Design and Implementation of a Server Cluster Backend for Thin Client Computing

Ashish Khurange
IIT Bombay
ashishk@it.iitb.ac.in

Om P. Damani
IIT Bombay
damani@it.iitb.ac.in

Abstract

Thin Client systems provide affordable solution in environments where little computation power is needed, such as academic institutes, and small and medium businesses. Existing open source Thin Client systems are however not scalable and have the server as the single point of failure. In this paper, we present the design and implementation of a server cluster backend which makes the Thin Client solution highly available and scalable. We have used LTSP (Linux Terminal Server Project) Thin Clients for our implementation but our design is generic and can be applied to other systems as well. Our design goals are to use open source software and to keep hardware cost low.

1 Introduction

Schools, colleges and small and medium businesses need computers mostly for document processing, Email, and Internet access. They need very little processing power. While PC prices are falling rapidly, many organizations still can not provide separate PC per user. Thin Client system provides an affordable solution in such scenario.

A Thin Client system consists of multiple heterogeneous Thin Client terminals (TC) connected to a single server. Hardware requirement of a Thin Client is keyboard, monitor, mouse, and some computation power. The server performs all application processing and stores all user data. Thin Client plays the role of input and output device only. Figure 1 shows Thin Client Network.

Thin Client sends keyboard and mouse inputs to the server. Server performs the computation accordingly and replies back with the screen updates. Figure 2 shows working of Thin Clients.

Advantages of Thin Client system are :

1 Thin Clients are less expensive and maintenance free. Old heterogeneous PCs can also be used as Thin Client. In our lab at IIT Bombay, 16 Thin Clients are running for over 3 years without any maintenance.

2 Only the server needs to be secured and not the TCs, thus reducing the cost of security.

3 All data is stored on the server. Hence only the server needs to be backed up.

4 Software installation and upgradation happens only on the server. This reduces the cost of software installation.

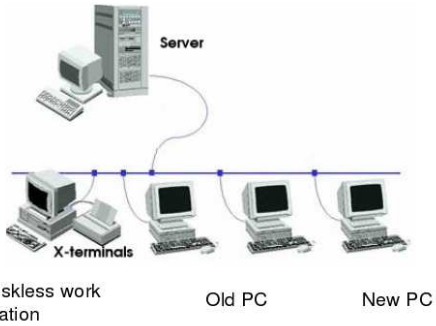

Figure 1. Thin Client Network

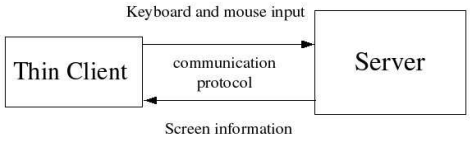

Figure 2. Thin Client computation system

But the current Thin Client systems have following limitations:

1 In current implementation of Thin Client system there is only one server to which all the Thin Clients are connected. If the server is down, none of the Thin Client

can carry on its computation. It has the server as a single point of failure.

2 Since there is only one server, it is not a scalable solution. Few of the schools from UK have reported that they want to support around 1000 Thin Clients [7]. Which is not possible for a single server.

3 Adding more independent servers helps with scalability but not with high availability. Because if a user has his home directory on one server and if it goes down, then that user can not use the computation facility even if other servers are running.

4 Adding more independent servers does not guarantee load balancing and optimal use of resources.

To overcome these limitations, the single server in Thin Client system needs to be replaced by a cluster of servers. To our knowledge, except for Windows Thin Client system, no other system has a clustering solution. Microsoft and Citrix [2] have developed 'Windows Server 2003 Terminal Services' [8] to provide cluster backend. This product has the following features :

1 Thin Client in Windows network are called Windows Based Terminals (WBT). It's cost is in the range of $500 (without monitor). Unlike Unix based systems, any 'dumb terminal' can not act as a Thin Client.

2 For load balancing in this cluster it is recommended to use a third party (Citrix) load balancing solution.

3 All the servers in the cluster are required to be running Windows Server 2003, Enterprise Edition or Windows Server 2003, Datacenter Edition only.

4 This cluster needs a separate highly available file server.

All the above factors increase the cost of Window's solution. We have designed and implemented a server cluster backend for open source Thin Client computing, which provides scalability and high availability. We have used LTSP system [6] but our ideas can be applied to other open source Thin Client systems as well such as PXEs Thin Clients [9]. Our emphasis is on providing high availability at low cost. We are not aiming for transparent failover. When a server fails, all the Thin Client terminals connected to it need to be rebooted.

The rest of the paper proceeds as follows. In section 2, working of Thin Client is discussed. Design issues of cluster are discussed in section 3. In section 4 we look at how, the cluster design provides highly available DHCP, TFTP, XDM and NFS services. In section 5 we describe main software components we developed for building the cluster. In section 6 we describe working of cluster. We discuss high available filesystem for the cluster in section 7. Conclusion is discussed in section 8.

2 Working of Thin Clients

To understand the issues in cluster design first the working of existing Thin Client system needs to be explained. Figure 3 shows stepwise working of the LTSP Thin Client. LTSP needs four basic services - DHCP, TFTP, NFS and X display protocol.

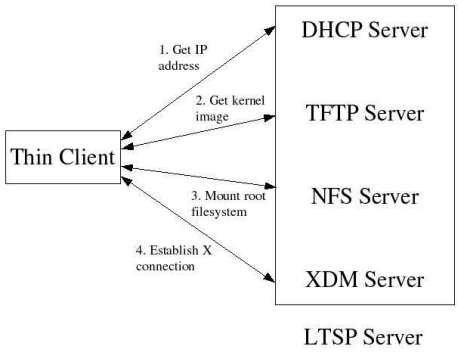

Figure 3. Working of Thin Client

1 DHCP (Dynamic Host Configuration Protocol) : When a Thin Client is turned on, it broadcasts a DHCP request to the local network. DHCP server running on the network replies back with the IP address for the workstation, netmask for the local network, pathname of the kernel to download, and pathname of the root filesystem to mount. The Thin Client receives the reply and it configures its TCP/IP interface.

2 TFTP (Trivial File Transfer Protocol) : Using the the pathname of the kernel to download, provided by the DHCP server, Thin Client contacts the TFTP server, downloads the kernel, and boots up using that kernel.

3 NFS (Network File System) : Using the pathname of the root filesystem to mount, provided by the DHCP server, Thin Client mounts the root filesystem via NFS. The root filesystem has all the configuration files, Xserver binaries, and various scripts required to connect to the server.

4 XDMCP (X Display Manager Control Protocol) : X is the display protocol used in LTSP. The mounted root filesystem has a configuration file, which contains IP address of the XDM server. Depending on its video

card, Thin Client selects the appropriate X server binary. The Thin Client sends an XDMCP query to the XDM server, which offers a login dialog.

3 Design issues

To make Thin Client computing scalable and highly available we need to replace a single server in Thin Client network with server cluster. To make the cluster highly available we need to remove all the single points of failures. Following issues need to be addressed to build scalable and highly available cluster.

1 Provide highly available DHCP, TFTP, NFS, and XDM services.

2 Static binding of Thin Clients to XDM server needs to be made dynamic.

3 To keep the system cost low, we do not want to use any special purpose storage device. Data should be stored on servers only. Each user's home directory must be replicated on multiple servers to provide high availability. Replicated filesystem must be consistent.

4 We need software component for finding health status of servers, load balancing, and managing clusters.

For our experiments we are using LTSP Thin Clients and X protocol for display. Our aim is to do minimum changes to LTSP, so that our changes can be easily incorporated in LTSP. We haven't done any changes to the X protocol. Our design is generic, and can be used with any Thin Client and any display protocol. In the following sections we explain how our system addresses above issues.

4 High availability of various services

In this section, we discuss, how to provide highly available DHCP, TFTP, NFS and XDM services.

1 **DHCP Server** : In DHCP Protocol [10], the period over which a network address is allocated to a client is referred to as a 'lease'. The DHCP client can extend its lease from the DHCP server which initially assigned it IP address or another DHCP server which has the information about this client's lease. Failing the lease renewal, the client reinitiates the process of getting an IP address. If client gets the previous IP address back, it continues network processing, else it drops all existing network connections.

Breaking existing network connections will cause a Thin Client to terminate its X session and disrupt the current user. Note that Thin Client network is a stable netwok - clients are not added or removed frequently. Hence unlike existing usage of DHCP, we use static IP allocation feature of DHCP. In our design, there is static binding of MAC address of a client to its assigned IP address. This static binding is same for all DHCP servers and hence they return the same IP address to a particular Thin Client. In this arrangement even if the DHCP server which assigned an IP address to a client goes down, on lease expiration, the client can get its original address back from other DHCP servers. If we arrange n servers in 'Static binding', we can tolerate upto $n - 1$ failures. In contrast, the DHCP 'Failover protocol' [5] arrangement can tolerate failure of only one DHCP server.

2 **TFTP Server** : Thin Client contacts TFTP server to download kernel. TFTP servers do not have any state like DHCP. Hence multiple TFTP servers working independently provide high availability. In our design, TFTP server runs on each of the servers where DHCP server is running. Each DHCP server, in its reply to client, advertises its own address as path to download the kernel.

3 **XDM Server** : In our solution, each server acts as XDM server. Each XDM server offers X session to only those users who have their home directories on that XDM server.

Now we discuss, in our cluster design how a Thin Client selects a XDM server from the cluster for its session. After replacing a single server in Thin Client network with server cluster, static binding between Thin Clients and XDM server is not going to work.

Our cluster design makes this binding dynamic using username of the user who wants to use the Thin Client. Each time before establishing X connection with XDM server, Thin Client prompts for username. Thin Client gets the username from user and sends the username to the server acting as Load balancer in the cluster. Load balancer selects the least loaded server which hosts this user's home directory, and replies Thin Client with IP address of that server. Then the Thin Client establishes X connection with the selected server, and the user gets log-in to the server which hosts his home directory. Thus binding between Thin Client and the server is made dynamic using the user id.

4 **NFS Server** : Thin Client uses NFS server to mount the root filesystem. Multiple NFS servers working independently can provide high availability. When Thin Client establishes a X connection with a XDM server, it has two independent dependencies. First is the NFS server from which Thin Client mounted its root file

system. It contains Xserver binaries and all configuration files, which are necessary for running X session. Second is the XDM server where user logs in. To reduce the impact of failures, we merged these two separate dependencies into a single dependency by remounting the root file system from the XDM server itself. Hence alongwith XDM server, we run NFS server on every machine in the cluster.

Figure 4 shows merging of two separate dependencies.

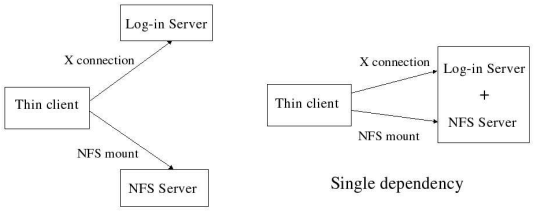

Figure 4. Merging two separate dependencies.

5 Software components of the system

In the last section we saw how our cluster design removes all single point of failures. We next discuss the software component which schedules user's session to a server and manages the cluster.

1 **Health Status Service** : This component computes load on all the running servers in the cluster. It is divided into two parts : Health Status Server and Health Status Client. Health Status server runs on each of the server in the cluster. It accepts an incoming connection and replies back with its load. Health status client is used by Load balancer to detect whether server is alive or dead, and to get the load on the running server.

2 **Load balancer** : Load balancer ensures that a user logs into the least loaded server that hosts this user's home directory. It has a database which maps username to server group which hosts home directory for this user. Using Health Status client, Load balancer finds load on these servers and returns the IP address of the least loaded server to the Thin Client.

Load balancer is also a single point of failure, so multiple servers in the cluster run Load balancer. These Load balancers function independently. They only need to synchronize the database mapping username to servers. This mapping changes only when the cluster configuration changes or the number of users changes drastically. As presented, our cluster design has two separate dependencies; DHCP servers and Load balancer. To reduce impact of failure, we merge these two separate dependencies, by running these two components together on same servers in the cluster.

3 **Cluster Manager** : This component is used by system administrators to manage the cluster. It is used to add or remove a server to cluster transparently without bringing the system down. It provides services to start or stop load balancer on any of the server in the cluster. It also provides a service to add or remove users from the system. The Cluster Manager takes care that each of the server in the cluster has the latest information about the entire system.

6 Working of the system

We have already explained how the cluster design provides highly available DHCP, TFTP, NFS and XDM service. We also discussed the main software components of our system. Putting these two together, we next present the working of the entire system.

Arrangement of various components in the cluster is as follows. Each server in the cluster runs XDM server, NFS server, and Health Status server. DHCP server runs on some of the servers in the cluster providing high availability. On these servers along with DHCP server, TFTP server and Load balancer also run. The number of servers on which DHCP, TFTP and Load balancer services run, determines the fault tolerance capacity of the cluster.

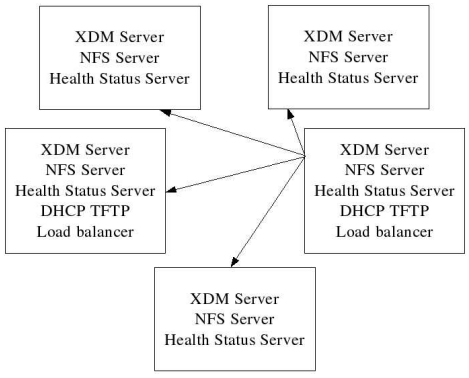

Figure 5. Arrangement of various components in cluster

Figure 5 shows design of a five servers cluster, with a fault tolerance of two. Figure also shows, Load balancer module of a node contacting all other nodes to find out load on them.

When a Thin Client is turned on, it broadcasts a DHCP request. All DHCP servers reply back offering an IP address for the Thin Client, and advertising their own address as TFTP server and NFS server. Thin Client randomly selects one of the replies. After configuring its TCP/IP interface, the Thin Client contacts the TFTP server (running on the same machine as the DHCP server whose reply the Thin Client chose) and downloads the kernel. After booting the kernel, Thin Client mounts the root filesystem via NFS from the NFS server. This NFS server also runs on the same machine as the DHCP server whose reply the Thin Client chose.

Figure 6 explains the working of the system after the Thin Client has mounted the root filesystem:

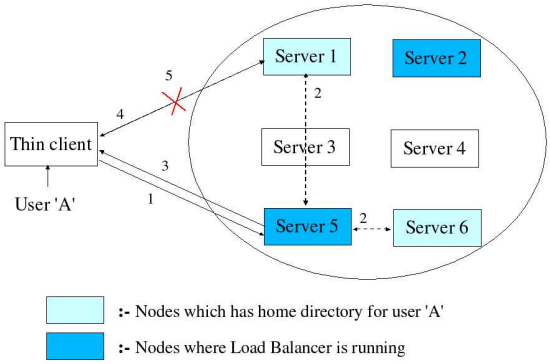

Figure 6. Summary of working of cluster

1 In original system, once Thin Client boots, it sends XDMCP query to the XDM server. We need to select the XDM server based on the user id. Hence we have changed this step. At this point, the Thin Client prompts for the username. The mounted root filesystem of Thin Client has a configuration file which lists IP addresses of all the Load balancers in the cluster. Thin Client gets the username from the user and sends it to one of the running Load balancer. In case the Load balancer does not reply in a specified time interval, the Thin Client tries the next Load balancer in its list.

2 Each Load balancer has a database mapping username to XDM servers. From this database, Load balancer finds out the server group hosting the home directory of the given user. Using Health Status Service, Load balancer finds out the least loaded server from this group.

3 Load balancer sends address of this least loaded server to the Thin Client.

4 Thin Client sends XDMCP query to the selected XDM server and establishes X connection with the server.

5 In the original system, when the user logs out, the X connection between Thin Client and the XDM server remains persistent. In our system, we do not know who the next user is and whether they have home directory on this server or not. Hence when a user logs out, the X connection between Thin Client and XDM server is broken and the control is returned back to Thin Client, which again prompts for the username. Steps 1 to 5 are repeated for the next user.

Now we list all the changes we made to LTSP Thin Client system, to build this cluster.

- In the existing LTSP system, Thin Client's binding with the XDM server is static. Every time Thin Client establishes connection to the same server, whose IP address is mentioned in the configuration file. We made Thin Client's binding with XDM server dynamic, depending on username. User is first prompted for username. Then X connection is established with the server hosting user's home directory, selected by Load balancer.

- The XDMCP query to establish X connection with XDM server, is sent with -terminate option, which breaks the X connection between Thin Client and XDM server after the user logs out.

These are the only changes we introduced in to LTSP Thin Client system. Hence our changes can be easily incorporated in existing LTSP Thin Client system.

7 Filesystem for Cluster

The filesystem of our cluster is based on High Available NFS (HA NFS) [1]. HA NFS is designed using DRBD (Distributed Replicated Block Device) [3] and Heartbeat [4].

DRBD is a kernel module that maintains a real time mirror of a local block devices on a remote machine. In DRBD, pair of nodes work together. One of them acts as 'primary' and the other as 'secondary' server. The block device being replicated, is accessible on primary node only.

Heartbeat provides a cluster membership service, which detects node liveliness. Heartbeat runs between the pair of nodes running DRBD. All applications accessing the replicated block device run on the primary node. If the primary

node fails, Heartbeat makes the secondary node primary. If the failed node comes up again, it is made a secondary node and it has to synchronise its content to the current primary.

7.1 HA NFS

User's home directories can be mirrored in real time using DRBD. Thus we can provide highly available filesystem to users. But DRBD grants read-write access to the primary node only. Which means users who gets log-in on primary server can only access their files. To allow users to acess their files on the secondary server, we have implemented HA NFS. We export partition which has users home directories through NFS, so that it can be mounted on both the nodes. We create a *virtual* NFS server, which points to DRBD's primary server. This virtual IP will be used by the current 'primary' DRBD node. Both the nodes mount the home directory from this virtual server.

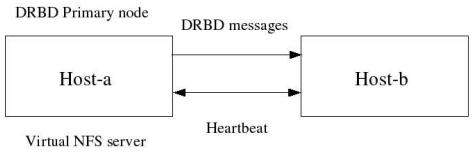

Figure 7. Working of HA-NFS

Figure 7 shows working of HA-NFS, when both the nodes are running. Host-a is DRBD primary node.

When both the nodes are running, user's home directories are mirrored using DRBD. The primary DRBD node uses the virtual IP address and it exports user's home directories through NFS. Both the nodes mount user's home directory from this virtual server. DRBD running between the nodes does real time mirroring of filesystem. Heartbeat running between the hosts does detection of node failure.

When the primary node fails, Heartbeat daemon running on secondary finds out primary is down. Secondary takes over virtual IP address and starts NFS server. All the users sessions, who got their log-in on primary are terminated. Since NFS can recover from disconnection, all the users who got their log-in on on secondary continue without interruption.

Users in the system are divided into mutually exclusive groups. Then home directories of users in each group are replicated between a separate pair of servers. Users get log-in to any one of the two servers where their home directory resides. Thus HA NFS provides high available file system for this cluster. Our solution can tolerate failure of one server for each user group.

8 Conclusion

We have implemented the server cluster backend for Thin Client computing. In our cluster design DHCP, TFTP, and Load balancer can tolerate up to $n - 1$ failures. Users in the system are divided into mutually exclusive groups. Filesystem provides fault tolerance of one server for each user group. Thus we have provided high availability.

Our cluster has four servers and 16 Thin Client terminals. Right now, we use HA-NFS for cluster filesystem. Our solution can tolerate failure of one server per user group. We want to make our filesystem solution scalable. For this purpose, in future, we plan to explore the use of Coda filesystem [11] as the cluster filesystem and compare its performance to HA-NFS.

References

[1] Article on highly available nfs (ha nfs), http://wiki.linux-ha.org/drbd/nfs.

[2] Citrix homepage, http://www.citrix.com.

[3] Drbd homepage, http://www.drbd.org.

[4] Heartbeat project, http://linux-ha.org/heartbeat.

[5] Internet draft dhcp failover protocol, http://www3.ietf.org/proceedings/04mar/i-d/draft-ietf-dhc-failover-12.txt.

[6] Linux terminal server project homepage, http://www.ltsp.org.

[7] Ltsp discuss mailing list, http://www.ltsp.org/mailinglists.php.

[8] Microssoft windows terminal server 2003, http://www.microsoft.com/windowsserver2003.

[9] Pxe thin clients, http://pxes.sourceforge.net/.

[10] Rfc 2131 dhcp protocol, http://www.freesoft.org/cie/rfc/2131.

[11] Satyanarayanan M., Kistler J.J. Kumar P., Okasaki M.E., Siegel E.H., and Steere D.C. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers Vol. 39, No. 4*, April 1990.