Project 5b
Scalable High Volume Transaction Processing Using Hot-Pluggable Devices

Team Members:

Name	RollNo	Contact	Email
Kanika Khattar	MT2011062	9986768038	kanika.khattar@iiitb.org
Kriti Nasa	MT2011067	8861043170	kriti.nasa@iiitb.org
Paritosh Heera	MT2011099	8277274483	paritosh.heera@iiitb.org
Ramyashri M	MT2011120	9844438020	ramyashri.m@iiitb.org
Sangeeta T S	MT2011136	9980740234	sangeeta.ts@iiitb.org

Team leader: Kanika Khattar

Contents

1 Introduction

High-Volume Transaction Processing

Business Transactions in today's world are very huge and often complex. Almost all organizations have business transactions as one of the critical and core activities. The organizations may incur loss if the transactions are too slow, cannot handle huge data and give wrong results.

The transaction processing system proposed here aims at handling high volume of transactions efficiently. The system deploys a server that will be responsible for balancing the load on the system. The load will be distributed amongst the hot pluggable resources (which can be added or removed without rebooting the system). The resources here will be the real servers which will handle the transactions. The processing speed of the proposed system is scalable, depending on the number of hot-pluggable servers available. The reliability of the system is maintained by continuous monitoring of the hot-pluggable resources.

2 Project Description

2.1 Objective

The project aims at designing a system that can process a very high volume of transactions per second in a way such that the processing speed can scale upwards when more resources are available and gracefully degrade when they are not.

2.2 Project Goal

- To provide a basic framework for building highly scalable and highly available transaction processing services using a cluster of hot pluggable resources.
- Allow processing of large number of transactions per second by distributing the incoming transaction requests to different hot-pluggable resources connected to the server.
- Allow the addition and removal of the hot-pluggable resources to handle scalability.

- Monitor the health of the resources continuously so as to make the system reliable.

3 Gap Analysis

The end users need applications that are fast and easy to use. The existing transaction management systems are not fast enough for high performance situations and under perform when the number of users increases drastically. The adding and configuring of the new resources is not flexible and takes considerable amount of time.

Furthermore, in the existing systems like IBM Websphere, the solution to the system failure is not completely transparent to the end user. If the system fails while processing a user request, the service is interrupted and the clients are intimidated about the failure before redirecting their request; which incurs a considerable overhead.

The proposed system firstly aims at providing persistent connections to the end user despite failures. An efficient algorithm is designed to distribute the load among the available servers to enhance the performance during heavy use periods. The system is further enhanced by enabling the dynamic addition or removal of the resources. To prevent the user from being notified about the failure the system continually monitors the resource's health and takes suitable measures.

In this way the overall utilization of the system is improved. Thus, the proposed system attempts to overcome the shortcomings of the existing system.

4 System Architecture

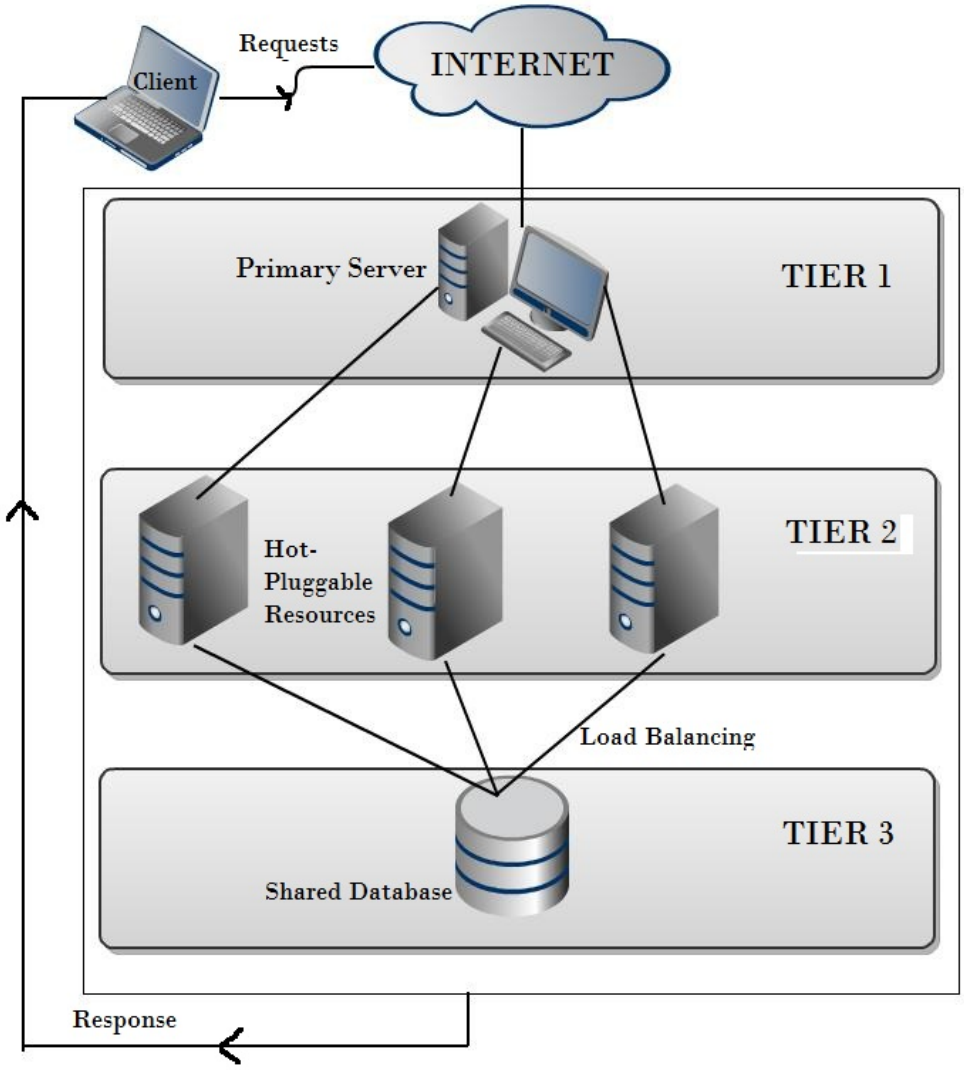

Figure 1: System Architecture

In this section we present system architecture for building high volume transaction processing system. The three-tier architecture of the system illustrated in Figure 1 includes:

- **Tier 1:** It is the topmost level of the system. It includes the primary server which is the front end to the service as seen by the outside

world. The primary server is implemented in the Linux kernel. Its main functionalities are:

1. Selecting a server (from the cluster of hot pluggable resources present at Tier2) for the proccessing of new transaction requests using scheduling algorithms.
2. Forwarding the transaction requests from the clients to the selected server using techniques like IP Tunnelling, NAT, Direct Routing. This will hide the internal architecture of the system from the clients and give them the illusion that they are working with a single server only.
3. Monitoring the health and keeping the count of the hot pluggable resources that are connected to the primary server.

   Hence this tier works as load balancer by distributing the load amongst the hot pluggable resources.

- **Tier 2:** The processing of transaction requests by clients is pulled out from tier 1 to this tier. All the hot pluggable resources which are the actual working servers are present here. The requests that are routed from tier 1 are fully processed in tier2. After successful processing of requests the response is sent back primary server present at tier1.
- **Tier 3:** This tier consists of a shared/centralized database. The entire data that is shared among all the hot pluggable resources resides here. The actual database transaction takes place at this tier.

Illustrating the architecture with an example: If three hot pluggable resources are available and 3000 requests are made to the primary server, then it will forward 1000 requests on each of the available hot pluggable resources. Assuming the current rate of processing of each server is 1000 requests/sec, 1 server alone takes 3 sec to process 3000 requests. With the help of three hot pluggable resources, 3000 requests can be processed in 1 sec which is a major advantage.

5 Development Plan

5.1 Implementation Details

- **Tier 1:** Installation and conguration of Primary Server. The setting up of transaction processing system starts with installation and configuration of primary server. A Linux Virtual Server [1] will be used to set up the same. Following packages will be installed to provide the above said functionalities of tier 1:

  Ipvsadm: The package will be used to select a server from the available hot pluggable resources and forward the transaction request to it. [2] The selection of server will be done with the help of Least Connection Scheduling Algorithm according to which a hot pluggable resource with the least number of active connection will be selected.
  The forwarding of packet to the selected hot pluggable resource will also be handled in this package with the help of layer 4 switching in the Linux Kernel [3].

  ldirectord: A daemon that will be used to monitor the health of hot pluggable resources that are connected to the primary server. This package will be responsible for informing the primary server if any of the hot pluggable resource goes down [4].

  heartbeat: A daemon that will allow primary server to know about the present count of available hot pluggable resources.

- **Tier 2:** Conguration of the secondary servers (hot-pluggable resources). We will be using Apache2 to set up the secondary servers.

- **Tier 3:** A shared database will be used that will be accessed by the hot pluggable resources.

5.2 Stages

- **Stage 1:**
  1. Tier 1 setup: Installation and configuration of primary server using LVS (Linux Virtual Server).
  2. Tier 2 setup: Configuration of secondary servers (hot-pluggable resources) using Apache2.
  3. Tier 3 setup: Creating a shared database.
- **Stage 2:**
  1. Setting up LAN/WAN connection for detecting hot pluggable resources.
  2. Setting and configuring of primary server as load balancer.
- **Stage 3:**
  1. Setting up of the required scheduling algorithm in the primary server to manage the incoming traffic. Implement a Scheduling Algorithm [5] for selecting a particular resource from the available resources for redirecting the transaction request. Scheduling algorithm will be based on number of available hot pluggable resources, active connections on each server and characteristics of the available resources.
  2. Testing of traffic redirection mechanism.
- **Stage 4:**
  1. Generating 10000 transactions for testing of the system.
  2. Updating the scheduling algorithm according to the results obtained.

6 Milestones

Date	Task
January 17th	Background study of the project goal and first draft submission.
January 27th	Setting up of CVS/SVN/Git version control repositories and final draft submission.
February 7th	Setting up of Tier 1, Tier 2 and Tier 3. Implementing detection mechanism of hot-pluggable resources.
March 1st	Implementing load balancing techniques, and a scheduling algorithm at primary server.
April 3rd	Generation of large number transactions and checking of the redirection mechanism.
April 19th	Updating of scheduling algorithm depending on the results obtained.

References

[1] *Linux Virtual Server (LVS) for Red Hat Enterprise Linux*. Red Hat, Inc, 2009.

[2] M. Ghaleb, "*How To Set Up A Loadbalanced High-Availability Apache Cluster Based On Ubuntu 8.04 LTS*," December 2008. [Online].Available: `http://howtoforge.com/set-up-a-loadbalanced-ha-apache-cluster-ubuntu8.04`.

[3] E. Searcy, "*Scalable Linux Clusters with LVS*," April 2008. [Online].Available: `http://linuxcommand.org/man_pages/ipvsadm8.html`.

[4] S. Horman, "*Linux Virtual Server Tutorial*," July 2003. [Online].Available: `http://www.ultramonkey.org/`.

[5] Noveck, "*Choosing the right scheduling algorithm for a Linux Based Load Balancer*," February 2011. [Online].Available: `http://noveckg.blogspot.com/2011/02/choosing-right-sceduling-algorithm-for.html`.