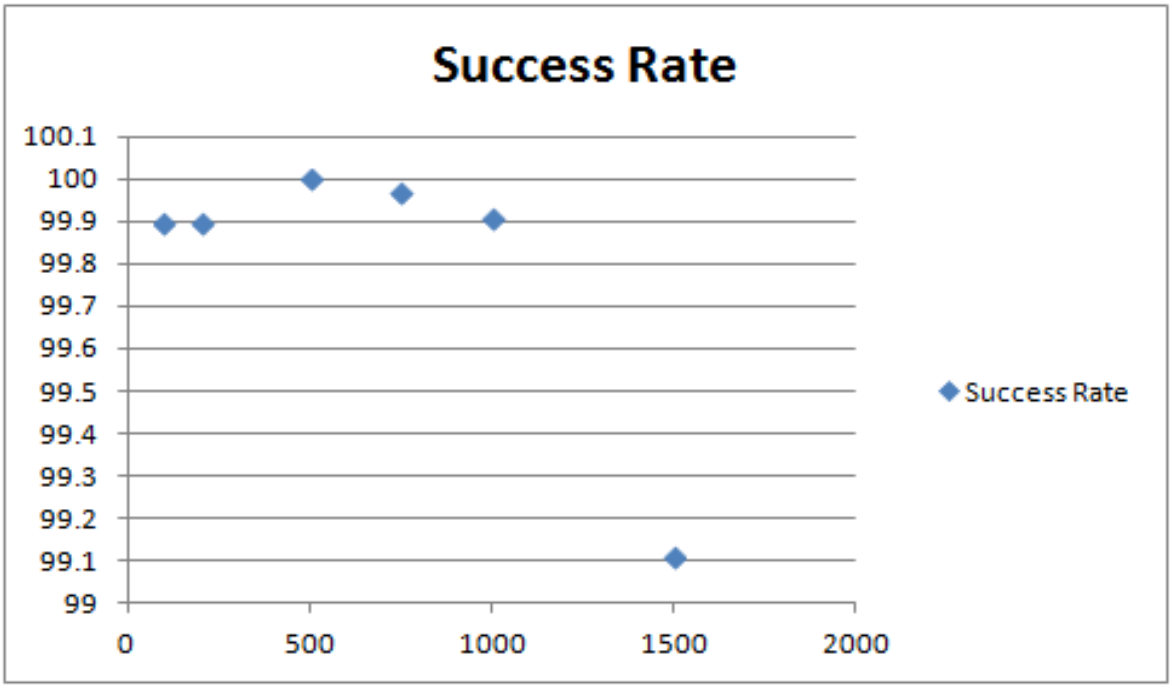


8.1.2 Two Servers :

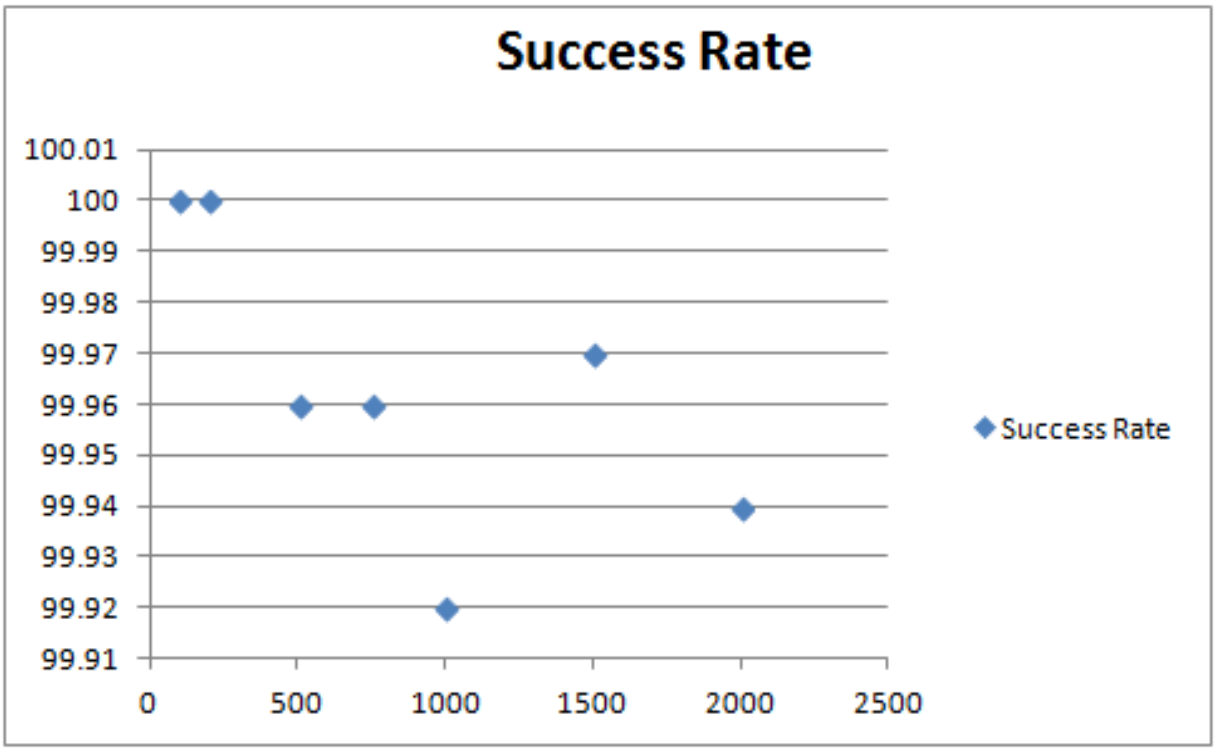


8.2 Across Cloud :

8.2.1 Run 1



8.2.2 Run 2



- As it can be seen from these graphs , the performance increases as the users are distributed among more number of databases.
- The performance becomes much better when the load is distributed in a hybrid cloud architecture.

IX Conclusion

The conclusions drawn from the test runs are:

- The average response time is reduced in a hybrid cloud setup
- The application can be used to benchmark a cloud in a trade market scenario and gather details about the cloud’s expected performance in a real life scenario
- The response time reduces when the number of instances hosting the application are increased
- The benchmark can be used to take decisions about when to scale to a hybrid cloud setup

X Further Enhancement

10.1 Complete Hybrid Setup with MySQL

- The current set of runs is done with Derby database which is embedded within the Apache geronimo server
- The setup will have two Scenarios
- Scenario 1: The database would be setup in one cloud environment and the application servers in a second cloud environment. Thus this would be simulating application servers in a public cloud and all the database servers in a private cloud.
- Scenario 2: The application and database servers are setup in both cloud environments thus simulating a application setup across both private and public clouds

10.2 Vary Different Parameters

- The no of users in the database should be varied
- The no of Backend servers can be changed
- Setting up the database as master and slave and measure the read/write ratio
- Use wlan simulator to simulate a delay in communication across the cloud

10.3 Research Paper

This work will be submitted as a research paper to an IEEE conference

XI Bibliography

- Apache Geronimo Documentation
  http://geronimo.apache.org/documentation.html
- Apache DayTrader Documentation

  http://geronimo.apache.org/GMOxDOC20/daytrader.html
- Right Scale solutions to design Hybrid Cloud structure

http://www.rightscale.com/lp/private-hybrid-cloud-white-paper.php

- Read me of Apache Day Trader :

http://svn.apache.org/repos/asf/geronimo/daytrader/tags/daytrader-2.2.1/README

List of figures and tables

Figure 6.1

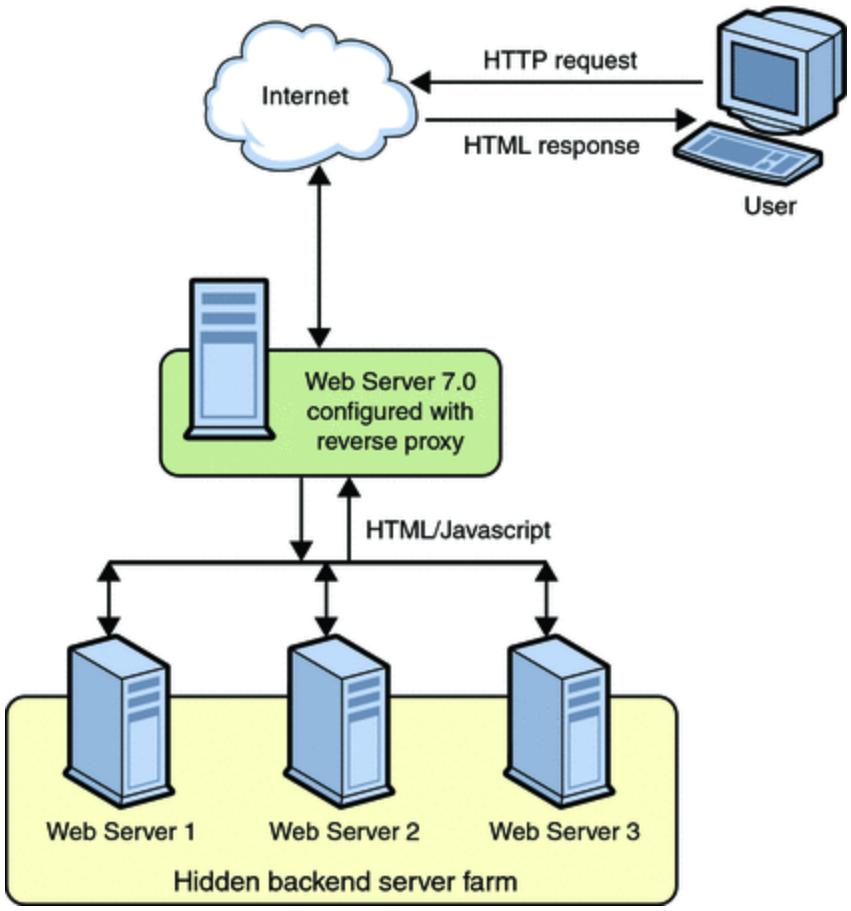


Figure 6.2

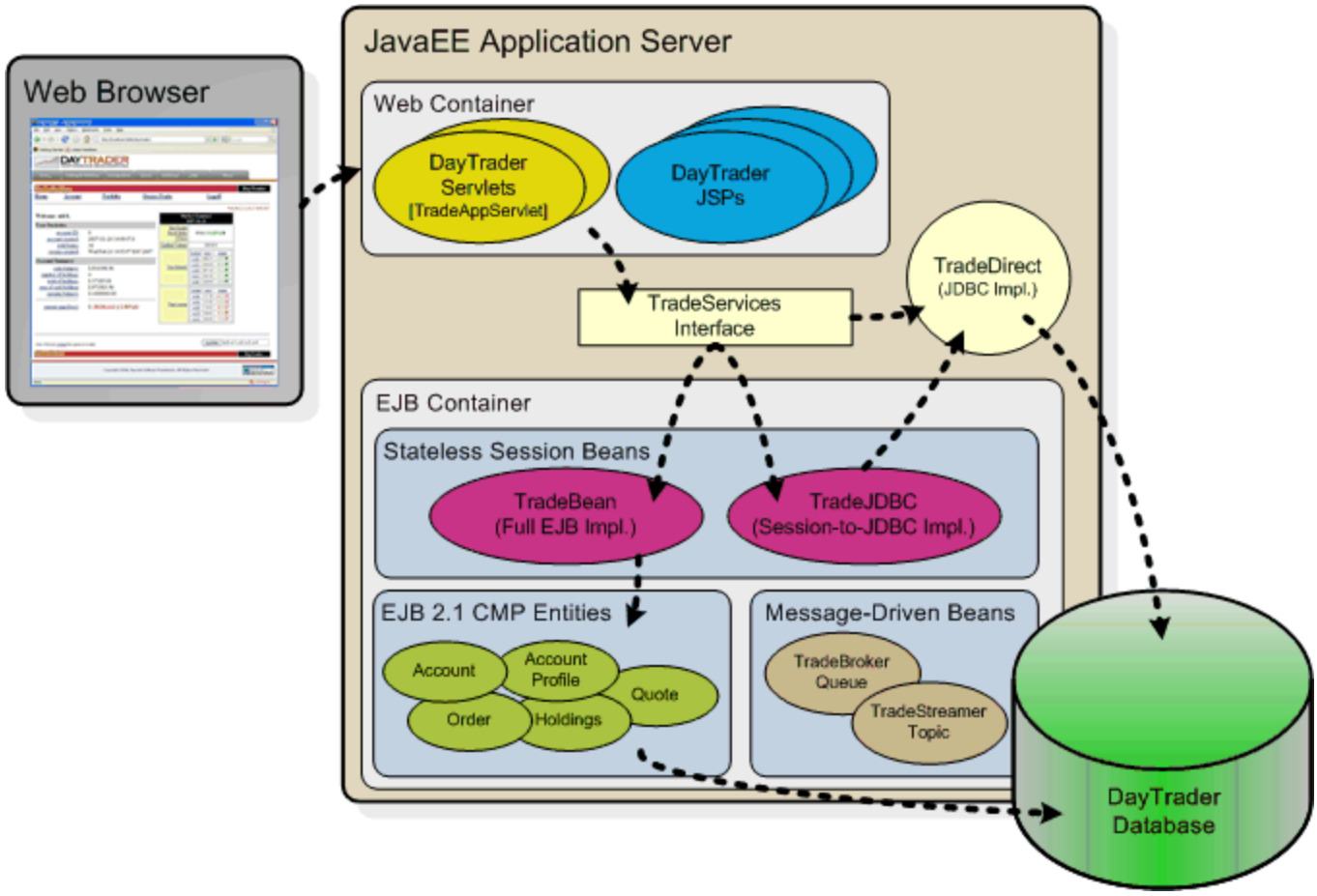

Web Browser
JavaEE Application Server
Web Container
DayTrader Servlets [TradeAppServlet]
DayTrader JSPs
TradeDirect (JDBC Impl.)
TradeServices Interface
EJB Container
Stateless Session Beans
TradeBean (Full EJB Impl.)
TradeJDBC (Session-to-JDBC Impl.)
EJB 2.1 CMP Entities
Account
Account Profile
Quote
Order
Holdings
Message-Driven Beans
TradeBroker Queue
TradeStreamer Topic
DayTrader Database

Figure 6.3

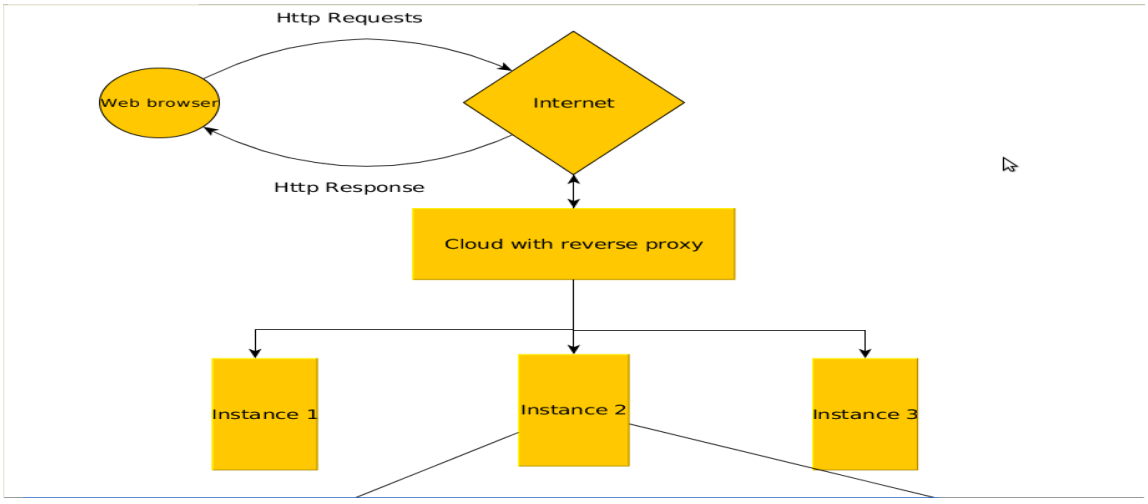
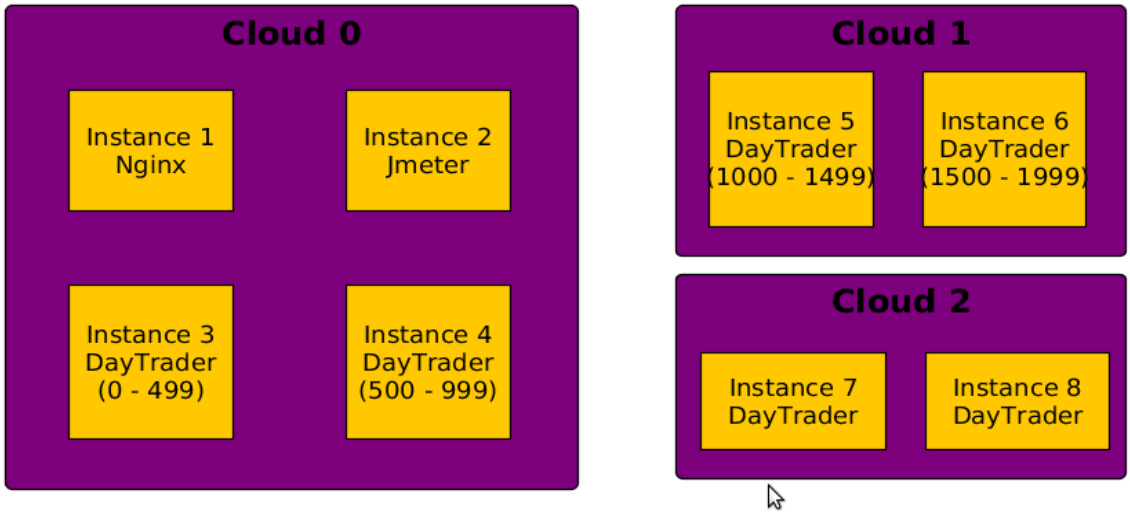

Http Requests
Web browser
Internet
Http Response
Cloud with reverse proxy
Instance 1
Instance 2
Instance 3

Figure 6.4


Cloud 0
Instance 1
Nginx
Instance 2
Jmeter
Instance 3
DayTrader
(0 - 499)
Instance 4
DayTrader
(500 - 999)
Cloud 1
Instance 5
DayTrader
(1000 - 1499)
Instance 6
DayTrader
(1500 - 1999)
Cloud 2
Instance 7
DayTrader
Instance 8
DayTrader