

PLANNING FOR LOAD BALANCING IN V.M. ADAPTING THE CHANGES IN LOAD DYNAMICALLY USING CENTRAL JOB DISPATCHER¹

Rajesh Kumar Pathak, Dr Setu Kumar Chaturvedi

Department of computer Science, TIT Bhopal

ABSTRACT

With the advent of World Wide Web, the life of every person has changed drastically. No one can imagine a computer without being connected to the Internet. This dependency is likely to grow in coming years with the adoption of technologies like virtualization and cloud computing. Cloud Computing and Virtualization is one of the foremost technology which has attracted many researchers recently, which is directly going to benefit the end users and data center service providers with the help of virtualization technique, we can run multiple instances of different operating systems simultaneously.

Since, multiple OS are running on a single physical server, and multiple servers are running in the data centre. all connected via high speed network links. At some instance of time, one server may become overloaded, while other server may remain underutilized. This again poses challenge to distribute the load and make things work perfect in this situation. This situation can be handled using load balancing mechanism over the virtual machines.

This paper work tries to find a mechanism to balance the load based on computational time parameter of the virtual machines. We have tried and implemented the existing approaches of load balancing of jobs for balancing server loads by migrating guests to lightly loaded server. The loads of the nodes in distributed system are subject to change from time to time; thus it is important for load balancing policies to adapt to the changing load quickly. Our goal is to make the best use of the global state information in order to achieve the best performance without incurring too much overhead compared to simple distributed policies.

We propose a load-balancing policy with a central job dispatcher called “Central Load Balancing Policy for Virtual Machines (CLBVM)”, which makes load-balancing decisions based on global state information. This policy has centralized information and location rules. The transfer rule is partially distributed and partially centralized.

INTRODUCTION

A distributed system is considered as a collection of autonomous systems(nodes) located at possibly different sites and connected by a communication network. Through the communication network, resources of the system can be shared by users at different

¹ How to cite the article:

Pathak K.R., Planning for Load Balancing in V.M. Adapting the Changes in Load Dynamically Using Central Job Dispatcher, *International Journal of Advances in Engineering Research*, April 2013, Vol 3, Issue 4, 15-32

locations. With the help of virtualization technology, we are able to run multiple virtual machines or guests on a single physical hardware, and with the use of distributed computing all servers can be connected hosting several guests. The purpose of such a system is not to provide centralized control over all units of the system; rather it is to provide mechanisms which allow the otherwise autonomous processors to cooperate in a consistent and stable manner. One of the most important mechanisms which a distributed operating system can provide is the ability to move a process from one processor to another. This mechanism is called process migration.

Migration of Virtual Machines

If we talk in terms of guest OS, the most important mechanisms which virtualization supports in distributed system is that it provides the ability to move a guest from one system to another. This mechanism is called the migration of operating systems. Migrating operating system instances across distinct physical hosts is a useful tool for administrators of data centers and clusters. It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance. By carrying out the majority of migration while operating systems continue to run, it achieves impressive performance with minimal service downtimes; Migrating an entire operating system and all of its applications as one unit allows us to avoid many of the difficulties faced by process-level migration approaches. With virtual machine migration, on the other hand, the original host may be decommissioned once migration has completed. This is particularly valuable when migration is occurring in order to allow maintenance of the original host. Detailed approaches adopted for live migration of guests in Xen can be found in C. Clark et. al [1].

Migration of operating systems can be done by pausing the system and then migrating it to another system and resume the guest. It can also be done on the fly, i.e. using live migration of operating systems over in a distributed systems environment over a LAN.

Performance enhancement is one of the most important issues in virtualization based distributed systems. Obvious but expensive ways of achieving this goal are to increase the capacity of the participating nodes i.e. the host servers and to add more nodes i.e. guests to the system. Adding more nodes or increasing the capacity of some of the nodes may be required in the cases in which all of the nodes in the system are overloaded; however, in many situations, poor performance is due to uneven load distribution throughout the system. The performance of the system can often be improved to an acceptable level simply by redistributing the load among the nodes. Therefore, load redistribution is a cost effective way

for the reliability and performance improvement of the overall system. This kind of problem is categorized as load balancing or load sharing [5].

LOAD BALANCING

For parallel applications, load balancing attempts to distribute the computation load across multiple processors or machines as evenly as possible with objective to improve performance. A load balancing scheme consists of three phases: information collection, decision making and data migration. The jobs being carried out in distributed environments, require load balancing of the jobs as it may compute intensive tasks and need long running time. There are many issues related to job migration and has been studied extensively in distributed environments. Many load balancing algorithms has been in the market which has solved most of the issues related to distributed computing environment. Few of the work done in this area include policies in which each node periodically broadcasts its status (light-loaded or heavy-loaded) to all other nodes by Kunz [4]. L. M. Ni [7] used three values (Heavy, Normal, and Light) to represent the load status of a node. Zhou [8] and Eager [2] demonstrated that algorithms using a threshold policy generally achieved better performance. Lin and Raghavendra [5] proposed a load balancing mechanism called “Dynamic Load Balancing Policy with a Central Job Dispatcher(LBC)”.

A. Need for Load Balancing in Cloud Computing

Load balancing is the process by which inbound internet protocol traffic can be distributed across multiple servers. Load balancing enhances the performance of the servers, leads to their optimal utilization and ensures that no single server is overwhelmed. The term load balancing is used if the goal is to equalize certain performance measures such as the percentage of server idle times, marginal job response times, etc. On the other hand, if the objective is to improve some performance measure such as average job response time by redistributing the workload, it is called load sharing. However, the objective of load sharing can sometimes be transformed to an equivalent load-balancing objective.

We can have multiple servers in a server farm or data centres which can hosts multiple guests. Each guest may differ on load, leading to a situation where some servers may become heavily loaded, moderately loaded or lightly loaded in terms of computational resources, memory resources and I/O devices. For simplicity and without loss of generality, we will consider only in terms of computational resources or CPU time.

In this case, when one server is struggling to allocate sufficient CPU slice due to heavy demand by other VMs running parallel on the same server and another server is free/idle.

In this situation, we can distribute the load from one server to another server by migrating one guest to the idle server. This would be an ideal situation when we require load balancing policy or mechanism in Cloud Computing scenario, thus improving the percentage of server idle times, marginal job response times, etc.

Definition: According to H.C Lin [5], a dynamic load-balancing policy consists of three components - namely, information rule, transfer rule, and location rule.

- The information rule describes how to collect and where to store the information used in making decisions.
- The transfer rule is used to determine when to initiate an attempt to transfer a job and whether or not to transfer a job.
- The location rule chooses the nodes to or from which jobs will be transferred.

Each of the three rules can either be performed at a central location or at local sites in a distributed manner.

It is clear that using global state information has a better chance of making correct decisions and providing better performance than using partial information. However, it is expensive in terms of overheads for a fully distributed policy to keep the global state information in each of the nodes. Therefore partial information is usually used in making decisions in distributed load-balancing policies.

B. Issues related to load balancing of jobs in distributed environment.

In order to achieve, short response time and high system throughput, we need to consider these factors:

1. The load balancing process generates little traffic overhead and adds low overhead on the computational and network resources.
2. It keeps up to date load information of the systems participating in distributed computing.
3. It balances the system fairly. i.e. It must balance the heavily loaded and lightly loaded systems first.
4. The load balancing takes small time, else it may affect the overall performance.
5. The load balancing algorithm may take action instantaneously or on periodic basis.

- 6. It can run on a dedicated system, or it can be decentralized effort where everybody is fairly participating in decision making.

PROPOSED POLICY AND APPROACH

A. Motivation and Related Work

A variety of load balancing algorithms has been studied in the past. Kunz [4] used a policy in which each node periodically broadcasts its status (light-loaded or heavy-loaded) to all participating nodes. L. M. Ni [7] used three values (Heavy, Normal, and Light) to represent the load status of a node.

Our work is primarily motivated from the earlier work carried out by H.C.L in and Raghvendra [5] and few other work related to same domain for balancing load in distributed environment.

B. Proposed Approach

In our system, we denote each guest OS as a job and each participating servers as a node. See Figure 1. We have tried and implemented the existing approaches of load balancing of jobs for balancing server loads by migrating guests to lightly loaded server. With the advances in communication technology, the speed of the communication networks in distributed systems is becoming faster and faster.

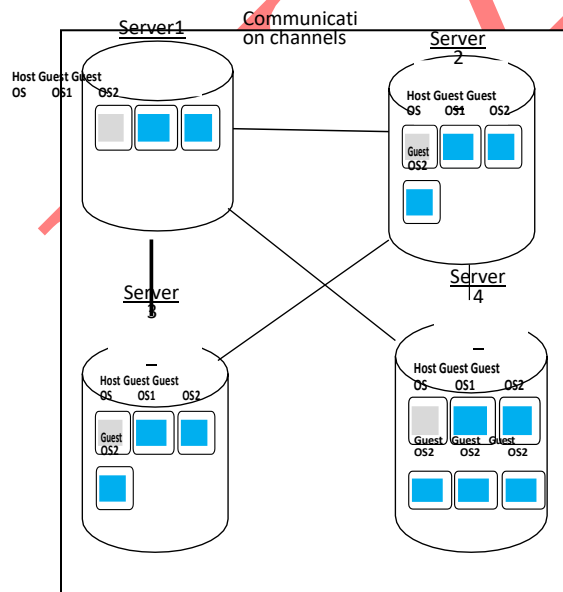


Figure1: Multiple Server Nodes interconnected with High-Speed communication links

High-speed communication can be easily achieved for local area networks. The loads of the nodes in a distributed system are subject to change from time to time; thus it is important for load balancing policies to adapt to the changing load quickly. Our goal is to make the best use of the global state information in order to achieve the best performance without incurring too much overhead compared to simple distributed policies.

We propose a load-balancing policy with a central job dispatcher called “Central Load Balancing Policy for Virtual Machines (CLBVM)”, which makes load-balancing decisions based on global state information. This policy has centralized information and location rules. The transfer rule is partially distributed and partially centralized.

We will show that CLBVM can adapt to the changing load by demonstrating the insensitivity of the average job response to heterogeneous load. We also study the scalability of the central job dispatcher to handle large number of nodes. We expect that a dedicated virtual machine/Host OS also known as Central Server (possibly the host OS on any one of multiple servers) can easily handle more than a thousand nodes. The drawback of any load-balancing policy with a central load balancing system is that of reliability. Failure of this virtual machine will make the load-balancing policy inoperable. One approach to deal with this problem is to design a highly reliable load balancing system. Another solution is to have a secondary (backup) virtual machine, acting as fault tolerant system which collects all the needed information in a passive manner.

C. Central Load Balancing Policy for Virtual Machines(CLBVM)

1. Network load is constant and does not change frequently.
2. Each virtual machine has different identifications and IP Address.
3. The load information collector daemon process, runs continuously in each participating servers collecting aggregate CPU load and system utilization by guests.
4. Based on the data collected by the participating servers, it will mark itself as Heavily Loaded (H), Moderate Load (M) or Lightly Loaded(L).
5. The messages will be exchanged with the Master Server (Central Server), which will take decisions periodically related to load balancing and migration of virtual machines from one server to another on the fly.
6. Heavily loaded systems are balanced first with lightly loaded systems to achieve fairness in balancing.

7. If all are moderately loaded, no migration is carried out.
8. Frequent change in state is taken care by the period of running the load balancing algorithm in master server, so that unnecessary migrations are avoided.

EXPERIMENTAL SETUP

A. Hardware Configuration

Systems Configuration: 1

Intel(R) Pentium(R) 4 CPU 2.00 GHz (~ 1994 Hz)

RAM : 1GB

HDD : 40GB

Host OS : Fedora 8, Xen enabled Linux Kernel

RAM : 1015 MB

No. of CPUs: 1

IP : 10.100.64.52

Work : Acting as Central Server

Systems Configuration: 2

Intel(R) Pentium(R) 4 CPU 2.00 GHz (~ 1994 MHz)

RAM : 1 GB

HDD : 40GB
Host OS : CentOS 5.2, Xen Enabled Linux Kernel (2.6.18-92.el5xen)
RAM : 1015 MB
No. Of CPUs: 1
IP : 10.100.64.29
Work : Acting as Participating Server

Systems Configuration: 3

Intel(R) Pentium(R) 4 CPU 2.00 GHz (~ 1994 MHz)

RAM : 1GB
HDD : 40GB
Host OS : CentOS 5.2, Xen Enabled Linux Kernel(2.6.18-92.el5xen)
RAM : 1015 MB
No. Of CPUs: 1
IP : 10.100.64.30
Work : Acting as Participating Server

Systems Configuration: 4

Intel(R) Pentium(R) 4 CPU 1.8 GHz (~ 1794 MHz)

RAM : 1GB

HDD : 40GB

Host OS : CentOS 5.2, Xen Enabled Linux Kernel (2.6.18-92.el5xen)

RAM : 1015 MB

No. Of CPUs : 1

IP : 10.100.64.32

Work : Acting as Participating Server

B. Software Tools Used

WebServer : Apache

Http Load Generator : Httpperf

Performance Measurement : Xenmon

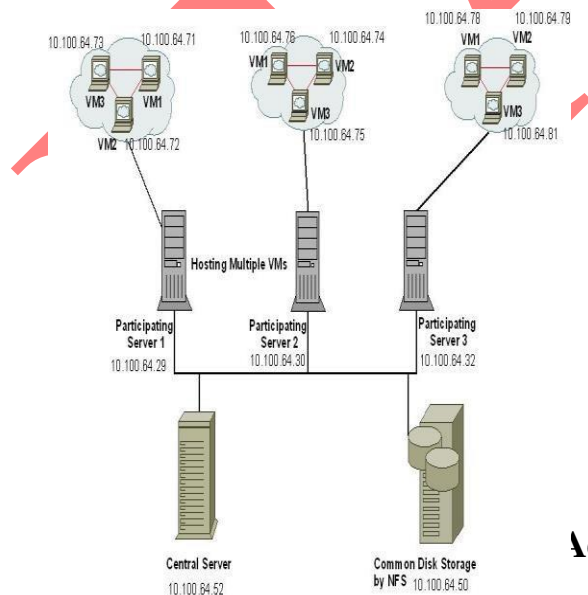


Figure 2 : Experimental Setup

Apache WebServer:

Apache web server supports multiple pthreads, and serves a trivial CGI programs and deploy html pages and web-services.

Httpperf:

It is an open source web server load generator. It provides a flexible facility for generating various HTTP workloads and for measuring the web server performance [6].

XenMon:

xenmon reports a metric execution count that reflects how often a domain has been scheduled on a CPU during the measurement period[3].

C. Experiments Performed

Experiment 1: To measure the performance of isolated web server and establish a benchmark for making a benchmark, we measured the performance of the web server by varying the load at different rates i.e., 512, 1024, 2048, 4096(connections/sec).

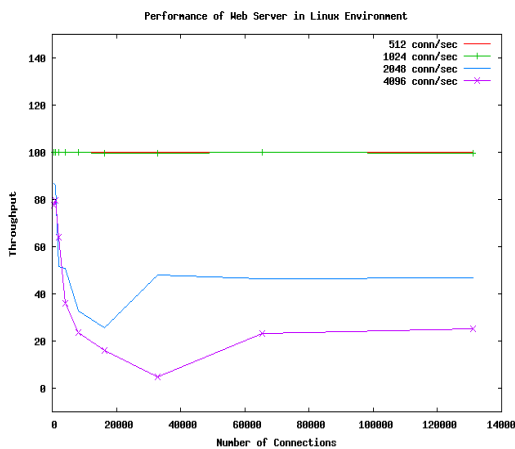


Figure 3 : Web Server Performance on Core Linux Kernel

Interpretation: We found that when data rate was 512 and 1024 conn/sec, the webserver responded almost 100% times without any failure. Refer Figure 5.2. But, as we increased it to 2048 and 4096 conn/sec, we noticed many timed-out connections and file descriptor unavailability. We noticed that after providing sufficient delay time for closing the previously opened file descriptors, we found that the throughput increased from 25% to 45% when the data rate was 2048 connections/sec. Similarly, with the connection rate of 4096 connections/sec, it showed slight improved throughput, though it was far less than that of slower data rates i.e. 512 and 1024. The throughput is low with higher rates because of the web server is unable to handle so many requests due to its limitations on number of file descriptors and number of processes and threads it can spawn.

From this observations, we found that when the data rates were less i.e. 512 and 1024 the Web Server hosted on core Linux kernel was able to handle almost all connections with very few time-outs and file descriptor-unavailability, but as we increased the data rates, we saw a drastic decrease in the throughput. The dips in the graph is due to the unavailability of file descriptors to handle the requests. There are only a limited number of file descriptors available for single processes; httpperf assumes that the maximum is 1024. When a socket closes it enters the TIMEWAIT state for sixty seconds, so we must avoid reaching the port number limitation. We therefore run each benchmark for successive number of connections (i.e. 512, 1024, 2048, 4096, ..., 131072), and then wait for all sockets to leave the TIMEWAIT state before we continue with the next benchmark run. We treated this results as our benchmark for the further experiments to analyse the behavior of Xen hypervisor, OpenVZ Kernels, VMs hosted on Xen hypervisor and OpenVZ containers, and overall performance of the proposed load balancing policy. From these results we can compare and conclude which virtualization technique would meet the better throughputs in which scenario.

Experiment 2: To measure the performance of web server hosted on Dom0-XEN Hypervisor

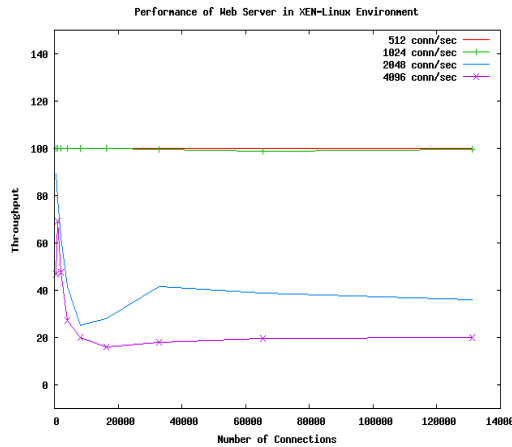


Figure 4: XEN-Linux Web Server Performance

Interpretation: We found similar behavior for Xen-Enabled Dom0 VM. It also showed almost 100% throughput when the data rates were 512 and 1024. Refer Figure 5.3. As the data rates were increased twice and even more, we noticed many timed-out connections and file descriptor unavailability, which resulted in decreased throughput. It also, resulted with approximately 40% and 20% throughput at 2048 and 4096 connections/sec. It showed slight decrease in throughputs because we believe it was due to the additional layer of Xen hypervisor. This is because; Xen uses synchronous calls from a domain to Xen using a hypercall, while notifications are delivered to domains from Xen using a synchronous “event” mechanism. It remains to be seen whether this request processing latency is due to:

- Accepting incoming connections
- Writing the response (nonblocking write () sys.call)
- Managing the cache
- Some unforeseen problem.

Experiment 3: To measure the performance of web server in VM hosted on XEN

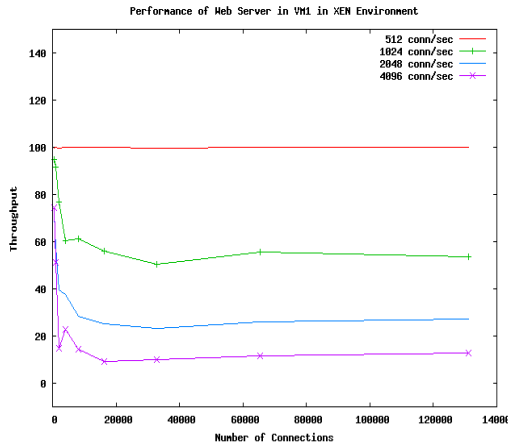


Figure 5: VM1 Web Server Performance hosted on XEN

Interpretation: We found that, the web server running on the VM1/Dom1 hosted on Xen, showed 100% throughput, when the data rate was slower i.e. 512 conn/sec. Refer Figure 5.4. As the data rates were increased to twice and more, we noticed decrease in the throughput and many timed-out connections. Though, it showed similar throughput at higher data rates i.e. 4096 conn/sec when compared to Dom0 running on Xen hypervisor. This is mainly because of the bridge connections used to multiplex the ethernet cards and maintain network connections with guests. Also, Dom0 i.e. HostOS (Linux) takes some processing time which resulted in this behaviour. Refer Figure 1.5. Communication from Xen to a domain is provided through an synchronous event mechanism, which replaces the usual delivery mechanisms for device interrupts. Xen uses a ring implemented as circular queue of descriptors allocated by a domain which is accessible from within Xen. The descriptors do not directly contain I/O data; Instead I/O data buffers are allocated by the Guest OS and indirectly referenced by I/O descriptors. Access to each ring is based around two pairs of producer-consumer pointers: domain place request on a ring, advancing a request producer pointer, and Xen removes these requests for handling, advancing an associated request consumer pointer. Responses are placed back in the same fashion. The VM associates a unique identifier with each request which is reproduced in the associated response. This allows Xen to unambiguously reorder I/O operations due to scheduling or priority considerations. We believe this was the main reason behind the degradation of throughput on the VMs running on Xen.

Experiment 4:To measure the performance of web server in VM hosted on Xen, with compute intensive task on another VM

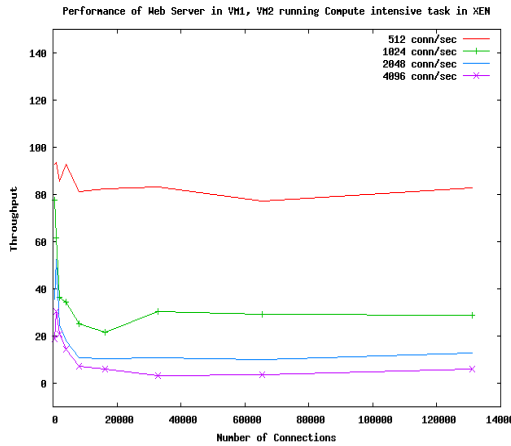


Figure 6 : VM1 Web Server Performance, VM2 running Compute intensive task, hosted on XEN

Interpretation: It showed slightly degraded behavior, as expected in all the connection rates as another VM was running compute intensive task consuming more CPU time. Refer Figure 5.5. At 512 connections/second, throughput varied from 80% to 95%. We noticed quite degraded performance with higher data rates, and would be unable to meet the needs of real traffic on the web server on virtual system. After investigating the scheduling mechanism, we found that it was because of the working of credit scheduler. The other VM which was running compute intensive job consumed more credits as it also supports WC mode, which means the shares are merely guarantees, and the CPU is idle if and only if there is no runnable client. It means that in a case of two clients with equal weights and a situation when one of these clients is blocked, the other client can consume the entire CPU. Since, our script gave sufficient delay to close all the opened file descriptors, the other VM2 used to consume entire CPU, but when VM1 needed more CPU time, it suffered and got only its share percentage, leading to relatively poor performance.

This behavior of the VM1 in this scenario made us to think and resolve this issue with the load balancing mechanism running on multiple servers with virtualization support. Detailed description and results are provided in Experiment 5.

Experiment 5: To measure the performance of web server in Guest OS hosted on Xen adopting CLBVM Policy

Interpretation: From the results, we found that after adopting our CLBVM policy, the results were quite surprising and interesting; it behaved the way we expected and worked for this kind of setup. It showed slightly better throughput, though we expected even better results. Refer Figure 5.9, 5.10, 5.11, 5.12. If we compare our results with Experiment 4 where we found quite degraded performance, the results were quite acceptable and we hope it would give better throughput if we would execute the same experiment on a real world traffic and activity.

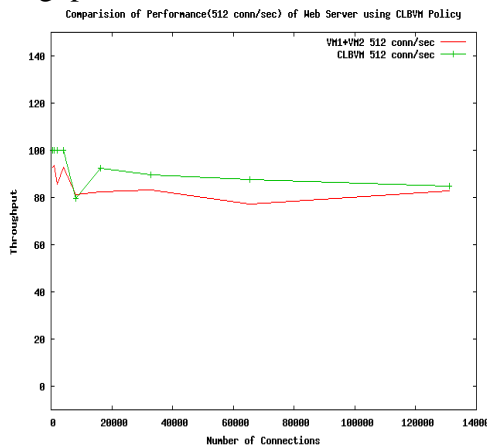


Figure 7 : VM1 Web server performance (512 conn/sec), running on Xen adopting CLBVM policy

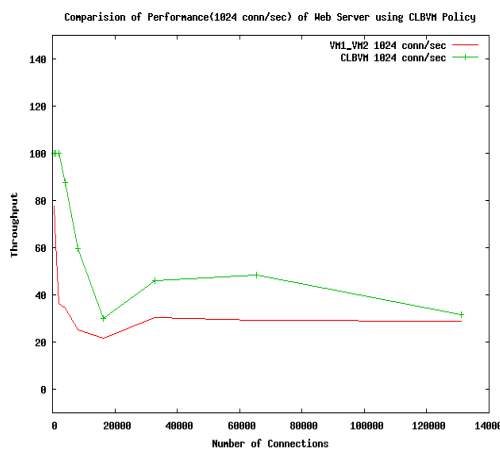


Figure 8: VM1 Web server performance(1024conn/sec), running on Xen adopting CLBVM policy

Our CLBVM algorithm had two sections where client code is running as a daemon service on the participating servers. It used to inform the central server when, the participating server changes its state from High to Low, Moderate and vice versa. The central server used to run the load balancing algorithm every N Minutes (for experiment we have fixed it to 10 minutes) and instructed the highly loaded server to transfer any lightly loaded VM to Lightly loaded server. During, the course of our experiments it migrated the VM on which we were performing our experiment 2 times. It can be seen during the data rate 1024conn/sec making 131072 connections which took approximately 128 seconds to finish the experiment.

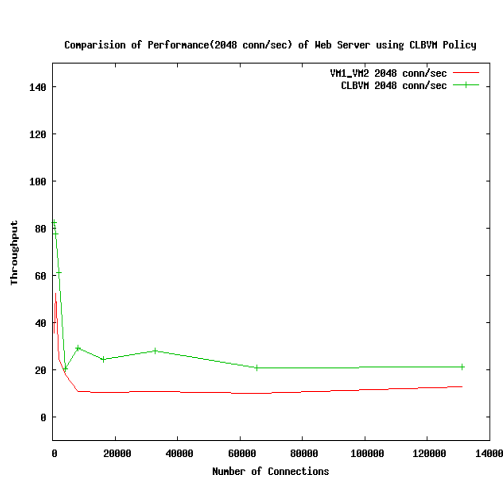


Figure 9: VM1 Web server performance(2048 conn/sec), running on Xen adopting CLBVM policy

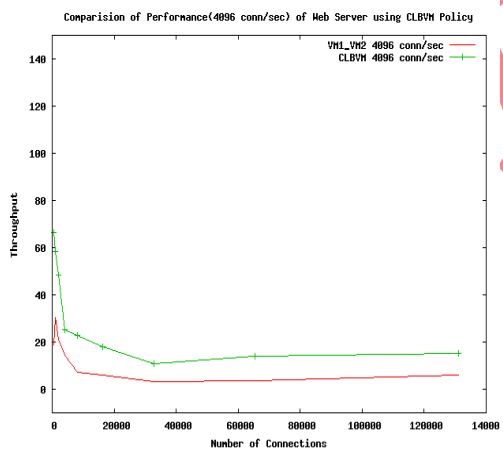


Figure10: VM1 Web server performance(4096 conn/sec), running on Xen adopting CLBVM policy

We believe, despite adding extra burden to CPU by executing client code on each server, we achieved a better throughput. The client code hardly consumed 1-2% of CPU time, but this can be ignored if we achieve better results by losing this percentage of CPU time.

RESULTS AND CONCLUSION

From the above experiments we noticed, that OpenVZ performed well in terms of the throughput of the web server and has less overhead than the Xen hypervisor. The reasons were due to design approach in both the virtualization technique. OpenVZ uses CFQ I/O Scheduler and Fair share scheduler whereas Xen uses Credit scheduler and its own I/O model to handle requests by implementing ring mechanism supported with queuing mechanism. The main drawback with OpenVZ virtualization or container based virtualization is that it only supports Linux kernel and currently has no support for other operating systems acting as guests. So, we decided to choose Xen as it supports almost all operating systems with para-virtualization and full-virtualization support. We tried to make the system completely distributed such that, if the performance of the VM gets affected by other VM, it can move to lightly loaded server on the fly with adoption of our load balancing policy. It adds an extra cost to the overall system by consuming 1-2% of CPU time with each participating servers. Also, we need a server which can also act as central server, which also consumes around 1-2% of CPU time as observed using top command on Linux terminals. The migration of operating system adds an extra cost to the overall system, which we tried to minimize this by selecting the inactive or least active virtual machine. The only issue with this type of design is that we rely only on central server; we could find some better approach to resolve the dependency on a single central server. We could design another system where we can deploy a fault tolerant system acting as a backup and can be brought in place if the master server fails to work in any case. We haven't taken this thing into account in this implementation.

After achieving this result, we are sure that it can be scaled to handle even more nodes in real situations making realization of true cloud computing using virtualization over distributed environment. We also found that if we need to support only Linux environment, we could opt OpenVZ kind of container based virtualization technique, which adds little or negligible burden to the Linux kernel. OpenVZ is a patch over the Linux kernel, so basically it is just a Linux kernel with little modification to support container based virtualization. This policy can even be used in OpenVZ environment as it also supports features of live migration and check pointing based storage and retrieval of the running operating system.

REFERENCES

- [1] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. (2005), Live migration of virtual machines. *In NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, USENIX Association.
- [2] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. (1986), Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Eng.*, 12(5):662–675.
- [3] D.Gupta, R. Gardner, and L. Cherkasova. (2005); Xenmon: Qos monitoring and performance profiling tool. *Technical Report HPL-2005-187*, HP Labs.
- [4] T. Kunz. (Jul 1991); The influence of different workload descriptions on a heuristic load balancing scheme. *Software Engineering, IEEE Transactions*, 17(7):725–730.
- [5] Hwa-Chun Lin and C. S. Raghavendra. (1992); A dynamic load-balancing policy with a central job dispatcher (lbc). *IEEE Trans. Softw. Eng.*, 2(2):148–158.
- [6] D. Mosberger and T. Jin. (1998); httpperf: A tool for measuring web server performance.
- [7] L.M. Ni, Chong-Wei Xu, and T.B. Gendreau. (October, 1985); A distributed drafting algorithm for load balancing. *Software Engineering, IEEE Transactions*, SE-11(10):1153–1161.
- [8] S. Zhou. (1988), A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Softw. Eng.*, 14(9):1327–1341.
- [9] Rajesh Pathak, Asif Khan, (Jan-Jun 2012); Enhancement in Performance of CPU Scheduling using improved scheduling in Xen, *Titulo Clave Ekansh Journal of Engineering, ICT and Management*; ISSN 2230-9756, pages 45-52.