SCS: A VM-Based Approach to Achieve High Scalable and Available Server Environment

Bin Chen and Nong Xiao
School of Computer, National University of Defense Technology, Changsha 410073, China
{chenbin, nongxiao}@nudt.edu.cn

Abstract

This paper presents a novel approach to achieve high scalability and availability in a server environment based on operating system level Virtual Machine (VM). This approach mainly uses VM technology and Maximum Amount and Most Available Service Capacity Finding (MA2SCF) algorithm designed by us to achieve Service Capacity Sharing (SCS) for overloading relaxation and fault management in a server environment. And we have implemented a prototype system including two software components VMMaster and VMAgent to interact with the server environment and VMs, and to automatically manage the SCS process. The current implementation has been tested in Linux Virtual Server environment (LVS) and the approach itself can be applied to a more general server environment, and even a distributed or grid environment which are our ongoing work. The experiment we present in this paper proves the feasibility and effectiveness of our approach.

1. Introduction

In recent years, as the Internet becomes increasingly popular and even an indispensable part of many people’s life, the user and the data traffic the Internet carries are presenting an exponential order increasing. Some hot web sites providing popular network service, such as WWW, are often confronted with server failure, heavy load and even sometimes overloading situations especially during the peak periods of activity, thus unable to ensure QoS for customers and maybe result in immeasurable loss. Many Internet service providers (ISP) feel great pressures towards such situations and worry about whether their IT infrastructure has enough service capacity to satisfy the increasing needs. Therefore, how to improve the scalability and availability of IT infrastructure and thus to provide better QoS for customer becomes one of the issues the ISPs most care about.

The IT infrastructure that the ISPs or other organizations own are usually including tens, hundreds, even thousands of interconnected servers. These servers may be heterogeneous or homogeneous, high-end or low-end, local or widely distributed, and may be grouped into multiple server pools providing different services. We in this paper uniformly call it a server environment.

The straightforward solution to the overloading problem in such a server environment is to prepare enough hardware resources once overloading occurs. However, the hardware resources are static, and the user and traffic over Internet are highly dynamic and increasing. How could the ISPs keep up with the changing and afford the costly hardware upgrading?

Currently, most ISPs can provide multiple types of service for customers, for example, not only WWW, but also FTP, Email, etc. For those ISPs with mid or large-scale server environment, the costs to deploy all types of service in each server are prohibitively high. Actually, some services may be deployed in different server pools to avoid interferences between services, and also some low-traffic services are deployed in the same server pool for the sake of resource saving. However, such a service-clustered server environment may result in resource underutilization in some situations. For example, in peak periods of activity, the servers running the high-traffic services may be overloaded but those running the low-traffic services may be in a light load status. Service availability is another main concern. To some uninterruptible services, such as e-business, and some services that QoS must be guaranteed,

The work was partially supported by National Natural Science Foundation of China under Grant No.60573135, and National Basic Research Program (973) under Grant No.2005CB321801 and No.2003CB317008, and 863-2006AA01A118.
high availability is necessary and must be ensured even in the case of system failure.

We in this paper introduce a novel approach—Service Capacity Sharing (SCS)—to further improve the scalability and availability of such server environment, which are done quite well in some projects or products such as LVS [7,14]. Here, the term service capacity refers to the maximum capability of a service hosting environment (i.e., a server pool) to serve the user. It is clear that the more hardware resources one service uses, such as processing power, memory, the larger capacity the service has. We define SCS as a process proceeding in a server environment hosting multiple services (i.e., multiple server pools) and during which the service capacity of one service can be extended by sharing the service capacities of other services under some situations such as overloading, system failure. We design a MA2SCF algorithm to find maximum amount and most available service capacity to perform effective SCS process, and take the form of Virtual Machine (VM) to perform SCS in a timely fashion. By using our approach, the utilization of existing hardware resources in a server environment can be further improved, and the unnecessary and costly hardware upgrading can be avoided. However, realizing this goal is not an easy task. When the SCS proceed? How much capacity can be shared to relax the heavy load or overloading? When and how to return the shared capacity—Service Capacity Returning (SCR)? We in this paper give our solutions to these problems.

The Linux Virtual Server (LVS) project [7,14], which is a famous open source project, aims at providing scalable, high available and manageable network services environment through IP layer load balancing techniques. However, the IP layer load balancing techniques in LVS is more suitable for homogeneous services environment, i.e., every server is running the same one or multiple services. For a more general server environment, LVS approach may has some problems which we will describe in Section 2.3. Our approach and the prototype implementation are currently based on LVS and aims at solving these problems so as to achieve higher scalability and availability.

The rest of this paper is organized as follows: Section 2 describes the design of our system. Section 3 presents the implementation of the main components in our system. Section 4 describes our experiment on the prototype system. Section 5 describes related work in this field and section 6 presents the final conclusion of this paper.

2. Design

Our system has two main software components: VMMaster and VMAgent. They are both daemon programs. VMMaster receives monitored load information of servers from VMAgents, sends orders to VMAgents to perform some tasks according to the load information, and executes the MA2SCF algorithm. VMAgent running in every server monitors the load of the server, reports them to the VMMaster periodically, and also interact with Virtual Machines according to orders from VMMaster. With the help of the two software components and existing OS-level Virtual Machine technologies, we are able to achieve high scalability and availability in a server environment which has the following salient features:

- It uses software-managed automatic SCS process in the form of VM to achieve heavy load or overloading relaxation in a timely fashion.
- It can ensure better QoS for customers through the SCS process for fault management by not decreasing the service capacity when some hardware or software system fails.
- It can ensure providing uninterrupted service through the SCS process for fault management when the hardware or software providing the uninterruptible services fails.
- The efficient MA2SCF algorithm ensures an effective SCS process for heavy load or overloading relaxation and fault management.

In this section, we will describe the detail design of our system.

2.1 System Overview

The overview of our system is shown in Figure 1. The server environment consists of four server pools providing four different services. Intersections are allowed to exist between different server pools, which means each server in the intersections is providing multiple services.

VMMaster is running in a server that performs some global administration tasks, for example a front-end load balancer. VMAgent is running in each server of each server pool. VM Store is a large-capacity storage server and stores VM images that encapsulate different services the server environment is providing. The VM images of some overloaded services will be transferred to specific servers to be instantiated there by VMAgents as needed by the SCS process. The MP-LLS (Matching Power and Lightest Load Set), which is the execution result of the MA2SCF
algorithm, includes those servers most appropriate to share their service capacities. The SCS processes proceed between different server pools. Figure 1 shows the SCS processes for both overloading relaxation and fault management.

**Figure 1. System Overview**

### 2.2 Why SCS with Virtual Machine?

Operating system level Virtual Machine [9] provides a useful abstraction layer which can greatly simplify addressing many issues in distributed computing environments [4]. The SCS will be very difficult to be achieved without VM for the diversity of services and service hosting environments. We consider VM the most appropriate tool to achieve our goal for VM has the following advantages:

**Encapsulation:** VM has the abilities to encapsulate a whole guest OS and its applications in image files stored in the host file system, and to boot the VM from the image files as needed.

**Mobility:** VM image files can be copied or moved to any physical machine as regular files and then the VM can be booted on them, only requiring corresponding VMM [9] (Virtual Machine Monitor) has been installed in the physical machine for most VM technologies.

**Controllable:** A running VM is under the full control of VMM. VMM can boot a VM from power-off state, suspend or reboot or shutdown a VM at its running state and can restore a VM to its previous running state from suspend image files in a faster fashion than booting from scratch.

**Security and Isolation:** Most operating system level VM technologies can achieve stronger software security and isolation than a conventional multiprogrammed operating system approach [10]. The failure of the software running inside the VM will not affect the host environment.

**Compatibility:** VM supports legacy applications at the level of binary code [4]. The legacy applications are able to run in a VM without modifying and recompiling the source code only requiring the supports of the OS environment by the VMM.

Since VMs have the above many advantages, we can utilize VMs to solve some problems encountered in a server environment. We can encapsulate the whole running environment of different type of services providing by the server environment in different VM image files, and can transfer the VM image files to any physical server that has the capacity to run the service, and then start the service by instantiating (booting) the VM from the image files. If the service running in the VM is no longer needed, we can suspend or shutdown the running VM, and thereafter can also restore it from the suspended VM image files quickly or re-instantiate the VM if needed at any time.
2.3 LVS Problems

The Linux Virtual Server (LVS) project [7, 14] is a famous open source project. LVS mainly uses IP layer scheduling techniques to provide load balancing ability for enterprise server environment. The most important feature of LVS is that it implements those load-balancing scheduling techniques in Linux OS kernel, which incurs little scheduling overhead.

LVS has solved the load balancing problem quite well which is the basis of our work but not what we care about. We consider LVS more suitable for homogeneous services environment than for a general server environment for the following reasons in two aspects:

**Scalability**: LVS is most concerned with load balancing of a server pool and it can not dynamically extend the service capacity of a service running inside a server pool when all of the servers are overloaded.

**Availability and QoS**: When some system fails in a server pool, LVS uses zero weight value to ensure the following user requests will not be scheduled to the fails so as to do some maintaining work. Though effective is this approach, the overall service capacity will decrease for some time, which may cause problems in some situations, such as overloading in peak periods of activity. In such situations, the high availability would be compromised and the customer may get bad QoS.

Our work aims at solving these problems so as to achieve higher scalability and availability.

2.4 Main Components

2.4.1 VMMaster

VMMaster is a master daemon program which manages multiple slave daemon program (i.e. VMAgent) running in every physical and virtual servers (VM). VMMaster is running in a server that performs some global administration tasks, for example in a front-end load balancer. To avoid single point failure, a backup VMMaster can be deployed in another server. The backup VMMaster uses UDP heartbeat signal to communicate with the main VMMaster periodically and will take over the work of the main VMMaster once the main VMMaster fails. VMMaster receives monitored load information of servers from VMAgents periodically, makes decisions about when to start SCS and SCR processes based on received load information, executes the MA2SCF algorithm, and sends orders to VMAgents to perform some VM-related tasks.

Another function of VMMaster is interaction with the local server environment, such as get/set some information from/to the server environment. For example, when a VM has been booted up like a new "real" machine to be appeared in the server environment, VMMaster has the responsibility to notify the server environment. Due to the diversity of server environments, VMMaster uses a high level abstract Server Environment Interface (SEI) to interact with different server environments in a uniform way. However, server environment specific adaptor must be implemented according to the SEI.

2.4.2 VMAgent

VMAgent is a slave daemon program which is managed by VMMaster and needed to be deployed in every physical server and VM. VMAgent monitors the load of the server and reports them to the VMMaster periodically. VMAgent also receives orders from VMMaster and does some VM-related work, such as transferring VM image files from storage server, VM instantiation, suspending or shutdown. For different VM technologies, VMAgent uses a high level abstract VM Interface (VMI) and VM-specific adaptor to interact with specific VM technology. Below the high level abstract VM Interface, the underlying mechanism differs for different VM technologies. For example, for VMware GSX Server, the corresponding VM adaptor uses Perl APIs provided by VMware to control the VMs, and for UML, the VM adaptor uses management console utilities provided by UML project [12, 13] to interact with UML VMs.

Note that the load information the VMAgent gets from a physical server is different from that the VMAgent gets from a VM for only a proportion of total CPU time of the real machine is dedicated to the running VM. The "virtual" load information is only useful for load balancing in which the "virtual" load information seems to be from a "real" machine. However, when VMMaster executes the MA2SCF algorithm, it uses the load information from real servers not from the VMs.
2.4.3 Service Capacity Sharing (SCS)

VMMaster is responsible for making decisions about when to start the SCS process which relies on the policies made by the administrator. In our system, we have made the following default policies which prescribe at least one of the following conditions to be satisfied when the SCS process starts.

1. Some server pools are overloading (not caused by previous SCS process), but there are still server pools with relatively lighter load in the server environment.
2. Some servers carrying more traffic than the others in the server pool fail, which may cause the server pool overloading due to the importance of the servers. It is the same situation as 1.
3. Some servers providing the uninterruptible service fail. Regardless of the load status, the failure of such kind of service may not be allowed.
4. Some servers providing the QoS-prioritized service fail, which may cause the load of the server pool increase even to the overloading level and thus make the QoS not guaranteed.

Finding maximum amount and most available service capacity is the key issue in the SCS process, i.e. finding a set of physical servers which are most appropriate and most possible to provide maximum amount of service capacities. The MA2SCF algorithm is responsible for the job. Since the dynamic characteristic of a server environment, finding maximum amount service capacity sometimes may conflict with finding most available service capacity. Thus the algorithm would better be able to find an optimal tradeoff. MP-LLS is the result of the algorithm execution and contains those servers with the lightest load and matching processing power with the average level of overloaded or failed servers. Lightest load is to ensure the service capacity “contributor” will not be overloaded or be severely affected during the SCS process. Matching power is to ensure the “beneficiary” will get enough service capacity to relax the overloading as quickly as possible. The number of VMs used to do service capacity sharing is based on an initial estimated base number specified by the administrator, and is dynamically adjusted according to the extent the service load has been relaxed to. Once the MP-LLS and the base number are determined, the VMMaster will send orders to the VMAgents on the physical servers in MP-LLS to start VM transferring from the VM Store to local storage and then instantiation. After VM instantiation, VMAgent will configure the VM with a temporarily allocated IP address from a reserved IP address pool and then starts the service inside the VM.

2.4.4 Service Capacity Returning (SCR)

Once the overloading is relaxed to a certain level, the SCR process will be started during which the shared capacity from other services will be returned. VMMaster will give orders to some VMAgents running in the physical server to stop the running VMs. On receiving orders, the VMAgents call the VMI to suspend or shutdown the running VMs. The SCR process should proceed in a step by step manner to avoid overloading again quickly. Once the load increases to a certain level, the SCR process should be stopped to avoid overloading again caused by abrupt load increase. If the SCR process has been stopped but the load continues increasing to the overloading level, the SCS process will be again started. Since the server environment may be highly dynamic, we must have mechanism to avoid thrashing in our system, i.e. avoiding repeatedly transition between the SCS and the SCR processes in short period.

2.4.5 VM Supporting in SCS

Our prototype system currently supports VMWare GSX Server and UML (User-Mode Linux). And at the time of this writing, we have finished our implementation on Xen [8] which is a more scalable and high performance Virtual Machine Monitor. VMWare and UML both support COW (copy-on-write) mechanism, which is the most desirable characteristic for our system. Most VM technologies encapsulate the whole running environment of a machine in image files, such as virtual disk in VMWare, root file system in UML. Since having encapsulated the whole running environment, the image files usually have big size, at least several hundreds of Megabytes, which is not suitable for application in distributed environments. The COW mechanism can greatly loosen this restriction for the COW files usually have small size and can be quickly transferred between sites. The COW files for different VMs can have the same base VM image files, such as an original snapshot or independent non-persistent virtual disk in VMWare and read-only root file system (hacking file) in UML. The COW files only capture the differences of different VMs sharing the same base VM image [2]. So the small-sized COW file of an individual VM can be quickly transferred to any host which has the base VM image files, and can be instantiated there. For performance consideration in our system, we prefer deploying the big base VM image files in the destination server beforehand where the services are running and only transferring necessary COW files from the storage server.
3. Implementation

We have implemented a prototype system based on LVS server environment but not tied to it. It is important to note that LVS is not the focus of our work and is used only as our sample server environment and a load-balancing tool. Other load-balancing tools can be substitutes for LVS, such as IBM NetDispatch [11]. And we use UML as our VM technology.

3.1 VMMaster and VMAgent

VMMaster and VMAgent have been implemented using C++ programming language. They are lightweight and easy to deploy. They consume little system resources and have little effect on the other components of the server environment for they are quiet most of the time. Except the periodical load information reporting, they are triggered only in some special situations such as overloading, system failure, and can deal with these situations in an unattended manner once these situations occur.

VMMaster needs to be deployed in a front-end server that performs some administration tasks and where some global information can be gotten from, for example, a front-end load-balancer in the context of LVS. We have implemented an adaptor for LVS environment-ILsadapter according to SEI. The adaptor executes the underlying information getting and setting, more specifically, adding/removing the IP addresses and initial weight value of the new instantiated VMs to/from the schedule list inside IPVS module so that the VMs can share some loads of the original overloaded physical servers.

The main tasks of VMMaster are executing the MA2SCF algorithm according to received load information and ordering VMAgents to execute the SCS process, which will be described in detail in Section 3.2. To improve system reliability, the same VMMaster program can be optionally deployed in another server as a backup VMMaster. They use the same UDP port to report the health status of each other periodically.

The main tasks of VMAgent are monitoring and reporting load status of servers and perform some VM-related tasks during the SCS process. In current implementation, monitored load information only includes CPU, memory and disk space which have the closest relationship with service capacity. VMAgent collects load information from the server and reports them to the VMMaster periodically. The default time interval between two reporting is 10 seconds which can be specified by the administrator. Note the reporting should not be too frequent, otherwise some overhead will be brought to the server environment. And we suggest the time interval to be between 5 and 20 seconds.

3.2 SCS Process

The SCS process is the key to achieve high scalable and available server environment. By using the VM technologies and MA2SCF algorithm, the overloading or system failure situations in a server environment can be dealt with in a timely fashion.

3.2.1 MA2SCF Algorithm

The execution of MA2SCF algorithm during the SCS process includes the following three phases:

Overloaded Service Identifying

VMMaster firstly needs to identify which services are overloaded and need to share the service capacities of other services. VMMaster stores three times of history load information. Each time VMMaster calculates an aggregate load value-AggregateLoad for each physical server. An example of AggregateLoad calculation formula is as follows:

\[
\text{AggregateLoad} = r_1 \times \text{CPU} + r_2 \times \text{Mem} + r_3 \times \text{Disk}, \quad r_1 + r_2 + r_3 = 1
\] (1)

The formula can be defined by the administrator during the configuration of VMMaster according to what factors make contributions to the server load. In formula (1), all load values are normalized to make the value between 0
and 1, so are the coefficients $r_i$. Therefore $\text{AggregateLoad}$ is also between 0 and 1. The coefficients are also specified by the administrator according to the importance of contributions to the load of certain service. Usually, the more contribution a factor makes to the load, the bigger corresponding coefficient the administrator should specify. Different services may have different coefficients. The administrator may need to adjust the coefficients to correctly reflect load status for each service.

VMMaster stores three $\text{AggregateLoad}$ values for each server and calculates an $\text{AvgServiceLoad}$ value for each service, which is calculated as follows:

\[
\text{AvgAggregateLoad}_i = \frac{\sum_{j=1}^{3} \text{AggregateLoad}_{ij}}{3}
\]

\[
\text{AvgServiceLoad}_m = \frac{\sum_{i=1}^{n} \text{AvgAggregateLoad}_i}{n}
\]

Note that subscript $i$ denotes the $i$th server providing $\text{service}_m$ and subscript $j$ denotes the $j$th $\text{AggregateLoad}$ value of the $i$th server. So $\text{AvgServiceLoad}_m$ is the average of all history $\text{AggregateLoad}$ value of all servers providing $\text{service}_m$ in default $3 \times 10 = 30$ seconds. The advantage of using history load information is being able to avoid misjudgment of load status in the case of occasional abrupt load variation. VMMaster will consider a service overloaded when corresponding $\text{AvgServiceLoad}$ value exceeds a specified threshold value (default 0.98). In this sense, a service is overloaded usually means all the servers providing the service are overloaded.

Lightest Load Set (LLS) Calculation

If a service overloaded, VMMaster begins to calculate $\text{LLS}$ to include those servers with the lightest load. Importantly, for more correct calculation of $\text{LLS}$, the $\text{LLS}$ calculation must consider different characteristics of different services in the server environment. For example, a server running a CPU-intensive service may have high aggregate load according to formula (1), but may has light load for a memory-intensive service for they use different coefficients $r_i$ to calculate the aggregate load. So when a service overloaded, VMMaster uses coefficients of the overloaded service to calculate the aggregate load of every server except those in the overloaded server pool. The results using this calculation method can truly reflect the load status of other servers in the eye of the overloaded service.

VMMaster also calculates a standard deviation to reflect load variation magnitude in the past three time intervals (default 30 seconds) by using history load information. Those with both smaller $\text{AvgAggregateLoad}$ and smaller standard deviation will be included in the $\text{LLS}$ by using two threshold values which are default 0.8 and 0.2 for upper limits of $\text{AvgAggregateLoad}$ and standard deviation.

If the total number of servers included in $\text{LLS}$ is less than a base number which in our system is specified as $\alpha \times \text{ServerNumber}$ and at least equals 1 (default value of $\alpha$ is 0.1, $\text{ServerNumber}$ denotes the number of physical servers providing the overloaded service), it indicates either the two threshold values are not appropriately specified and can be adjusted to include more servers, or the server environment has not enough service capacity to share with the overloaded service and it should be reported to the administrator. Finally VMMaster sorts $\text{LLS}$ in ascending order of load if $\text{LLS}$ is not empty.

Note that the $\text{LLS}$ calculation starts only when some services are overloaded, after which $\text{LLS}$ is periodically calculated every 10 seconds until overloading relaxed. We represent the calculated $\text{LLS}$ as a sorted set as $\text{LLS} = \{s_1, s_2, ..., s_L\}, L \geq 1$.

Power Matching

Besides lightest load, well-matching processing power of physical servers is another consideration. Otherwise the chosen server with the lightest load but also a much lower power than the average of the overloaded servers may be overloaded quickly during the SCS process.

We use weight value to represent the power of a server. The calculation of weight value is the same with $\text{AggregateLoad}$ in formula (1) except that the input values are CPU main frequency, memory and disk sizes of a server not load information. VMMaster only calculates the weight value of servers in the $\text{LLS}$ and the overloaded servers. We represent the calculated weight values of $\text{LLS}$ and overloaded servers as $W_{\text{LLS}} = \{w_{s_1}, w_{s_2}, ..., w_{s_L}\}$ and
$W_{OL} = \{w_1, w_2, ..., w_n\}$ respectively. Then the power matching process is as the following steps:

1. Calculating the average weight value of overloaded servers:
   \[
   \text{AvgWeight} = \frac{1}{n}, \quad w_i \in W_{OL}. \tag{4}
   \]

2. For each weight value $w_i (i \leq L)$ in $W_{LLS}$ do the following comparison:
   \[
   \begin{align*}
   \text{If } w_i & \geq \delta \times \text{AvgWeight} \text{ then} \\
   \text{MP-LLS} & = \text{MP-LLS} \cup \{s_i\} \tag{5}
   \end{align*}
   \]

   The factor $\delta$ is default 0.5. MP-LLS is initially an empty set and if at last is still empty, it probably indicates either factor $\delta$ is too high and should be lowered to include more servers, or the server environment has not enough service capacity to share and should be reported to the administrator.

   Note that the number of servers in MP-LLS may be less than the base number $\alpha \times \text{ServerNumber}$. In this situation, some servers in MP-LLS may need to run more than one VMs to reach the base number during the SCS process.

3. Calculating the priorities of the servers in MP-LLS using the following formula and sorting MP-LLS in descending order of priority.
   \[
   \text{Priority}_i = \beta \times (1 - \text{Load}_i) + \gamma \times \text{Weight}_i, \\
   \beta + \gamma = 1 \quad \text{and} \quad i \leq L \tag{6}
   \]

   The factors $\beta$ and $\gamma$ both are default 0.5. If two servers have the same priority, the server with bigger weight value will be selected first.

   Finally, MP-LLS includes those servers with matching power and relatively lightest load. We represent MP-LLS as $MP - LLS = \{S_1, S_2, ..., S_p\}, P \geq 1$.

3.2.2 SCS for Overloading Relaxation

Having MP-LLS, VMMaster begins to select servers in sequence from MP-LLS and sends order to its VMAgent to instantiate specific VM. The order includes information about service type, VM type, actions to be taken (VM instantiation, suspend, shutdown), temporarily allocated IP address, etc.

When receiving order of VM instantiation, the VMAgent starts transferring necessary VM image files from VM Store according to the service type if not found locally, and then calls the function of VMI using VM type information as parameters to instantiates specific VM. All VMs are instantiated simultaneously not one by one, which is important to relax overloading in a timely fashion. After VM instantiation, VMAgent calls VMI to configure VM with the temporarily allocated IP address, and then starts the service inside the VM. After all these completed, VMAgent returns an operation code to VMMaster to indicate success or failure.

If all succeed, VMMaster calls the function of SEI to set the addresses of new booted "servers" into kernel space of the load balancer in the context of LVS. If a server fails to instantiate VM, VMMaster will reselect another server from the MP-LLS as a substitute. The initial weight of each VM is also set as the same value with the real server. Then the load balancer will schedule successive requests to VMs to relax the overloading.

In our system, we consider overloading relaxed if the load has reached the level $\theta$ (default 10) percent under the overloading threshold. The space set between overloading and overloading-relaxed states can help avoiding repeatedly transition between the overloading and non-overloading states and effectively relaxing overloading.

If the overloading is still not relaxed after base number of VMs have been instantiated, VMMaster will continue to select more VMs from the newly calculated MP-LLS.

During the SCS process, some unexpected events may happen:

1. If the instantiated VM causes the host overloading according to monitored load information and (2), VMMaster will immediately notify VMAgent to shutdown or suspend the VM so as not to affect the service running in the host.

2. If the newly calculated MP-LLS is empty but the overloading is still not relaxed, it indicates not enough service capacity and it should be reported to the administrator.
3.2.3 SCS for Fault Management

In a typical server environment, server failure can be fallen into two types: service software failure and system failure. To detect server failure, VMMaster periodically sends service requests and ICMP packets (ping) to each server every default 10 seconds. If both responses are correct, VMMaster considers the server healthy. If no response to the service request but the ICMP responses are correct, then VMMaster considers the service software has failed. VMMaster will notify the VMAgent to instantiate a specific VM to take over the service providing in the same server. If no responses at all, VMMaster considers the system failed. System failure also can be divided into two types: system software failure and hardware failure for which some manual work have to be done by the administrator, such as rebooting, software and hardware maintaining, and may need more time. VMMaster deals with this type of failure in a simple but effective way. Usually, VMMaster does nothing to deal with system failure except the following three situations:

1. If overloading quickly caused by the failed servers, VMMaster will do what should be done to deal with overloading, i.e. SCS process for overloading relaxation.
2. If all servers providing the uninterruptible service fail, VMMaster will select servers from MP-LLS to instantiate the same number of VMs as the failed servers to take over the interrupted service.
3. If the QoS of the service must be guaranteed and for avoiding great QoS variation caused by system failures, VMMaster will select servers from MP-LLS to instantiate the same number of VMs as the failed servers to back up the "impaired" service.

Note that the current implementation of SCS for fault management is working at the service granularity not the session granularity. When a server serving a user fails, the already built connections will be lost and the user may need to resend the service requests.

3.3 Service Capacity Returning

If the peak periods of activity have gone and the load of overloaded service has been relaxed to a certain low level, it is necessary to return the shared service capacity. This process should proceed in a conservative manner, i.e. stopping the VM one by one not all at one time. Otherwise, overloading may quickly recur. We differentiate the following three states during the SCR process by load level:

**Service Capacity Returning Starting (SCRS):** Once the load of an overloaded service has been relaxed to be below default 60 percent, SCR process will be started. One by one, VMs are shutdown or suspended to return the shared service capacity.

**Service Capacity Returning Continuing (SCRC):** If SCRS state just passed and the load level is between default 60 percent and 90 percent, VMs continue to be shutdown or suspended.

**Service Capacity Returning Finished (SCRF):** If SCRS state just passed and the load exceeds default 90 percent but is below the overloading threshold (default 98 percent), VMMaster will finish SCR process immediately to avoid overloading again.

The complete state transition between SCS and SCR is illustrated in Figure 2 also including an additional start state “S” in which a service is in non-overloading status and no running VMs are providing the service.

When a service is in SCRF state, and the load continue increasing to exceed the overloading threshold, the SCS process will be restarted as (1).

In SCRS and SCRC state, VMs are shutdown one by one. If all VMs are shutdown, the two states will switch to start state “S” as (2) and (3).

Note that the SCRF and SCS states can not switch to SCRC state directly but need to switch to SCRS first and then SCRC. This can help avoiding repeatedly state transition between SCRF and SCRC, and between SCRF and SCS in short period.
3.4 VM Supporting in SCS

In our prototype system, we assume that all physical servers have been installed the VMMs we support and necessary VM tools such as VMWare GSX Server, UML kernel and utilities, which is the prerequisite for most VM-based solutions. And all servers have direct high-speed physical links in a local environment. For our system to work, the following preparatory work should be done beforehand.

**Base VM image making:** For each type of OS the server environment supports, making a base VM image only including the basic OS. For VMWare, the base VM image files are mainly the virtual disk files after you install an OS on it. For UML, the base VM image file is a file named default root_fs made by specific tools [13] and including minimum and necessary software to boot UML kernel.

**Service COW file making:** According to the service types the server environment is providing and specific COW file making mechanism for different VM technologies, the COW file or device for each service can be made by installing necessary software and configuring the service in corresponding base VM image to make the service running properly. For VMWare, service COW file can be made by using none-independent virtual disk and snapshot, or non-persistent virtual disk and redo-log file mechanism once the service installation and configuration completed. For UML, COW file can be easily made by specify the COW file name and the backing file name (i.e. the base VM image) as the UML booting parameter, and can also be made by other tools in UML utilities. Different VMs can share a common read-only root file system and the differences are captured in small-sized COW files in UML. If a server is running multiple services, the COW files corresponding to each service are not independent. For a service to run properly, the COW files of previous installed services are all needed or needed to be merged to make new base VM images for unmerged COW file. The inheritance relationships between service COW files must be kept correctly.

**VM deployment:** We currently simply store all base VM images in each physical server, and store the service COW files in individual directory corresponding to the service in VM storage server. The hierarchical directory structure can exactly represent the relationships between COW files. However, we can optimize the deployment scheme according to service characteristics or by using some cache techniques. For example, for a high traffic and frequently overloaded service, besides the base VM image, we can also beforehand deploy the service COW files on every server. Thus the images transfer time can be eliminated and it can greatly speed up the VM instantiation process.

During the SCS process, if the running VM supports to be suspended and next time if the same VM needs to be instantiated, restoring the VM from the suspended state will greatly speed up the VM instantiation process. Unfortunately, this characteristic is supported by VMWare but not by UML.

4. Experiment

We have done preliminary experiment to show the feasibility of our approach based on our prototype system. We simulated a simple server environment which had two server pools one providing WWW service and the other FTP. All servers were directly connected to a 100M Ethernet switch. Each server pool included three servers, two of which had Pentium-4 1.7GHz CPU, 512MB RAM, 40GB disk and Red Hat Linux 9 (2.4.32 kernel, SKAS patch [13]), and one had Pentium-3 1GHz CPU, 512MB RAM, 40GB disk and Red Hat Linux 7.3. The load balancer had the same configuration with the Pentium-4 server. We ran a Java program in two Windows PCs with Pentium-4 2.4 CPU, 1GB RAM and 80GB disk to generate heavy load in WWW server pool and light load in FTP server pool.
separately. The Java program could generate many threads each of which gets a group of html pages from the server pools periodically using HTTP or FTP protocol. To change load, we only needed to change the thread number or frequency of requests issued by each thread. We used LVS to do the load balancing and UML (2.4.27 kernel, uml-2.24.27-1 patch) as our VM. We started from 10 threads with one request every 200ms and doubled the thread number every 30 seconds until overloading. All coefficients were default except those in table 1. The experiment results are shown in Figure 3. We can see that the overloading of the WWW service occurred between the 2nd and the 2.5th minute during which the SCS process started, the overloading was relaxed between the 3.5th and the 4th minute, and the load stabilized at about 90 percent. The total relaxation time is less than 1.5 minutes which we think is acceptable. We also have observed the instantiation of a UML VM in a Pentium-4 server, which cost most of the relaxation time. At the 4.5th minute, we restored the thread number to the original 10. We can see that the load rapidly dropped and stabilized at 18 percent at about the 5th minute. The SCR process started between the 4.5th and the 5th minute. After the SCR, there was little load increase in the WWW server pool due to shutdown of the VM. The load of the FTP server pool varied little. There is only little load increase during and after the SCS process for the service capacity sharing with the WWW server pool.

Table 1. Coefficients for calculating server weight and service load

<table>
<thead>
<tr>
<th>server</th>
<th>weight</th>
<th>service load</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>0.5, 0.4, 0.1</td>
<td></td>
</tr>
<tr>
<td>FTP</td>
<td>0.6, 0.4, 0.0</td>
<td></td>
</tr>
</tbody>
</table>

5. Related Works

Since the renaissance of VM technology in recent years, VM technology has been applied to many fields, such as enterprise application, grid computing, for its many advantages by using system-level virtualization technologies. SoftUIC [1] mainly uses VM virtualization technologies to aggregate servers, network, and storage resources into a single, centrally and easily managed pool. In SoftUIC, a modified VMM and a management VM must be installed in every server to achieve their goals. However, our work only uses VM to further improve the scalability and availability of a general server environment to which VM are only as a complementary. vioCluster [2] uses VMs to lend and borrow compute nodes between clusters managed by different domains, which has some similarities to our work. However, vioCluster is working on cluster environments which are usually homogenous and managed by special cluster management software, such as PBS. Our work is towards a more general server environment which may be highly heterogeneous and dynamic, and may include more server pools providing multiple services. The SODA [3] project presents a service on demand architecture based on VM. However, they aim at addressing the issue of on-demand application services hosting in a Hosting Utility Platform and use software entities to manage the VMs hosting the services, which are different from our goals. In [4], Renato Figueiredo, etc first proposed applying VM to wide-area grid computing. Our work differs from theirs in that we aim at improving the availability and scalability of a server environment by utilizing VM technology. The same is we both base the work on the many advantages of VM technology. VMPlant [5] provides a general and flexible architecture for automatically VM creation, customization and fast instantiation, which we consider a useful complementary to our work, especially the
VM Store in our system. RAM Grid [6] presents an architecture for wide-area memory sharing. However, by using VM technology, we have achieved service capacity sharing at a coarser granularity, being able to share not only memory but also processing power.

6. Conclusion

We have presented the design and implementation of our VM-based approach to achieve higher availability and scalability in a server environment. Our approach can be a useful complementary to some load-balancing tools used in a server environment, such as LVS. Our preliminary experiment has proved the feasibility and effectiveness of our approach. Though our prototype system only has been tested in LVS, the approach itself can be applied to a more general server environment, and even a distributed or grid environment which are our ongoing work. VM deployment optimization in combination with dynamical load prediction ability based on history information to achieve further performance improvement is also our future efforts.

References


Dr. Nong Xiao is a professor in the School of Computer at the National University of Defense Technology, China. He received his PhD degree in Computer Science and Technology from the National University of Defense Technology in 1996. His research interests are in the areas of grid computing, parallel and distributed computing, data grid, virtual machine.

Bin Chen is currently a PhD student in the School of Computer at the National University of Defense Technology, China. He received his MS degree in Computer Science and Technology from the National University of Defense Technology in 2004.