High Availability Using Virtualization

Amartya Dasgupta
IIIT - Bangalore
Bangalore, India

Syed Eqbal Alam
Taif University
Taif, Saudi Arabia

Arijit Mitra, Salone Gupta, Sneha Joseph, Shrisha Rao
IIIT - Bangalore
Bangalore, India

Abstract

We describe the design of an inexpensive high availability solution that does not require the use of additional hardware. This allows us to achieve complete fault tolerance on multiple servers including application servers as well as DNS servers using virtualization. Our architecture provides instant backup to the client at run-time with minimal data loss by using hot-standby backup servers. The arrangement uses multiple virtual servers connected to a DNS terminal server acting as the controller. The system can scale easily by connecting more secondary servers to the terminal server. With this architecture, a backup virtual server comes into action as soon as the primary server goes down, and will continue the operations that the main server was running, exactly from the point of failure, with minimal loss of data.

I. Introduction

We describe a software system that provides system level high availability with the help of virtual servers. Our approach exploits the ability of virtualization to migrate running virtual machines. It extends the technique to replicate snapshots of an entire OS or partial instance between primary and secondary servers. Using this technique, our system provides minimal data loss while migrating from primary to secondary server by starting execution of the interrupted service from the point of failure.

By allowing multiple servers to be consolidated on a smaller number of physical hosts using virtual servers, it is easy to decrease the cost of the overall system setup. But this benefit comes with a hidden cost in the form of increased hardware failure. Our system architecture provides an efficient solution to this problem by providing high availability to the client using these virtual servers. Also by using virtualization, the system is independent of underlying hardware resources, thus making it easier to transfer the virtual operating system from one physical resource to another without hardware complications.

This system allows a host to execute speculatively and then to checkpoint as well as replicate its state at a very high speed.

[1]The work was done while the second author was a student at IIIT Bangalore.

This architecture can be described in terms of the following salient features:

- A “heartbeat” mechanism for health checking of application servers. In case of a primary server failure, the secondary server is promoted to act as the new primary server (failover).
- Server mirroring, which efficiently helps in backing up data in a faster and secure way.
- A terminal server which directs the client request to a secondary server in case of a failure of the primary server. It also minimizes the overhead on the application servers. There is no separate program running on the primary application server for the heartbeat mechanism or server mirroring purposes.

Propagating the whole system state each time from the primary to the secondary server is impractical as it decreases the efficiency of the system as well as taking a huge amount of time and bandwidth to transfer large amounts of data across the network. So we use delta encoding [1] technique for mirroring of server data which minimizes data transfer whenever possible. This keeps the process running for the clients without any discontinuity under an idealistic case; practically there is a very small processing delay time, which may cause a small discontinuity experienced by the end users. This system ensures that regardless of the moment at which the primary fails, no externally visible state is ever lost. In this architecture, the heartbeat mechanism is server independent. It checks the heartbeat of the application servers on an echo port, thus reducing the overhead on the application servers, as no separate program has to be initiated on them. The system is also very cost effective as all the tools involved are open source tools. It does not require any external hardware or specialized commercial software for its implementation.

The advantages of the proposed system are as follows:

- Low overhead on primary servers as bulk of the applications are distributed among terminal server (Section III) and secondary servers.
- Heartbeat mechanism as explained in Section III is independent of the application servers. As this mechanism is being executed using an echo port, the overhead on the application servers is low.
- For server mirroring, explained in Section III, the

open source tool rsync is used. This tool uses delta encoding. In this technique, instead of the total instance of data, only the minimal changed data gets transferred, thus reducing the required time and bandwidth for data transfer and making it faster and maintaining the data integrity.

- The transfer of data is over a secured remote shell SSH or RSH, which ensures the security of data.

The rest of the paper is as follows. Section II talks about various works done in the past that are related to this project. Section III describes the architecture and working of our high availability framework and the scalability issues. In Section IV, an evaluation of the system is presented. Section VI concludes the paper.

II. Related Works

There has been a lot of work for the past few decades to provide high availability in the presence of failures. Zhiming, Xiaohua and Jichang et. al [2] have designed an adaptive heartbeat fault detection model of dual controller, which can adjust heartbeat cycle based on the frequency of data read-write request that can improve the high availability of dual-controller RAID storage system. In [3] feasibility of deploying IP communications infrastructure on virtualized platforms, with high-availability and reliability is studied. Another approach of high availability is in-memory replication [4] which is based on creating a checkpoint of current state of the whole server and then transferring it to another server. Socket level recovery [5] is similar to the previous approach except that it uses an application-level protocol which is built on both the client system as well as the server system for the purpose of client reconnection. Distributed object middleware [6], [7] is basically implemented in distributed systems where all the data from one server is sent to a middleware server which acts as a backup server for the whole system. This middleware server can be a single physical server or a set of them. Another approach is transport-layer load balancing [8] which provides fault tolerance at the transport layer level. In case of primary-backup systems [9], [10] the way fault-tolerance is provided is as follows. The state of the currently active server is replicated to one or more backup servers. The client is connected to the primary server. The backup servers check the health of the primary server and in case of any failure, one of the backup servers is promoted as the primary server. But in this case, the backup servers have to run a health check program continuously to sense the failure of the primary server. This is a big overhead on the application servers.

Our approach is based on system level recovery. Under this approach, the user remains unaware of any event of failure, as the terminal server automatically redirects the client to backup servers. As for example an FTP client that restarts automatically in case of aborted transfers. Other examples in this category are Samba [11] and NFS clients [12] which have the ability to recover from short connection failures transparently. But in case of already-running applications, this approach is usually not applicable. One of the approaches is to use socket-level recovery [13], [14], where the lower layers provide a reliable socket to reconnect the broken connection even on failure. Some recent progress can be checked in the area of high availability in the papers [15], [16], [17], [18].

III. Design and Implementation

This section presents the design and implementation of our prototype system. Figure 1 explains the basic architectural design of the system. The figure is only representative, because though the system is scalable and multiple servers can be put into the server pool only two servers are shown here for simplicity of understanding the implementation technique. Later on, we will describe how this can be used for multiple servers.

As per our model, the system has three or more physical resources (machines) depending on the scale of the implementation, where the terminal server is on one physical resource and the array of application servers is distributed over the remaining physical resources. Thus, if one of the physical resources fails we still have backup application servers on the other physical resource. This retains high availability in case of hardware malfunction. Since our application servers are virtual operating systems (We have used VMWare as virtualization software) which reside on the same physical resource, the transfer of data between these servers is much faster as compared to data transfer between two servers across the physical external network. Thus, the system will have minimal data loss and faster expected response times as compared to a system with separate physical systems with no virtualization.

In Figure 1 we can see that there are three servers, terminal, primary and secondary servers. Here the primary and the secondary servers are virtual servers present on multiple physical hosts. The terminal server can also be implemented on separate physical host so as to retain high-availability even on the occurence of hardware malfunction. A client machine connects to the terminal server and in turn the terminal server directs the client to the application servers. The total design implementation is explained in some steps. The steps are being marked over the arrows denoting the flow of the process.

Terminal Server Significance and function

The first step in Figure 1 illustrates how a terminal server works. The terminal server is shown subdivided into

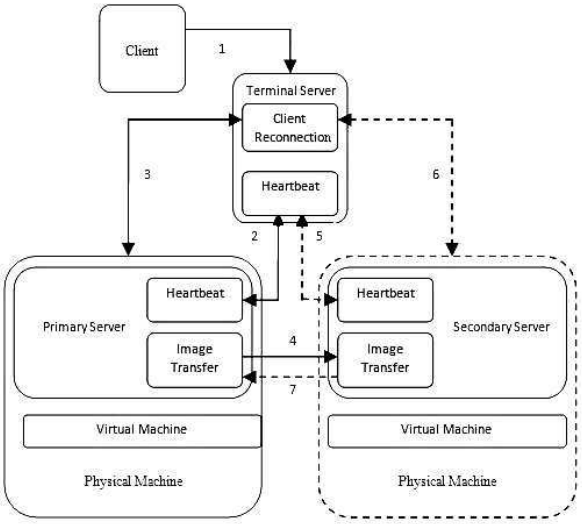

Fig. 1. Basic Architecture—Representative

two modules, one being the client-reconnection module and the other being the heartbeat module. The terminal server is connected to the primary and secondary servers. When the client requests URL of our domain on his browser, the terminal server will connect the client to one of the primary servers during normal execution. In case of failure it will promote a secondary server as primary server and connect the client to that server instead.

Heartbeat and Client Reconnection

Significance of Heartbeat mechanism: Now in step 2 of Figure 1, the heartbeat module comes into play. The terminal server checks the health of the servers from the server pool. For checking the health of the application servers, i.e., checking which server is alive, the terminal server pings the servers in the pool using an echo port. All servers which are in an active state reply with an ack to the terminal server. Step 3 in Figure 1 concerns the client reconnection. During normal execution primary servers and secondary servers are connected to the terminal server. If the terminal server finds that a primary server is down, then it will promote a secondary server as a new primary server and start the execution from the point of failure.

Client reconnection significance: After getting back the reply from a primary server, the terminal server directs the client to this server, which in turn starts to execute the client requests. At this point, all the other servers that are alive but not taking part in any interaction with the client become secondary servers. So we see that the terminal server is responsible for connecting the client with the primary server through this client reconnection technique.

Image Transfer or Server Mirroring

Now when the primary server is identified and has already started working with the client, and issue that has to be taken into account now is to take the backup of the server data to make sure that no data is lost in case of failure. So, in Figure 1 we see that the data transfer module comes into the picture in step 4.

Significance of image transferring: The secondary servers get the backup at regular intervals from the primary server. In our implementation, we use the open source tool Rsync as mentioned earlier to transfer the image. Rsync uses delta encoding to minimize the data transfer between primary and secondary servers. Using this technique, a primary server sends only that data which is not present on the secondary server. This reduces the amount of time as well as the network bandwidth required for data transfer.

If a failure occurs in the primary server and it stops working, then very shortly thereafter, the heartbeat mechanism of the terminal server is able to check its status. Since the primary server does not respond to the pinging of the terminal server, the latter now promotes any other server in the server pool which is alive in round-robin order. In step 5, we see the terminal server doing the same thing as done in step 2 to check which server is alive on the wakeup of a failure. When it gets a reply from a server, it then redirects the client to that server as it has been doing in step 3.

Now the secondary server chosen by the terminal server takes its position as the new primary server. Though it is a new primary server, the client remains unaware of this fact. The new primary server has the necessary backups from the previous primary server by means of the image transfers, and starts the job from the exact point where the primary server has left off, with a nminimal loss of data. This is the illustration of step 6.

We assume in Figure 1 that the server which went down is now active but is not interacting with the client. So this server will now act as a secondary server and will get the backup from the primary server to maintain the consistency and integrity of the data by making sure no loss of data ever happens due to any failure. This is also the illustration of step 7.

IV. Performance Analysis

This section discusses the feasibility and performance of our implementation. We have tested our system using two applications:

1) File download application
2) File upload application

Both applications leverage the high-availability characteristics of the system. We compare our system perfor-

Data Transfer Time - Ours (in secs)			
File Size	No Fault	1 Fault	2 Faults
1Mb data	2.5	27.1	55.7
5Mb data	12.0	37.0	64.3
20Mb data	40.2	64.8	92.7
100Mb data	196.0	222.4	248.5

TABLE I
DOWNLOAD TIME WITH MAXIMUM OF TWO FAULTS

Data Transfer Time - Ours (in secs)			
File Size	No Fault	1 Fault	2 Faults
1Mb data	3.5	27.5	56.7
5Mb data	13.5	37.6	66.1
20Mb data	43.1	67.0	96.1
100Mb data	200.0	224.4	253.7

TABLE II
UPLOAD TIMES WITH MAXIMUM OF TWO FAULTS

mance with the performance data of Marwah, Mishra, and Fetzer [19].

Download Application

In this application client downloads a data file from the server. Initially the terminal server connects the client to the primary server and start the file transfer. In the background, Rsync is used to start sending chunks of the data to the secondary servers. In case of failure, the heartbeat module will promote one of the secondary servers as a primary server and the client will be connected to that server, but this process takes some amount of time to actually shift from one server to another.

We have tested our system for various files without fault as well as with multiple faults. A brief description of the performance is given below in Table I.

Upload Application

In this application, a client uploads a data file to the server. Initially, the terminal server connects the client to the primary server and starts the file transfer. In the background Rsync start sending chunks of the data to the secondary servers. In case of failure, the heartbeat module will promote one the secondary servers as a primary server and the client will be connected to that server. We have tested our system for various files without fault as well as with multiple faults. In Table II a brief description of the results have been shown.

Failover Time — Marwah et al. (in secs)	
File Size	One Fault
1Mb data	22.58
5Mb data	24.01
20Mb data	20.80
100Mb data	21.76

TABLE III
PERFORMANCE ANALYSIS DATA FROM [19]

We have compared our performance analysis data with the data available from [19]. The reason being, in the referenced paper the size of data being experimented is similar to our size of data. In Table III a description of this external data is shown.

The comparison generates a graph similar to the Figure 2.

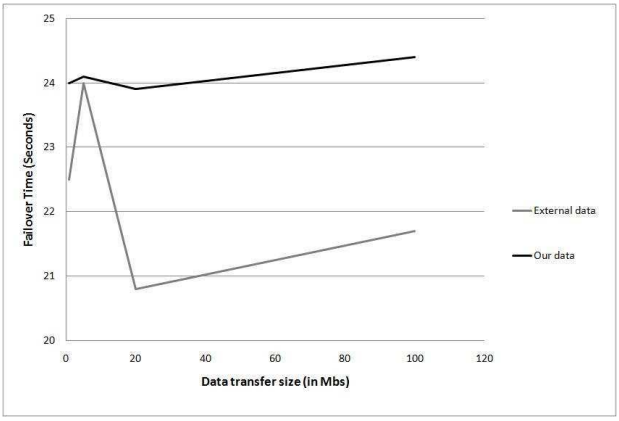

Fig. 2. Performance analysis graph for uploading application and data available from [19]

As per the comparison we can say that our system has relatively small fluctuations in failover time, i.e., it is closer to being constant and predictable. Also, our system is not dependent on the size of the files being transferred as is the case elsewhere [19]. The data collected by testing also shows the efficiency of the system in case of multiple failures because the failover time is almost constant during the server failure. We can see as in Figure 2, that the failover time period is about 5–15% higher as compared to the available data from [19], mainly due to the DNS IP redirection delay. This is unlikely to be significant in practical systems, but can be reduced by utilizing high end physical servers.

V. Scalability Issues

In the basic system of Figure 1, we have a single point of failure in the form of the terminal server. To overcome this problem we can introduce a backup for the terminal server—a secondary terminal server. Since the terminal server is a DNS server that reconnects the client depending upon the heartbeat of the active application server, the backup terminal server would do exactly the same job as the primary DNS server. Both the terminal servers work simultaneously with the application servers. Whenever the primary terminal server fails, the backup terminal server takes over the client reconnection operation, provided that the client or authoritative DNS has the IP addresses of the primary and secondary terminal servers. In other words, if the client fails to get a response to the query made to the primary terminal server, then it automatically tries again and queries the backup terminal server, thus avoiding the single point of failure. The advantage of this kind of setup is that there is no need of a program so as to check whether the primary terminal server is alive or not. The concept can of course also be extended for multiple (more than two) terminal servers as well.

In case of multiple active application servers the system can be extended where a single load-balanced domain or multiple application domain system, i.e., servers with different applications or services can be established. Considering a load-balanced system in a single domain where multiple application server represent the same service, the server mirroring not only takes place on the backup servers but also within the primary servers to mantain data consistency. For the multiple application domain system, where each active server supports a separate service, the data from all the active servers will be stored in all the backup servers such that any backup server can replace any failed active server in a round robin fashion.

In our system, the backup servers can be added seamlessly without affecting the ongoing system operations. Here the new backup server will get added to the end of the list of the backup servers and will come into play as per the round robin fashion.

VI. Conclusion

In this paper we have introduced a framework for high availability using virtualization. The proposed solution keeps the failover time low as well as independent of the size of the data transferred across the servers. Thus the data loss is minimal. Currently the system has been tested using multiple application servers with one terminal server for a single client. It has also been tested on client-server data uploading and downloading services for bulk data transfers. It has provided near-zero data loss achieving our minimal data loss as well as high availability goals. As this system is efficient in utilizing network bandwidth with a high rate of data transfer, we have also shown how this system can scale to a large number of servers. The experience is seamless to a client, which make this platform viable for massive applications also. The client needs an application that is capable of automatically reconnecting to an application server upon failure, and our system is then able to take care of the rest.

References

[1] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for http," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 4, pp. 181–194, 1997.

[2] L. Zhiming, Y. Xiaohua, S. Jichang, and W. Yaping, "Fault detection for high availability raid system," in *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on*, pp. 27–32, aug. 2010.

[3] D. Patnaik, A. Bijlani, and V. Singh, "Towards high-availability for ip telephony using virtual machines," in *Internet Multimedia Services Architecture and Application(IMSAA), 2010 IEEE 4th International Conference on*, pp. 1–6, dec. 2010.

[4] A. Kantee, "Using application-driven checkpointing for hot spare high availability," in *Proceedings of the 3rd EuroBSDCon*, Oct. 2004.

[5] A. Cassen, "Processor allocation and checkpoint interval selection in cluster computing systems," *Linux Virtual Server High Availability using VRRPv2 Linux Virtual Server OpenSource Project*.

[6] C. S. Inc, "Management center for cisco security agents high availability a white paper published," Feb. 2009.

[7] A. Hirt, *Microsoft SQL Server 2005 High Availability*. Berkely, CA, USA: Apress, 2007.

[8] A. Cassen, "Ipvs syncd strong authentication extension linux virtual server opensource project," Mar. 1998.

[9] N. Budhiraja and K. Marzullo, "Highly-available services using the primary-backup approach," in *Management of Replicated Data, 1992., Second Workshop*, (Monterey, CA, USA), Nov 1992.

[10] H. Zou and F. Jahanian, "Real-time primary-backup (rtpb) replication with temporal consistency guarantees," in *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, (Washington, DC, USA), p. 48, IEEE Computer Society, 1998.

[11] W. Fischer and C. Mitasch, "High availability clustering of virtual machines -possibilities and pitfalls," in *Paper for the talk at the 12th Linuxtag*, vol. Version 1.01, (Wiesbaden, Germany), pp. 97–108, May 2006.

[12] V. S. Inc, "An introduction to system high availability, a white paper," 2009.

[13] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan, "Fine-grained failover using connection migration," in *USITS'01: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2001.

[14] F. Sultan, K. Srinivasan, and L. Iftode, "Transport layer support for highly-available network services," in *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, (Washington, DC, USA), p. 182, IEEE Computer Society, 2001.

[15] E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pp. 700–709, june 2011.

[16] M. Handa and Z. Hui, "Construction of high-availability teaching website," in *Management and Service Science (MASS), 2010 International Conference on*, pp. 1–4, aug. 2010.

[17] L. Perkov, N. Pavkovic, and J. Petrovic, “High-availability using open source software,” in *MIPRO, 2011 Proceedings of the 34th International Convention*, pp. 167–170, may 2011.

[18] R. Pinto, J. Rufino, and C. Almeida, “High availability in controller area networks,” in *EUROCON - International Conference on Computer as a Tool (EUROCON), 2011 IEEE*, pp. 1–4, april 2011.

[19] M. Marwah, S. Mishra, and C. Fetzer, “Tcp server fault tolerance using connection migration to a backup server,” in *This paper appears in the Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2003.*, (Los Alamitos, CA, USA), IEEE Computer Society, 2003.