

DEPARTMENT OF COMPUTER SCIENCE

Developing Co-scheduling Mechanisms for Virtual Machines in Clouds

Peng Jiang

Supervised by Ligang He

2012-2013

A dissertation submitted to the University of Warwick in accordance with the requirements

of the degree of Bachelor of Science in the Faculty of Engineering

Declaration

A dissertation submitted to the University of Warwick in

accordance with the requirements of the degree of Master of Science in

the Faculty of Engineering. It has not been submitted for any other

degree or diploma of any examining body. Except where

specifically acknowledged, it is all the work of the Author.

Peng Jiang, September2013

Content

[Abstract 3](#_Toc366653659)

[1. Introduction 4](#_Toc366653660)

[2. Background 6](#_Toc366653661)

[2.1 Virtualization 6](#_Toc366653662)

[2.2State of art of virtualization 8](#_Toc366653663)

[2.3 Computer Cluster 10](#_Toc366653664)

[3. Literature Review 11](#_Toc366653665)

[4. Design Strategies 15](#_Toc366653666)

[4.1 Problem analysis 15](#_Toc366653667)

[4.2 Possible strategies 16](#_Toc366653668)

[5. Implementation of the co-scheduler 21](#_Toc366653669)

[5.1 Domain management in Xen 22](#_Toc366653670)

[5.2 Domain management tools 23](#_Toc366653671)

[5.3 Implementation Environment 23](#_Toc366653672)

[5.4 Implementation 24](#_Toc366653673)

[6.Performance evaluation 26](#_Toc366653674)

[6.1Experimental environment 26](#_Toc366653675)

[6.2 Experiments objectives 27](#_Toc366653676)

[6.3 Design of the Experiments 27](#_Toc366653677)

[6.4 Experiments results 28](#_Toc366653678)

[7.Conclusions and Future Work 32](#_Toc366653679)

[7.1 Conclusions 32](#_Toc366653680)

[7.2 Further work 32](#_Toc366653681)

[Reference 34](#_Toc366653682)

[Appendix - User Manual 38](#_Toc366653683)

# Abstract

Virtualization technology, such as Xen, allows multiple virtual machines (VM) to co-exist in a single multi-core physical machine. Multiple VMs may host different processes of the same parallel program. Ideally, the multiple VMs should be scheduled to run simultaneously by the VM manager. However, the existing schedulers in Xen do not support this type of co-scheduling. This project aims to develop a co-scheduling mechanism for virtual machines in Xen. The co-scheduler is able to greatly improve the performance of parallel applications running in virtualized environments.

# 1. Introduction

As huge development in hardware, multi-core processors with growing ability of data processing is more and more common nowadays. To efficiently mine the vast potential of processing ability of a multi-core processor is becoming an exciting challenge of computer scientists community. One interesting thought of efficiently making use of this power of computation is to run multiple operating systems on single machine, which can be achieved by virtualization [1][2].

Virtualization, which has become an important direction of exploiting the processing ability of multi-core processors, is a technique that enables multiple standalone operating systems running on a single computer. By applying this technique, each standalone operating system is independently running on virtual hardware. Traditional system uses a software stack, which increases the complexity in communication. It also results in difficulty of managing and lack of stability and security. Unlike traditional approach, Virtual Machine Monitor(VMM) is used in virtualization system to virtualized physical CPUs ( PCPUs ) and to provide virtualized CPUs (VCPUs) which is employed by guest system. VMM is responsible for communication between guest operating system and hardware, and controls the access to other guest system.

The number of VCPUs in virtual machine is based on the how many PCPUs are in the physical system. The number of VCPUs is limited to be no more that the number of PCPUs, this is mainly because VMM scheduler assign each VCPU of virtual machine to each PCPU. As the PCUPs of a physical system, they are shared between different VCPUs of different virtual machines. As a result of different virtual machines with different programs running on it, the VMM applies a timing scheduling to arrange the use of PCPUs, that is, instead of letting a virtual machine with heavy computation task keeps occupying the PCPU to finish its task, this strategy allows each VCPUto occupy PCPU in a certain time. Therefore, it is guaranteed that the PCPUs are fairly shared between virtual machines.

Virtual Machine, where the guest operating system runs on, equipped with several VCPUs, is usually considered as a symmetrical multi-processing (SMP) system by the guest operating system. Those VCPUs of a virtual machine are not always ready to work, while PCPUs are always ready to do computation. It works well when a sequential program is running on the virtual machine since all VCPUs are not necessarily appearing to be ready, which makes the CPU scheduling sequential and easy to be implemented as VMM schedules the VCPUs asynchronously.

Building a virtual cluster, based on virtualization technology, is one idea of efficiently utilizing the growing power of computation of nowadays CPU. This idea of virtual cluster will enable people to reasonable allocate existing physical resources of a PC so that the CPU resources can be effectively employed. A virtual cluster can be built by some virtual machines to provide same services as a real cluster provides, namely, a platform on which a parallel program can run. Each virtual machine that used to build virtual cluster should have same operation system and same hardware configuration, which is same as a real cluster. VMM tool is responsible for allocation of the CPU resources, which means the cluster member virtual machines and non-cluster members virtual machines are scheduled together under the control of VMM tools.

However, there's a problem in CPU resource scheduling when runs a virtual cluster on such a system. Unlike traditional sequential program scheduling, a cluster requires all its members to be online at same time. There are four steps to run a parallel program on a cluster. First of all, the master node will divide the job into several parts according to the number of available nodes. Secondly, each part of job will be executed on each node at same time, communications between cluster members conducted along the entire executing process. Thirdly, the result from each part will be collected to the master node, and master node will be responsible for producing the final result. As can be seen from these three steps, the second part, involving the parallel execution, is the most crucial step of the whole process of running a parallel program on a cluster. The cluster only works if all nodes are online, otherwise the cluster will be stuck as online nodes will wait to communicate with one or more offline nodes.

In virtual cluster case, this problem probably causes a huge waste of CPU resources, which dramatically decreases the performance of the virtual system. When one of the virtual machines within the cluster is scheduled a time slot on CPU, the virtual cluster will not work unless the other cluster members are also online. This is due to the virtual cluster requires communications between virtual nodes, namely each virtual machine within the cluster. If only one virtual node is brought online, the online virtual node will send communicating messages to other virtual nodes of the virtual cluster and will keep waiting the acks from other virtual nodes. During the period of waiting the acks from other virtual node, the online virtual machines actually will not do anything. As the result, the CPU usage will be very low, which means that this CPU time slot can be considered as being wasted. All virtual machines within a virtual cluster online at same time needs to be guaranteed.

To overcome the shortcoming of traditional sequential task scheduling approach applied on virtual cluster, co-scheduling [3][4], considered as one of the solutions, is introduced to manage the CPU scheduling to reduce the negative effect on performance of running parallel program on a virtual cluster built by virtual machines. This approach cooperates the current scheduling method of a virtualization system and solves problem mentioned above by adding additional control on scheduling virtual nodes online.

The basic rule of co-scheduling is that virtual machines which are not within the virtual cluster should give way to the virtual cluster in terms of CPU resources allocation if necessary when a virtual cluster is scheduled. It's straightforward when there are enough available CPU resources for scheduling virtual nodes online. However, if there are not enough CPU resources for scheduling all the virtual nodes online simultaneously, this simply indicates that some virtual machines that are not within the virtual cluster should be forced offline to make enough CPU resources for bringing cluster members online. These virtual nodes will execute a parallel program for same period and conduct some communications between each other. Once they run out of the time slot, they will be brought offline at same time.

This co-scheduling could be implemented by manually setting those virtual nodes online, Co-scheduling requires user that firstly knows whether it is a parallel program running on virtual cluster, and then using VMM tools to manually bring virtual machines within the cluster online at same time so that the cluster can work without a problem. However, manually setting virtual nodes online has some drawbacks as well. Firstly, it brings overhead by manuallycontrollingVMM tools, especially the number of virtual nodes is very large. Secondly, manually setting cluster member may not give the good efficiency of CPU usage. Thirdly, the policy of fairly scheduling may not be followed well by manually setting. These three problems can be solved by using automatically co-scheduling method with athreshold of how often scheduling the virtual cluster online forwithin the virtual cluster. Accordingly, such an optimal co-scheduling, following the principle of fairly scheduling, is required to give good efficiency of CPU usage.

This project is about to minimize the negative influences that running a parallel job on virtual cluster brings on to the efficiency of CPU usage and improve the performance of parallel program running on a virtualized system. As mentioned, traditional sequential program scheduling has obvious drawback on scheduling virtual nodes, an automatic co-scheduling is necessary to implement to reduce the negative effects those shortcomings bring on to the performance of virtualization system. This automatic co-scheduler should be able to bring the VCPUs of a virtual machine online if a parallel program running on the virtual machine, in other words, this optimized scheduler is making scheduling based on the needs of virtual machine. By automatically co-scheduling, unnecessary waste of CPU usage can be avoided and the efficiency of CPU usage can be increased.

# 2. Background

## 2.1 Virtualization

Virtualization technology was firstly applied on IBM Mainframe System in 1960s, and it became popular on System 370 series Mainframe System in 1970s. This kind of machines was able to create many virtual machines (VM) [2] with independent operation systems running on them based on physical hardware of a single machine via a platform called Virtual Machine Monitor (VMM). In recent years, virtualization technique brings obvious advantages into the area of business application due to the fact that multi-cores system, cluster computing system, grid computing system and cloud computing system have been widely adopted. It doesn't only reduce the cost of hardware, but also improves the security and reliability of computing system. The concept of virtualization have been introduced to people's daily life.

In computer science aspect, virtualization is defined as abstracting hardware resource of physical computer system, and this is not limited to the concept of virtual machine. For example, the virtualization of physical memory leads the invention of Memory Virtualization Technique, and this results in that software applications believe that they have own continuously available address space while the real fact is that the data of software applications are probably divided into several fragments or segments. In some cases, those data of software applications are even transmitted to hard drive, flash drive and some other kinds of external storage.

Virtualization technique includes three types of implementation:

* 1. ***Platform Virtualization***, which contraposes computer system and operating system virtualization.
  2. ***Resource Virtualization***, which aims to virtualize particular system resource. For example, memory virtualization, storage virtualization and network virtualization.
  3. ***Application Virtualization***, which contains simulation, analog and interpretation technology. The Platform Virtualization is the type that people usually discuss about. By applying a particular control unit (control program, also called Virtual Machine Monitor or Hypervisor), the physical features of a particular computer system will be concealed while an abstract, unified and stimulated computing environment (so called Virtual Machine) will be provided to users. The operating systems that are running on virtual machines are called Guest Operating System (Guest OS), and the operating system that runs the VMM is call the Host Operating System (Host OS). Some VMMs are able to separate from operating system and independently run on the computer system (e.g., ESX). The real system that VMs run on is called the Host.

Platform virtualization can be sub-divided into five categories:

1. ***Full Virtualization***. Full virtualization means the VM simulates the whole integrated hardware that are on the base layer of a host, which includes CPU. physical memory, system clock and external devices (e.g., USB devices). Those operating systems that are specifically designed for hardware are able to run on VMs without any change, and as well as some other system software. The communication between operation system and real physical hardware can be considered to be realized through pre-defined port. In full virtualization, VMM provides all ports by virtualizing whole integrated hardware of a computer system (The privileged instructions have to be stimulated at the same time). Full virtualization is expected to simulate the entire process of implementation of those ports. This kind of virtualization requires special supports from hardware, otherwise this process of simulation will be extremely difficult and highly complicated. The reason for this is that VMM has to run on with the highest priority to completely control the host system, while guest OS has to decrease its priority in the system so that it cannot execute privilege operations.
2. ***Paravirtualization***. This type of virtualization partially modifies the situation of privilege accesses of guest OS, at the result, the guest OS are able to directly communicate with VMM. In paravirtualization, some hardware ports are provided the guest OS in the form of software., and this can be achieved by Hyper-call ( This call is provided by VMM to guest OS, which is similar to system call). Since this doesn't produce additional process of simulation of exception and hardware, the paravirtualization is significantly improving the performance of virtualization.
3. ***Hardware-Assisted Virtualization***. This virtualization is defined as to realize highly efficient full virtualization assisted by the supports from hardware ( These supports are mainly from CPU). For example, if we have support from Intel-VT technology, full virtualization will be much easier to implement. First of all, the implementation environment of gust OS and VMM will be automatically and completely separated from each other. Moreover, the guest OS will have its own set of registers, which allows it to run with the highest priority. Intel-VT and AMD-V are the two hardware-Assisted Virtualization techniques available on x86 architecture.
4. ***Partial Virtualization***. VMM only virtualizes some hardware on the base layer of computer system in partial virtualization. The guest OS cannot run on a VM if the guest OS is not modified to adapt VMM. Consequently, some other applications need to be modified as well. In the history of virtualization, partial virtualization was a milestone on the road to full virtualization. It was used on the first generation time-sharing system CTSS and paging system on IBM M44/44X.
5. ***Operating System Level Virtualization***. In traditional operating system, all users' processes are naturally running on a same operating system, thus, the faults of kernel or application may affect other processes. Operating system level virtualization is a basic virtualization technique which is applied on server operating system. The kernel of server operating system creates several instances of virtualized operating system (kernel and library) to insulate different processes, and processes of different instances of operating system have no knowledge of each other. A well-designed virtualization software usually integrates more than one techniques. For example, VMware Workstation is a famous full virtualization VMM, it uses a technique very similar to paravirtualization technique. For paravirtualization, if it is possible to efficiently utilize the feature of hardware, the VM management will be simplified a lot, while keeps impressive performance.

## 2.2State of art of virtualization

**2.2.1 VMware**

*VMware* [5][6] is a well known virtualization software which is developed by VMware company. This software provides impressive hypervisor named GSX, which is used to manage the allocation of hardware resources for VM (As VMM introduced before). There are two types of hypervisor existing in VMware, which are type 1 and type 2 known as GSX. This type 1 hypervisor are able to directly execute on hardware, while the GSX is designed to operate along the type 3 hypervisor. Full virtualization will be realized by using VMware, that is, a full set of hardware resources will be provided to guest OS. This can be achieved by virtualizing all hardware in a computer system including CPU, video card, hard disk or network adapter, even portable devices (USB devices) can be virtualized. VMware provides several products aiming for different uses, which includes VMware workstation and VMware player.

**2.2.2 XEN**

*XEN* [7][8] is an open source virtualization hypervisor developed by Cambridge University. This virtualization hypervisor enables maximum of 100 full operating systems running on a single physical machine. The operating systems have to be modified so that they can be ran on Xen, which makes Xen doesn't need some specific support from hardware layer, and this can be used to achieve high performances on virtualization.

Xen adopts Independent Computing Architecture (ICA) to achieve high performance via a virtualization technique called para-virtualization, even on some architecture (X86) which is not well-compatible with traditional virtualization techniques. Comparing with traditional virtualization approaches which virtualize physical components via software, the guest permission of virtualized system is required for connecting Xen API, which is supported by Intel VT-X. This virtualization technique is now available to apply on NetBSD GNU/Linux, FreeBSD and Plan 9 system. Sun Micro System is also planning to migrate Solaris on Xen virtualization platform.

Another important feature is that Xen is capable of migrating virtualized system between multiple physical hosts at running time. During the process of manipulating, virtual machine's memory will be repeatly copied to target machine without suspending current work of the virtual machine. Before the virtual machine executes on the final destination machine, there exists a very short suspension around 60 – 300 ms so that the final synchronization can be executed, which can be considered as a kind of seamless migration between physical machines. Similar technique is employed to switch the execution of virtual machine. Firstly, a running virtual machine will be suspended and stored in a hard disk. After that, another virtual machine will be brought online to work. Finally, the first virtual machine will be resumed to work.

Currently, Xen is able to work with X86 system, and X86\_64 , IA64, PPC are the targets that Xen is planning to work with. Due to the stable performance and low resource consumption of Xen, many of world class hardware company including IBM, AMD, HP, Red Hat and Novell choose to support this virtualization technique, and many companies use this technique to build high performance virtualization platform. Nowadays, Xen is widely adopted in following areas:

1. ***Server Application Integration***: virtualizing multiple servers on a one single physical machine, and elastically migrating between physical machines, which vastly exploits the computation capacity of a physical machine, accordingly, live migration of application can be realized in server.
2. ***Software Development Test***: the low overhead enables users to flexibly build a system with multiple development platforms installed.
3. ***Cluster Computing***: comparing with managing every single physical machine that is within the cluster, virtual machine can be manipulated more flexibly and efficiently. Moreover, in the area of balancing workload, it shows a better control and separating.
4. ***Multiple System Configuration***: on the purpose of developing and testing, multiple operating system can be run simultaneously.
5. ***Kernel Development***: kernel testing can be implemented in the sandbox of virtual machine, and a particular machine for kernel testing becomes unnecessary.
6. ***Providing Hardware Support for Guest Operating System***: developing new operating system will benefit from the hardware support of existing operating system.

A Xen virtual environment consists of *Xen hypervisor*, *Domain 0*, *domain management and control*, *Domain U PV Guest* and *Domain U HVM Guest five parts*:

* + 1. ***Xen hypervisor*** is the most basic and lowest abstract layer of this virtualization software, which is in charge of round robin scheduling those VCPUs of virtual machines on the physical machine. Hypervisor not only abstracts the physical layer of a virtual system but also controls the execution of each virtual machine, However, the hypervisor is not responsible for the network, external storage and other IO function.
    2. ***Domain 0*** is a modified Linux kernel, which is special virtual machine running on the Xen hypervisor. It controls all physical IO resources, and meanwhile, it communicates with other virtual machines that are running on the same virtual system. All the Xen virtual environment require a running Domain 0 to launch other virtual machines. Domain 0 contains two drivers: Network Backend Driver (NB Driver)and Block Backend Driver to (BB Driver) provide supports the requests from network and local disk drive of other virtual machines. NB Driver is directly communicating with local network device, and all requests from Domain U virtual machines are handled by it. BB Driver is responsible to exchange data with local disks, and read /write data based on the requests from Domain U.
    3. All paravirtualization virtual machines running on the Xen hypervisor are called ***Domain U PV Guests***, and they are running modified Linux OS, Solaris, FreeBSD. Comparing with Domain U PV, Domain U HVM Guests stand for all full virtualization virtual machine, and standard operating system and unmodified operating system can be ran. Domain U PV Guests are not allowed to directly visit physical hardware layer, and are also expected to know other virtual machines running on the same machine.
    4. ***Domain U HVM Guests*** are different from Domain U PV Guests, they have no knowledge about sharing CPUs resources and existences of other virtual machines running on this machine. PV guest also includes two drivers specifically to network and hard disk, PV Network Driver (PVN Driver) and PV Block Driver (PVB Driver.). HVM Guests haven't been installed on virtual machines, however, for every single launched HVM Guest, there is a special daemon ----- Qemu-dm. All requests of visiting internet and local disk from HVM Guests are supported by the Qemu-dm. HVM should be initialized so that the HVM Guests can be added to HVM Guests Xen Virtual Firmware are sued to simulate the BIOS to start the operating system.
    5. ***Domain management*** by a set of virtual machine management tools which provide access to control and manage those domains. Many Linux daemons are defined as management and control tools of domains by the open source software. These services which support the management and control of entire virtual environment are storing in Domain 0. Control of virtualization environment can be used to implement manual adjustment according to users. Management tools of virtualization will be detailed introduced later.

**2.2.3 Kernel- based Virtualization Machine**

*Kernel-based Virtualization Machine (KVM)* [9] is an virtualization module of open source operating system.It is default virtualization software in main Linux distributions, and it 's integrated into Linux after Linux 2.6.20. KVM uses Linux own scheduler to manage the resource allocation. Hence, it has less source code comparing with XEN. KVM uses Hardware-assisted Virtualization technique, that is, KVM is a full virtualization based on hardware. As the result, the supports from hardware (For example, Intel VT techniqueor AMD V technique) are necessary. KVM has small scale code than those virtualization software which have their own schedulers. KVM has become one of the main VMMs in academic area.

**2.2.4 Hyper–V**

*Hyper-V* [10] is a virtualization software developed by Microsoft. The aim of Hyper-V is to provide a more familiar and better cost-effectiveness to large group of Windows users, and the optimization of basic infrastructure and improvement of server utility are also expected to be achieved. Hyper-V adopts the architecture of micro kernel, which balances the safety and performance. The hypervisor on the base layer of Hyper-V runs under the highest priority, which is called ring -1 by Microsoft, while the guest OS kernel and guest OS drivers of VM runs with ring 0, and all other applications run with ring 3. The complicated BT (Binary-priority Translation) becomes needless to this architecture, which makes Hyper-v further improves the safety and stability of the virtualized system.

**2.2.5 VirtualBox**

*VirtualBox* [11] is an open source virtualization software developed by Innoteck Group. It provides various virtualization operating system including Windows ( all versions from Windows 3.1 to Windows 8, Windows2012), Mac OS X( 32 bit version and 64 bit version), Linux (2.4 and 2.6), openBSD, Solaris, IBM OS2 even Android. Comparing with other virtualization software, Virtual Box has full supports to Remote Desktop Protocol (RDP), Internet Small Computer System Interface ( iSCSI ) and Universal Serial Bus ( USB ). Virtual box is able to support USB 2.0 hardware devices so far if the particular package called VirtualBox Extension Pack is installed . This virtualization software also supports Clone VM, and the VM memory limit is improved to 1 GB so far.

## 2.3 Computer Cluster

Computer cluster, built up by tightly connecting a group of standalone computers, is a computer system that works under highly dependent cooperation between the computers within the cluster. It can be considered as one computer in a manner. In a cluster system, single computer is called node, and nodes are usually connected via Local Area Network (LAN), however, other connecting method is also possible[12]. Cluster Computer is usually used to improve the computing speed or computing reliability of standalone computer[13]. Cluster computer is becoming popular nowadays due to its high performance in computation. There are various cluster providing different level computing service, a normal cluster usually consist of a small number of cluster such as a university cluster while a supercomputer is composed by a large number of nodes like IBM's sequoia[14].

# 3. Literature Review

CPU scheduling has been exited since the first computer came into our society. Basically, the scheduling can be expressed as allocating CPU resources for the tasks which are expected to be executed. The CPU is of course, one of the primary computer resources[15]. This core resource of a computer system takes all responsibility of computation. Scheduling is one of the fundamental functions of any operating system, since almost all computer resources are scheduled before use[16]. CPU scheduling is important because it has a big effect on resource utilization and the overall performance of the system [17]. Due to this reason, reaching in developing new scheduling mechanism has always been popular and interesting in computer science society. The improvement of scheduling mechanism is considered as one of solutions to radically resolve the problem of efficiency of computer system.

The first-come-first-serve (FCFS) is the simplest scheduling mechanism. The basic idea of this scheduling algorithm is that the scheduler will schedule CPU time slot to the job which is submitted first. In effect, processes are inserted into the rail of a queue when they are submitted[18].This primitive scheduling algorithm doesn't take the fairness of scheduling into account. Each task will be allowed to occupy CPU resource as long as it completes. As the result, the other tasks in the queue will have to wait for a long time in general. One process with longest burst time can monopolize CPU, even if other process burst time is too short. Hence, throughput is low[19].

Comparing with FCFS algorithm, the Round Robin is a scheduling algorithm which fairly allocates CPU resource to tasks in the queue. The scheduler goes around this task queue, allocating the CPU to each process for a time interval of assigned quantum. New processes are added to the tail of the queue[20]. The CPU time slot allocated to each task has huge impact on the performance, an either too short or too long harms the efficiency of the computer system. Short time slot makes the CPU efficiency decreases since too many context switches happened in scheduling, and the long time slot also causes degradation of performance due to the long waiting time for tasks.

Priority scheduling uses the level of priority as a standard to check if a process is qualified to be scheduled. The task with low priority may be interrupted to give way to the task with high priority level. The obvious drawback of this scheduling algorithm is that there is a high possibility that a task with low priority is stuck as the result of those tasks with higher priority are always schedule first. The waiting time gradually increases for the equal priority processes[21].

Although there are advantages and disadvantages, these algorithms seem to reach the expectations in a early computer systems which are equipped with a CPU with one core. In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled [22]. In a multi-core CPU system, scheduler is expected to sufficiently handle some parallel programs which are designed to exploit the potential power of computation of multi-core CPU. The virtualization is one of the areas which has high requirement in the efficiency of scheduling CPU resource. HE recently resurgent research in server virtualization has fueled interest in using this technology to design consolidated hosting platforms[23].

In order to maximum the utilization of the hardware resources and minimize the capital costs by reducing physical infrastructure, Virtual Machine (VM) management has become an important research field of virtualization technology, and the scheduling of virtual machines on a physical host machine is crucial for the throughput of a system and thus affects the overall system performance[24]. There are several virtualization platforms, Xen is the most popular one in the academic area as it is an open source platform. Xen has several schedulers inside including BVT, SEDF and Credit scheduling algorithm, which provides a good performance in sequential scheduling.

BVT [25] algorithm is a fairness priority scheduling algorithm proposed by Kermeth J. Duda in 1999. This algorithm divides time into real time and virtual time two kinds. Many fair sharing algorithms have the concept of virtual time at their heart [26] [27] [28] [29] With BVT scheduling, thread execution time is monitored in terms of virtual time, dispatching the runnable thread with the earliest effective virtual time (EVT)[30]. Real time indicates the time information in the hardware timer, while the virtual time is obtained by a particular calculation on real time. The VCPU with the earliest effective virtual time will be scheduled every time slot. This scheduling algorithm considers the run time and interactive programs to guest operating system, and it allows such programs and guest operating system borrowing some time slices, in other words, borrowing the future time allocated to them within some set-up limits. These borrowed virtual time slices can only be some virtual time slices of the present real time slices, not the virtual time slices of next real time slices. This scheduling behavior is similar to weighted fair queuing [31] and start-time fair queuing (SFQ) [32]. At the initialization , every single VCPU will be assigned a weight which indicates that the CPU share that a VCPU can obtain, and the VCPUs will be fairly scheduled according to their weights. The virtual system uses virtual time and effective virtual time to record the execution status of each VCPU. These two types of virtual time can be obtained by applying the formula given below:

*Ai = At +t/wi* (1)

*Ei ← Ai – (Wrap?Wi:0)* (2)

In the formula (1) and (2), t indicates the real execution time of VCPU (calculated by real time), *wi* is the weight value of a VCPU, *Ei* shows the effective virtual time. Ai in the first part of this formula means the real virtual time, Wrap stands for the offset mark which indicates if the VCPU is allowed to execute in advance, *Wi* is the time that VCPU can be executed. Warping a task in virtual time subtracts a constant from its timestamp, which causes it to run sooner within a framework of long-term fair sharing[33].

BVT is a preemptive working-conversing (WC) mode algorithm. This algorithm is able to adjust the entire BVT algorithm via the value of *Wrap*, and this leads the executing slice of VCPU to be brought forward, namely, The VCPU borrows limited time slices from its future allocated time to get a higher scheduling priority. Another feature of this algorithm is that the value of *Wrap* of each VCPU can be limited by adjusting *Li* value and *Ui* value, and the frequency of manipulating *Wrap* can be controlled, which is designed to prevent from each VCPU over-borrowed virtual time.

The advantage of BVT scheduling algorithm is that this algorithm focuses on fairly scheduling CPU resources to every single guest operating system, and the time for waiting scheduling the same guest operating system second time will be limited within one real time slice. The task dispatch latency can also be reduced by applying this algorithm, which has been experimentally proved. These featured satisfy those applications which requires low latency, and has a good performance in scheduling operating system which require real time execution. The overall consumption of this algorithm in an environment of single CPU and multiple CPUs is good. The work [34] also tries to test that BVT can be used for a wide range of applications.

However, the shortcomings of BVT algorithm are as obvious as its advantage. Firstly, this algorithm doesn't support non-working -conversing, which means a domain will be allocated entire CPU resources when the domain is launched. The lack of NWC-mode in BVT severely limited its usage in a number of environments, and led to the introduction of the next scheduler in Xen [35]. Users can not set a limitation for the ratio of CPU resources that can be used by the domain. Secondly, every single guest can only borrowed its own time slice but cannot take time from other guest operating system's time, namely, when the ration the allocation of time slice of a single domain, this ratio will not be changed until next round of allocating the ratio of time slice.

SEDF[36], proposed by C. L. Liu in 1973, is another important scheduling algorithm in virtualization. This scheduling algorithm emphasizes the concept of Earliest Deadline First (EDF). SEDF uses real-time algorithm to deliver performance guarantee[37]. At the beginning of initialization of Xen, each PCPU is setup with a deadline as the reference of scheduling. During the scheduling process of VCPU, the scheduling program firstly allocates time slot to the VCPU with the earliest deadline. SEDF algorithm is considered as a dynamic priority scheduling algorithm, the priority level of a VCPU keeps changing along with the changes of its absolute deadline. At any moment, the VCPU with the earliest deadline has the highest priority. SEDF orders the domains in the run queue according to their deadlines and executes the domain with the earliest deadline[38].

SEDF, which is implemented by famous EDF scheduling algorithm, is a dynamic-priority real-time scheduler[39].There are several key features of SEDF scheduling algorithm. Firstly, it assigns every guest operating system a tripe (*s, p, x*) where s and p mean the guest operating system should run at least s ms within a period of p ms, and x indicates if guest operating system is allowed to take the rest of time within p after s. Secondly, every guest operating system has a “latest scheduling time”. For instance, within 0-200 ms this time slice, the guest operating system runs at least 150 ms, then the latest executing time is 50 ms. Thirdly, the SEDF algorithm chooses the earliest guest operating system with the “latest scheduling time”. Fourthly, if Guest 1 is still running, Guest 2 operating system will be scheduled when its time slice comes. Fifthly, SEDF supports WC and NWC (non-working-conserving) mode, which enables the virtual system allocates partial CPU to a guest OS.

An obvious advantage of SEDF is that it controls the priority of each guest operating system by configuring the configuration parameters of each guest operating system, in practice, the priority of guest operating system decreases with the parameter *p*. This algorithm has very good efficiency and is easy to be implemented in reality. The theoretical computation and induction are simple to be done, which is quite different from most of scheduling algorithms as the scheduling algorithms are focusing practical implementation more. SEDF provides great supports to some applications which require good performance in real time computation. When the workload balance is low, the usage of CPU can reach a very high level.

However, the drawbacks of SEDF algorithm are as evident as its benefit mentioned above. Firstly, in SEDF algorithm, the scheduling parameters of a VCPU cannot be changed according the present situation of execution once they get initialized. SEDF is designed to assure the most urgent tasks can be done on time, and this is realized by adjusting the priority level of task taking into account the urgency of satisfying the task's deadline. When the workload is low, this algorithm is extremely effective. However, when the workload of CPU is high, this approach will cause massive missing the deadline (MTDs), as the side effect, many processes may not be dealt with on time due to miss the MTDs [40]. These MTDs may even cause that the CPU spends too much time on the process scheduling. Under this circumstances, SEDF has even worse performance than FIFO which is the most basic preempted scheduling algorithm.

According to analysis, the performance of this algorithm decreases significantly when the workload balance is over 50%. The key factor of this phenomena is how to sort the processes by the latest scheduling deadline. This is because the scheduling deadline of process keeps changing during the time of execution, and this causes the problem that the processes cannot be sorted prior to scheduling. In fact, the priority of every process is computed again before the end of each process. Secondly, this algorithm only works for single CPU, and there is no control of balancing the workloads of multiple CPUs. For example, *domain A* gets 70% of *CPU 1*, and *domain B* gets 70% of *CPU 2*, if there is *domain C* which requires 50% of CPU resources. This *domain C* won't be able to be launched, even though the rest resource of *CPU 1* and *CPU 2* is enough for *domain C*. However, the rest resource of each single CPU is not enough for *domain C*, as the result, *domain C* will be waiting until one of the single CPUs has enough CPU resource for *domain C*.

Credit is Xen’s latest proportional share scheduler featuring automatic load balancing of virtual CPUs across physical CPUs on a symmetric multiprocessing (SMP) host[41 ]. Credit scheduling algorithm sets a tuple (*weight, cap*) for each guest operating algorithm. The weight's ratio between guests operating system decides each operating system 's proportion of time slice of CPU, while the cap value determines the upper bond of CPU time slice that a guest operating system can use. For example, setting a cap value to 60 simply means that a guest operating system can only use all time slices of 60% of a physical CPU in maximum, while setting a *cap* value to 100 indicates that a guest operating system can have all time slices of a single physical CPU in maximum. Credit scheduler is a non-preemptive works fairly in both environments WC (work conservative) and NWC (non-conservative) mode it does automatic load balancing in multiprocessors[42].

Credit algorithm divides all VCPUs into two queues: under queue and over queue. UNDER indicates that a VCPU still has credits to use, while OVER means that the VCPU has exhausted its credits [43].Only VCPUs in the under queue can be scheduled. At the beginning, all VCPUs are in the under queue, every domain has its own credit which is its corresponding value of weight. Every time the VCPU is scheduled CPU time slices, the relevant credit of the domain containing this VCPU will be reducing. When the credit of a domain becomes negative, this VCPU will be moved to queue of over. The credit scheduler will add initial credit value to every domain when all VCPUs are in the over queue. In the same way, the schedule will also add the initial credit weight of a VCPU to get a new weight. As the result, all VCPUs will be moved to under queue.

The VCPU, in front of queue, priority set to under, will be scheduled first by the system. When a PCPU is idle or no VCPU in the its under queue, this PCPU will seek a runnable VCPU from other PCPU's under queue. The guests are arrange in the run queue according to their status and not the credits [44].At present, most of researches on credit scheduling algorithm are focus its influence on the system performance. For example, Ludmia Cherkasova analyzed the read/write speed of virtual machine, network throughput and the precision of CPU allocation [45]. Diego Ongaro focus on the performance of effects that credit scheduler brings on the I/O performance of Xen virtualization.

Credit scheduling algorithm also has some shortcomings. First of all, this scheduling algorithm cannot guarantee the real-time scheduling. In some programs requiring high response speed, event responding delay is tightly related to the position of its queue, this cause the fact that responding delay to those programs which requires low response time is generally very long. Secondly, the scheduler is fairly sharing the processor resources only by approximation[46]. Thirdly, credit scheduling algorithm has a simple strategy of balancing global workload. The CPU is highly possible to be idle due to the reason that the process is not ready. The error rate of CPU allocation caused by global resource distribution is high, which make the global resource management complicated.

The common drawback of all these three is that no concentration put on parallelization of virtual machines in the same host. All existing scheduling algorithms in Xen focus on sequential scheduling, which is traditional and basic scheduling concept. In a non-virtualized environment, all CPU cores in a machine are physical and run online simultaneously, so the locks held by some CPUs will be released soon, and a lock requester is aware of lock releasing as soon as possible [47]. However, we expect a scheduling mechanism to have a good ability of handling parallel jobs and synchronized jobs in a virtual system. For a synchronized program, the job of each virtual machine is monitored locally in terms of domain. This is because the synchronized task is more about communications between processes within on virtual machines. For a parallel program, a global coordinator is required as the communications are between different virtual machines.

Co-scheduling is a popular scheduling mechanism to solve the problem of communications. Researchers have classified three types of co-scheduling: explicit co-scheduling, local scheduling and implicit or dynamic co-scheduling[48]. Local co-scheduling is adopted by some mechanisms proposed as solution to synchronized problem as each single virtual machine is responsible for scheduling its own VCPUs. Some works have been proposed based the local co-scheduling mechanism such as [49]. For a parallel job which requires communication between virtual machines, explicit co-scheduling is more appropriate. Explicit co-scheduling ensures that the scheduling of communicating jobs is coordinated by creating a static global list of the order in which jobs should be scheduled and then requiring a simultaneous context-switch across all processors[50]. This co-scheduling mechanism satisfies the meaning of global monitoring as mentioned before.

# 4. Design Strategies

This chapter introduces the process of co-scheduling strategy designing. Possible strategies that are designed for co-scheduling will be discussed in chapter. There are three parts of this chapter. Firstly, the problem of current scheduling algorithms in Xen will be stated and analyzed. Secondly, two possible strategies will be detailed discussed. In this part, I will go through these two possible strategies to see the advantages and disadvantages of these two strategies. Thirdly, the final strategy based on the strategies discussed in previous part will be given.

## 4.1 Problem analysis

The problem of current scheduling algorithms existing in Xen is that there is no particular support to the virtual cluster. The main reason is that these existing scheduling algorithms are designed as sequential scheduling algorithms. These scheduling algorithms work well with traditional sequential scheduling as they are responsible for scheduling one virtual machine each time, while virtual cluster requires a scheduling mechanism that is able to schedule multiple virtual machines simultaneously. The figure 4.1 below indicates the difference between two concepts. This is due to that the cluster requires all its member online at same time so that the communications between its members can be conducted.

Scheduler

Scheduler

Task

Task

Task

Task

Task

Task

Figure 4.1 Two concepts of scheduling, the left figure illustrates sequential scheduling and the right figure indicates the required scheduling mechanism.

## 4.2 Possible strategies

**4.2.1 Proactive strategy**

To satisfy the conditions of making a cluster work, a scheduling mechanism should be able to bring multiple virtual machines online. When there are enough CPU resources for a virtual cluster, those virtual machines within the cluster will be naturally assigned sufficient CPU resource, and then the whole cluster will be working as normal. However, if virtual machines that are not within the cluster occupy too much CPU resource, ideally, the scheduler should be able to bring the entire cluster online when one of the cluster member was called. In other words, the virtual machines which are not within the cluster should be ready to give way to the virtual cluster at any time when one of the cluster members is called, and the scheduler should be in charge of when and how to allocate the CPU resource to the cluster. Therefore, the co-scheduler should be designed as compliment to those existing scheduling algorithm to provide particular supports. The basic strategy is shown in figure 4.2.

This co-scheduler is expected to be able to cooperate with existed algorithms in Xen. According to the conditions mentioned above, this co-scheduler should contain three parts. Firstly, this co-scheduler should detect the current situation of Xen virtual system and retrieve most of up-to-date information. Secondly, the co-scheduler keeps checking the status of virtual cluster. If the one of the cluster members is scheduled online, this co-scheduler will be forcing those virtual machines that are not within the cluster off line to make sure there are enough CPU resources available to the rest virtual machines within the cluster. Thirdly, once the entire cluster has been scheduled, the co-scheduler will do a double check to make sure the entire cluster is running on the host machine. If the third part indicates that the cluster is not running correctly, it will make the other virtual machines waiting until the cluster becomes working correctly. Finally, the co-scheduler will re-allocate CPU resources to non-cluster members in the queue. The figure 4.3 shows how proactive strategy works.

Scheduler

Task

Task

Task

Task

Co-scheduler

Task

Figure 4.2 The Basic strategy

However, some problems may exist in the strategy described above according to different scenarios of virtual system. Even though the co-scheduler is considered as a tool to provide extra support to virtual cluster so that the entire virtual cluster can be brought online, it still has to stick on the principle of fairly scheduling. Obviously, the strategy described above may naturally break the fairness that existing algorithms attempt to reach, especially when the size of cluster is more than the number of non-cluster members. The virtual cluster might obtain much more CPU resources than other virtual machines.

C

C

C

C

NC

NC

NC

NC

Scheduler

Co-scheduler

Hypervisor

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

Figure 4.3 Proactive strategy

For example, there is an host machine with 8 PCPUs, 4 non-cluster virtual machines (*nc1 nc2 nc3 nc4*) and a virtual cluster (*c1 c2 c3 c4 c5 c6*) with size of 6 installed. Non-cluster virtual machine has one CPU time slice to run Each time it gets scheduled, while virtual cluster actually gets 6 time slices to run each time it gets scheduled. Comparing with the number of non-cluster members, the virtual cluster has more chances to be scheduled. When the existing schedulers do the allocation of CPU resources, they consider each virtual machine within the cluster (*c1 c2 c3 c4 c5*) is as same as non-cluster virtual machines (*n1 nc2 nc3 nc4*). In other words, the chance of the virtual cluster being scheduled is 6 times of each one of non-cluster member virtual machines. In such a virtual system, every time a non-cluster virtual machine scheduled 1 CPU time slice, the virtual cluster will be scheduled 36 time slices in total, and 6 time slices for every member of the virtual cluster. This will make the scheduling system not fair at all, which is what the co-scheduler should avoid

The example described above states an obvious drawback of the co-scheduling strategy. This strategy is considered as a proactive choice, which sacrifices the CPU time slices of non-cluster member virtual machines to guarantee that a cluster can be scheduled. In real world, non-cluster virtual machines and virtual cluster are expected to co-exist in a same host machine, which requires the performance of non-cluster member virtual machines should be guaranteed as well. Therefore, the proactive approach will not work in terms of the performance of non-cluster virtual machines.

**4.2.2 Reactive strategy**

According to the fact discussed above, the strategy of the co-scheduler should be adjusted so that the CPU resources allocated to non-cluster virtual machine can be guaranteed. One concept satisfies this philosophy is that the co-scheduler can just leave the virtual cluster to wait until there is enough CPU resource for bringing the entire cluster online. The con-scheduler will not proactively interrupt the execution of non-cluster members, as the result, the allocation of CPU resource to non-cluster members can be guaranteed. The strategy can be illustrated in figure 4.4. The benefit of this strategy is that the virtual cluster can be scheduled without sacrificing the performance of non-cluster member virtual machines, which solves the problem stated above.

This strategy basically concentrates on the currently available CPU resources rather than the virtual cluster is scheduled. The co-scheduler will monitor the available CPU, and once the number of available CPU reaches the size of the virtual cluster, it will schedule the virtual cluster online without bothering the co-existing non-cluster member virtual machines. If there is not sufficient number of PCPUs available to the virtual cluster, this co-scheduler will not do anything and let the virtual cluster waiting for enough PCPUs available.

The reactive strategy is considered as a kind of reactions of virtual cluster to other virtual machines. The available CPU does not only mean the PCPUs that are not occupied by other non-cluster virtual machines, but also mean those PCPUs which are taken by non-cluster virtual machines but in an idle status. Thus, this strategy needs to check both the number of PCPUs that are not used by virtual machines and the number of PCPUs that are occupied by other virtual machines but in an idle status. The components of this strategy are very similar to the components of proactive strategy discussed above. There are basic information collecting part, available PCPU checking part which includes checking the status of non-cluster virtual machines and PCPUs which are not in use, virtual cluster status confirming part and no-cluster virtual machines reallocating part.

Reactive scheduling strategy avoids interruption of non-cluster virtual machines to guarantee the CPU resource allocated to these virtual machines and schedules virtual cluster when it is available, however, this approach also has a vital shortcoming. In this strategy, the co-scheduler schedules virtual cluster only if there is sufficient number of PCPUs available, otherwise the virtual cluster will keep waiting until the number of available CPU reaches the number of size of the cluster. This may cause long waiting of virtual cluster, in some extreme cases, the virtual cluster may never get scheduled if those non-cluster member virtual machines are always busy. This problem will seriously harm the performance of virtual cluster, and it also disobeys the principle of fairly scheduling.

For instance, there is a host machine with 8 PCPUs, 6 non-cluster virtual machines (nc1 nc2 nc3 nc4 nc5 nc6) and a virtual cluster with size of 4 installed on the host. If the 6 non-cluster virtual machines keep running on the host machine, the virtual cluster will be waiting there until the available CPU resources are enough for bringing the entire cluster online. However, the chance of getting 4 available PCPUs are very rare since those non-cluster virtual machines are busy, each of these virtual machines will be re-scheduled after a CPU time slice. Therefore, the virtual cluster has very low possibility of being scheduled, it's highly possible that this virtual cluster keeps waiting for a long time and will never be schedule, which is not what the co-scheduler is expected to do.

C

C

C

C

NC

NC

NC

NC

Scheduler

Co-scheduler

Hypervisor

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

Figure 4.4 Reactive strategy

**4.2.3 Compromised strategy**

To make the co-scheduler working as we expect, both of two problems mentioned above need to be solved. As can be seen, both of proactive and reactive strategy partially solve the problem. The proactive approach takes the needs of virtual cluster into account, while the reactive method considers the situation of available CPU resources. However, in real world, both non-cluster virtual machines and virtual cluster are expected to be working well. This requires a strategy that considers both virtual cluster's needs and the available resource that a host machine can provide in order to make the co-scheduler working as expected. Considering the positive side and negative side of both proactive strategy and reactive strategy, a compromised strategy based on those two strategies can be obtained to do the expected task mentioned above.

The compromised strategy is simply a strategy which compromises the proactive strategy and reactive strategy. In other words , this strategy will have the characteristics of both proactive method and reactive method. This strategy will monitor the available CPU resources like reactive strategy, while it checks the CPU resource requirements from virtual cluster. Both these two areas will be taken into account if it is a good time to bring the virtual cluster online. The basic process of this strategy is working in four steps. Firstly, two thresholds should be specified, one for the number of available PCPU, the other is for the number of virtual cluster requiring to be scheduled. The first threshold is easy to understand, and the requirement of virtual cluster simply means how many times the virtual cluster member is scheduled. Secondly, both number of available PCPUs and the resource requirement of virtual cluster will be monitored. A counter is set to record the times that virtual cluster member being scheduled. Thirdly, interrupting actions will be taken according to the status of available CPU resources and running state of virtual cluster, so that the virtual cluster can be brought online. Fourthly, the non-cluster virtual machines will re-scheduled after bringing virtual cluster online, and the counter of virtual cluster member being scheduled will be set to zero. The figure 4.5 shows the concept of this compromised strategy.

C

C

C

C

NC

NC

NC

NC

Scheduler

Co-scheduler

Hypervisor

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

PCPU

Figure 4.5 Compromised strategy

The most important factor of this compromised strategy is the values of two threshold. These two values have vital influence on how often the virtual cluster is scheduled. If the threshold of available CPU number is set to a large number, then the co-scheduler will make virtual cluster waiting until the number of available PCPU number reaches the threshold; vice versa, if the co-scheduler expects a small number of available PCPU number, then virtual cluster will be brought online more often as the threshold is easy to reach. The threshold of virtual cluster requirement works in the same way, if this threshold is set to a small number, the co-scheduler will count how many times the virtual cluster member is scheduled, the virtual cluster will be brought online if the threshold is reached; if the threshold is set to a large number, virtual cluster will be waiting and will be scheduled until the number of virtual cluster member being scheduled reaches the threshold.

This strategy overcomes the drawback that the virtual cluster takes too many CPU time slices in proactive strategy, and also solves the problem that the virtual cluster waiting too long for enough available CPU resources. By setting the thresholds, the virtual cluster scheduling can be controlled according to the practical situation. The frequency of scheduling virtual cluster online can be manually managed, which gives a great flexibility in adjusting the co-scheduler. Therefore, over waiting and over scheduling can be avoided, which results a good performance of virtual system in terms of scheduling virtual cluster and non-cluster member virtual machines.

An example is given to illustrate the effectiveness of this compromised strategy. A host machine with 8 PCPUs, and there are 6 non-cluster virtual machines (*nc1 nc2 nc3 nc4 nc5 nc6*) and a virtual cluster with size of 4 (*c1 c2 c3 c4*) installed. The threshold of number of available CPUs is set to 3, and the threshold of number of virtual cluster member being scheduled is set to 2. The co-scheduler will detect the number of available CPU while monitors the number of virtual cluster member being scheduled. If there are 3 PCPUs available, then the co-scheduler will force all non-cluster virtual machine offline, at the same time, the virtual cluster will be scheduled. If there are only 2 PCPUs available and the virtual has been scheduled for 2 times, then the non-cluster member virtual machines will be force to offline, and the virtual cluster will be scheduled online. After that, the counter will be set to zero.

Each threshold is able to set the proportion of CPU resources allocated to non-cluster virtual machines and virtual cluster members. This enables users to manually adjust the weight of scheduling CPU resources to each side. The formula of calculating the proportion of CPU resource of non-cluster members is given below:

(3)

where

(4)

In the formula (3), *NC* is the number of non-cluster virtual machines, *C* indicates the number of virtual machines within the virtual cluster. The parameter n stands for the scheduling weight of virtual cluster, which can be obtained in formula (4). In formula (4), the *Th\_scheduled*\_*time* indicates the threshold of cluster members being scheduled, and the meaning of *C* is as same as of that in formula (3). The proportion of virtual cluster can be calculated via replacing the *NC* by *C*.

In practice, if the users prefer to specify the proportion of CPU resources allocated to virtual cluster or individual virtual machines, this can be achieved by transforming the formula into a function of n given the relative proportion, and the transformed formula is given below:

(5)

The formula (3) leads another formula to calculate the threshold value according to the proportion that the use expects. The formula (4) is shown below:

(6)

These formulas are based on the fact that the other threshold is fixed, namely, the threshold value of available CPU is fixed. The assumption is that there's no additional effects from the threshold value of available CPU. Therefore, if the threshold value is not zero, the effect of the formulas stated above will be influenced.

The effect of threshold of available CPU resource is not given here as the available CPU resource is a passive valve which only works if there's enough CPU resources. This threshold is designed to prevent the scheduler from wasting CPU resource. In practice, this threshold will not be as effective as the threshold of virtual cluster member being scheduled as all PCPUs should be in a busy mode when the workload of virtual system is heavy. In other word, there may no idle PCPUs when the virtual system runs with full workload. Therefore, we just state the effect of threshold of virtual cluster member being scheduled, and the threshold of available CPU is assumed to have no practical effect on the scheduling process.

# 5. Implementation of the co-scheduler

A prototype co-scheduler has been implemented based on the strategy introduced in previous chapter. In this chapter, the details of implementation will be discussed in third sections. In the first section, we will state how *domain 0* controls all other domains via communicating with Xen hypervisor. This section is very important to the implementation of the co-scheduler because the co-scheduler can be considered as a tool assisting domain 0 in managing all other domains, namely, virtual machines. The second section introduces a series of tools that are used in managing domains. In this section, the features of these tools and what these tools can provide will be discussed. The third section will discuss the system environment for this implementation. The fourth section describes the detailed work of the implementation of this co-scheduler. In this section, the APIs used for implementing this co-scheduler will be listed and explained, the structure of this co-scheduler also will be given.

## 5.1 Domain management in Xen

To implement the co-scheduler of Xen, firstly we need to know how Xen manages the virtual system. In a Xen virtual system, the Xen hypervisor is in charge of hardware resource management. The control domain (*domain 0*) manages all other domains (*domain U*) via communicating with hypervisor, and hypervisor will allocate adequate resource following the order from *domain 0*. In other words, the control domain is not directly communicating with *domain 0*. Therefore, the concentration should be put onto the method of communication between domain 0 and Xen hypervisor.

In Xen, *domain 0* communicates with hypervisor via a particular component called tool stack. This tool stack contains several possible tools that can be used to communicate with the hypervisor. Each one of this tools provides its own API to control domain, which leaves the application layer a great flexibility of communicating with hypervisor. For example, if a program specifies a particular CPU to run on, all this program needs to do is to use the API provided by different tools in tool stack, and then the communication will be between the hypervisor and control domain will be conducted, after that, the hypervisor will do the allocation of CPU to the process. The figure 5.1 shows the structure of domain management in Xen.

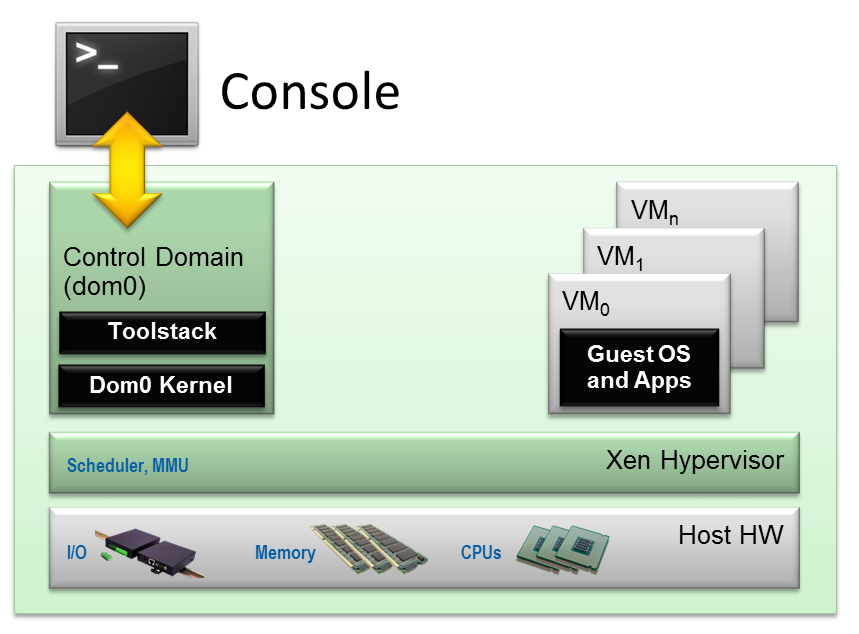


Figure 5.1 The structure of domain management in Xen

One feature of Xen should be mentioned is that between the tool stack and Xen hypervisor, an additional library Libxenlight has been introduced since Xen 4.2. In the earlier versions of Xen, the tool stack was difficult and expensive to used to manage Xen. The reason for this problem is that all of the tools in tool stack are provide certain services, and thus all of them will be used in managing those common operations in the low layer of Xen. As the result, it causes a lot of maintaining problem of Xen, for instance, code duplications , inefficient, bugs. and the Libxenlight is brought to solve a practical problem of Xen. Libxenlight is a compact C library dealing with those common operations in low level of Xen hierarchy. This small library benefits from easy handling from understanding, modifying and extending these three areas. This library is shared between all tools in the Xen tool stack and providing API's for those tool stacks to use. This library is considered as the basis of Xen operations in low level.

There are several tools in tool stack used by Xen given below, all of them provide services to *domain 0* to manage Xen. The functions provided by these tools may overlap a little bit with each other, however, some certain functions used by Xen are from different tools. Most of these functions provide APIs to *domain 0*, which can be manipulated by users to control the virtual system. Some APIs are only available to the *domain 0* to protect the entire execution of Xen.

## 5.2 Domain management tools

**5.2.1 Libvirt**

Libvirt is one of most important libraries in scale-out computing area. This library provides a series of APIs that is unknown to virtual machine monitoring program to securely manage those guest operating systems running on the host machine. Libvirt itself is composed based on an abstract concept. The main function of Libvirt is that it supports virtual machine monitoring program by providing a series of API that are used in general operations. This library was designed to be virtual machine management API for Xen, with the development of virtualization, the Libvirt has been modified to provide supports to the virtual machine monitoring programs. Libvirt 10.03 is used in this project.

**5.2.2 Xm**

Xm is a virtual machine management tool specialized in managing virtual machines in Xen virtual system. This tool is written in C and contains all necessary parts to manipulate the status of guest operating systems. All its APIs are provided in a header file *xenctl.h*. The provided header file can be used in the implementation of the co-scheduler.

## 5.3 Implementation Environment

**5.3.1 System environment**

Ubuntu 13.04 is adopted as the operating system for this implementation. This version of Ubuntu has the up-to-date support to Xen.

**5.3.2 Programming language**

This project uses C as the programming language for two reasons. First of all, most of Xen is written in C, using C guarantees that the co-scheduler will be fully compatible with Xen. Secondly, the APIs provided by packages mentioned above is C binding.

**5.3.3 Virtualization platform**

Xen 4.2 is installed for this project, the Xm management tool is included in Xen 4.2. This co-scheduler is designed to be fully compatible with Xen.

## 5.4 Implementation

The implementation of this co-scheduler includes four modules. The first module is initialization module, which runs the initialization for the co-scheduler so that the packages mentioned above are ready to use. The second module is implemented to collect all necessary system information for future use. The system information contains the information about the host machines, domains and etc. the command line input will also be dealt with in this module. The third module is in charge of all the detecting and monitoring. The fourth module is in charge of all the operation on scheduling, which includes pausing domains and resuming domains. These four modules will be detailed discussed in following subsections. The structure of the implementation can be illustrated in the figure 5.2shown below.

**Monitoring &detecting**

**Operating**

**Information collecting**

**Initialization**

Figure 5.2 Structure of the implementation

**5.4.1 Initialization module**

At the beginning of this program, some essential module will be initialized before the execution of other two modules, which is handled by the initialization module. There are two areas needed to be initialized. The first one is the connection to the hypervisor, which is provided by the Libvirt Package. The connection to the hypervisor is the basis of all Libvirt APIs, it provides a hypervisor connection status of current virtualization system. The second one is the interface of Xen hypervisor, which is defined in the Xm package. This interface of Xen hypervisor will be used in future manipulations in Xen domain control. Initialization of Xen hypervisor interface gives the current status of Xen hypervisor. After initializing both these two parts, the APIs provided by Libvirt package and Xm package are ready to use.

The initialization module is implemented in a function *init()*, which will be called immediately after the co-scheduler starts running. In *init()* function, *virtConnectOpenReadOnly()* function is called to initialize the connection to the hypervisor in the virtual system, by this function call, Libvirt package becomes ready for further employment. The Xen hypervisor interface is initialized by calling funtion *xc\_interface\_open()* provided by Xm package, which has Xm package primed.

**5.4.2 System information collecting module**

This module is in charge of collecting all system information which are required in the third module. Most of the information this module deals with are static information, which means these information will not change during the execution of co-scheduler. For example, the PCPU number of the host machine, the size of virtual cluster and the id num of the virtual cluster members. This information collecting module will firstly will take arguments that are input from command line. The input arguments specify the two thresholds of the co-scheduler and a list of virtual machines that are contained in the virtual cluster. The list of virtual cluster members is given by user, which consists of the domain names of those virtual cluster members. When the information collecting module takes the virtual machine list, it will count how many virtual machines are specified to build up the cluster. By counting the number of virtual cluster members given by user, the size of the virtual cluster can be obtained. After that, the module will then search the domain id for each member of virtual cluster.

There are some essential information expected to be retrieved in this module. Node information includes all hardware information about the host machine including host machine's name, node's memory and the this machine's PCPU. The node information can be obtained by calling function *vitNodeGetInfo()* provided by Libvirt package. The number of active domains is given by calling the function *virtConnectNumOfDomian()* which is also included into the Libvirt package. By getting these two values, the rest of the system information can be calculated including the number of non-cluster member virtual machines on the host node and the domain ids of these non-cluster member virtual machines. For instance, the number of non-cluster member virtual machine can be obtained by subtracting the size of virtual cluster by the number of alive domains.

**5.4.3 Monitoring and detecting module**

This module is responsible to keep tracking the scheduling process based the information prepared in previous modules. In this module, the co-scheduler repeatedly retrieves the information of the virtual system. In every single iteration of this module, the information will be extracted from two loops. The first loop is expected to checks the number of available PCPUs, while the other loop is designed to check the number of virtual cluster member being scheduled. Basically, these two loops do the core job of the entire co-scheduling algorithm. From this part, the Xm APIs will be used as main tool because Xen hypervisor opens domain information to only Xm tool. In each one of these loops, the co-scheduler checks the VCPU running state to see if a domain is online, this is based on the fact that each domain is set to have only once VCPU. Retrieving VCPU running state can be realized by two following steps.

Firstly, getting the domain information by calling function *xc\_domain\_getinfo()* provided by Xm APIs. This domain information will be stored in a *xc\_dominfo\_t* structure, which contains the domain status, domain id, max id of belonged VCPU and etc. The domain status can be directly used to filter domain by its running status. There are 8 running status defined in this structure including *dying, crashed, shutdown, paused, blocked, running, hvm* and *debugged*. For example, if we want a list of domains that are shutdown, we just have to check if the shutdown value equals to 1 for every domain existing in the host machine. In this co-scheduler, we only focus running and blocked two running states. Running indicates the domain is currently scheduled online, while blocked represents the domain is either waiting to be scheduled or scheduled online but nothing actually running on this state. When there's nothing running on a domain, the PCPU occupied by the domain is considered as idle status. Two lists will be used for storing the domain information of virtual cluster members and non-cluster member virtual machines. These two lists will be further used in the next step.

Secondly, for each one of the lists obtained in previous step, the co-scheduler will go through it and check the domain status, and then it stores the relevant information into blocked and online two lists. Two counters will be setting there for recording the number of available CPU and the number of virtual cluster member being scheduled. For the list of non-cluster members, the counter 1 will be added by one if a domain status is running. For the list of virtual cluster members, the counter 2 will be added by one as well if a domain status is running.

The number of available PCPUs can be calculated by the number of active PCPUs subtracting counter 1 which is used in recording how many non-cluster member domains in a state of running. The philosophy behind this calculation is that the number of busy PCPUs equals to the number of domains that are currently in running state. Therefore, the number of available PCPUs is the number of rest PCPUs that are not in busy. These PCPUs contain the PCPUs that are not occupied by domains and the PCPUs that are occupied by non-cluster member domains but currently not in busy.

**5.4.4 Operating module**

This module is in charge of taking operation on scheduling. Before this module actually take operation, a threshold check will be conducted. There are two thresholds mentioned in previous chapter, hence, we have two counters accordingly. The counter 1 is used to compare with threshold of available PCPU, and the counter 2 is used to compare with the threshold of virtual cluster member being scheduled. In this module, if either one of the thresholds is reached, the virtual cluster will be scheduled online. This can be implemented by calling function *vmPause()* which pauses all non-cluster member virtual machines. This action makes non-cluster member virtual machine to give way to virtual cluster in terms of CPU resources. After that, this module will check if all virtual cluster members are online. If all virtual cluster members are scheduled online, then co-scheduler will call the function *vmUnpause()* to re-schedule non-cluster members.

# 6.Performance evaluation

Several experiments have been done for measuring the performances and effectiveness of this co-scheduler proposed in this project. the experiment results will be shown and analyzed in this chapter.

## 6.1Experimental environment

A Computer equipped with a quad-core CPU (Intel Core I7-3615QM CPU @ 2.30GNz) and 8GB RAM is used as the host machine. The Xen 4.2.2 is adopted as the hypervisor of this virtual system, the Ubuntu 13.04 is taken as operating system installed in this machine. The proposed scheduler is placed in the operating system installed in domain 0. GCC 4.8.4 is also installed for compiling the source file. There are 6 non-cluster domains and 4 cluster domains created for the experiments. Each of them is assigned one VCPU and 512 RAM, and CentOS 6.4 x64 is installed in each guest domain as operation system. This experiment is designed to check the running state of the virtual cluster and the usage of CPU, therefore, both virtual cluster members and non-virtual cluster members are expected to run on the host machine.

## 6.2 Experiments objectives

This co-scheduler is proposed to be solution of coordinating the usages of CPU between the virtual cluster and individual virtual machines. Therefore, the first objective is to check if all cluster members are scheduled online at same time via the co-scheduling services provided by proposed strategy. Ideally, the virtual cluster should be brought online simultaneously. At meantime, the individual virtual cluster might be paused and forced offline. The second objective is to experiment different values of the tuple of the thresholds. This part of experiment is expect to find a pair of threshold values which maximizes the fairness of CPU resource allocation, and trying to bring the best efficiency of both virtual cluster and individual virtual machine. Thirdly, the general influence on CPU usage brought by using this co-scheduler. As the side effect of co-scheduling, the CPU usage of the individual virtual machine will be harmed with no doubt since individual virtual machines are very possible to be paused when the co-scheduler tries to schedule the virtual cluster online. In this part of test, we are going to see how much degradation caused by the proposed co-scheduler.

## 6.3 Design of the Experiments

Generally, there are many benchmark suits for testing the parallel efficiency and cluster performance. Fr example, NAS-LU and NPB. These benchmark suits provide comprehensive evaluation of a computer system including virtual system, cluster system and etc.

However, this experiment doesn't adopt these benchmark suits as the evaluation and testing tools for following reason. Firstly, this the main objective is to check the running state of a virtual cluster. If all virtual machines within the virtual cluster are scheduled online is the result that we are seeking for. The evaluation of communication quality between nodes and the performance of single virtual machine are not our goals, which are also contained in these benchmark suits. Therefore, to make the evaluation of proposed co-scheduler is straightforward, we rather take a simple approach which will be introduced later. Secondly, due to the problem of driver, the virtual machines cannot be connected by a network bridge, which caused failure of building a virtual cluster. As the result, the experiment is based on simulating a cluster using virtual machines. For example, we know the cluster only works if all nodes are online, then we can assume that a virtual cluster should be working well if all its virtual nodes are online. Therefore, we track the running status of virtual cluster members to prove the effect of proposed co-scheduler.

For the first objective, 4 individual virtual machines and a virtual cluster with size of 4 are lunched to run on the host machine. As the host node is equipped with 8 PCPUs, and each of the virtual machine is assigned with specified number of VCPU, the maximum number of virtual machine we can run on the host node is 8. The number of VCPUs which a individual virtual machine has will be changed, and each of the virtual machine is set to running with a max workload. This can be considered as both virtual cluster and individual machines are under a stressed situation, then the proposed scheduler will do the scheduling job under a circumstances which is similar to the circumstance in reality. The times of virtual cluster members being scheduled online at same time will be counted, and this number will be used for comparing with the results produced without using proposed co-scheduler. By comparing these two results, the effectiveness and the performance can be seen.

For the second objective, we used configuration of the first experiment, namely, 4 individual virtual machines and a virtual cluster with size of 4. However, two thresholds specified to the co-scheduler will be changed on the purpose of comparing the effect brought by the co-scheduler. The thresholds of the two co-scheduler are expected to have major impact on the performance of proposed co-scheduling strategy. The first threshold indicating the available CPU resources that triggers the virtual cluster being scheduled online. Therefore, by changing the values of these two thresholds, we can roughly get an idea of how these two values affect the performance of this co-scheduler and the efficiency of the system.

For the third objective, we are going to use different configurations of each virtual machine to see the influences brought by the this co-scheduler. In previous experiments, the configuration of each virtual machine is fixed as we concentrate the influence brought by other factors. As more VCPU a virtual machine equipped with, the higher scheduling frequency. According to this fact, the ability of co-scheduler can be measured. Therefore, in this experiments, virtual machines with different configurations, namely, virtual machines assigned with different number of VCPUs, will be used so that the influences of CPU usage of individual virtual machines brought by co-scheduling can be obtained.

For all three experiments, a timer will be used to limit the execution within a certain time. The timer starts right before the start of co-scheduling process which is the mainly loop in this co-scheduling program. Even though, the timer starts a little bit earlier than the co-scheduling process, it can be considered to be the same time as the start time of co-scheduling process as it doesn't affect the result two much.

## 6.4 Experiments results

The first experiment consists of 10 tests. We run the co-scheduler 5 times, each time the non-cluster virtual machine is assigned a different number of VCPUs. The two thresholds are set to 4 and 0 corresponding to the threshold of available PCPUs and the threshold of virtual cluster members being scheduled. This setting of threshold aims to ensure the co-scheduler providing the very naive co-scheduling service and brings primitive influence to the virtual system, and no weight of neither side will be applied. This configuration is helpful to significantly show the improvement brought by co-scheduling in allocating the requested resources to virtual cluster. The number of VCPUs of a domain is set to 1, 2, 3, 4 and 5, which indicates how many VCPUs are in each non-cluster virtual machine in each execution turn. To compare with the performance without using proposed co-scheduler, another 5 times of execution will only check the status of virtual cluster and no so-scheduling operations will apply. Each execution will be last for 100 secs. The total number of status checking, the number of virtual cluster online and the number of virtual cluster offline will be recorded in both parts of experiment. The corresponding result of each turn is shown in figure 6.1.

The figure 6.2 graphically shows the performance in scheduling virtual cluster online. As shown in the figure below, this proposed scheduler has achieved the goal we set for this project. The virtual cluster is scheduled CPU resource effectively in general, which is the expected result of this project. In each round, our proposed co-scheduler brings significant improvement in handling virtual cluster. The online rate of using a co-scheduler decreases with the VCPU number. The reason is that with the number of VCPU increases, the scheduler process becomes complicated as there more VCPUs waiting in a job queue. As the result of long job queue, the chance of the VCPU being scheduled online becomes smaller, the chance of bringing a virtual cluster online becomes less accordingly. Therefore, the online rate decreases with the PCPU number.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| VCPU number | Total | Online | Offline | Online rate | VCPU number | Total | Online | Offline | Online rate |
| 1 | 100 | 99 | 1 | 99% | 1 | 234 | 162 | 72 | 69% |
| 2 | 113 | 104 | 9 | 92% | 2 | 215 | 113 | 102 | 53% |
| 3 | 207 | 187 | 20 | 90% | 3 | 165 | 79 | 86 | 49% |
| 4 | 169 | 148 | 21 | 88% | 4 | 115 | 47 | 68 | 41% |
| 5 | 163 | 111 | 52 | 68% | 5 | 160 | 60 | 100 | 38% |

Figure 6.1 Results of first experiment

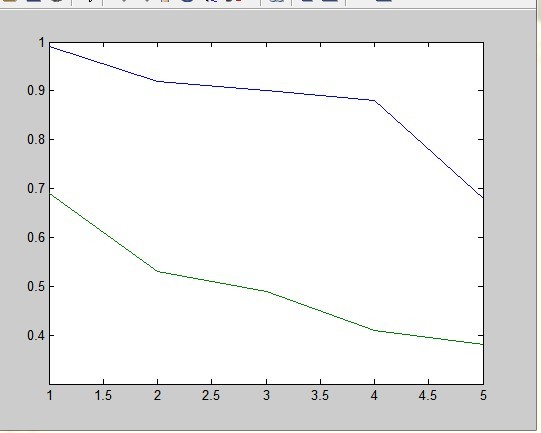


Figure 6.2 The performance of co-scheduler. The blue line indicates the performance of handling virtual cluster via proposed co-scheduler, while the green describes the performance of scheduling cluster without using proposed co-scheduler.

The second experiment contains 5 tests due to the size of virtual cluster is 4. Each single machine is assigned 2 VCPUs. We set the threshold of virtual cluster member being scheduled to 5 different values ranging from 0 to 4. When we set this value to 0, it means we give the full priority to the virtual cluster because the requirement of scheduling virtual cluster is the lowest. In other words, any virtual machine of the virtual cluster is scheduled will cause the entire virtual cluster is brought online. Oppositely, the value set to 4 gives the highest requirement of scheduling the cluster online. In fact, when the value is set to 4, the only situation that a virtual cluster can be scheduled is that the number of virtual cluster member being scheduled reaches the size of the virtual cluster. Considering the fact that the workload of each virtual machine is set to be very high in practice, thus, the effect of threshold of available CPU is considered to be limited. In this experiment, the threshold of available CPU is set to 8 to ensure that no effect comes from the threshold of available PCPU. The results are shown in the figure 6.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thresholds | Total | Online | Offline | Online rate |
| 8, 0 | 186 | 181 | 5 | 97% |
| 8, 1 | 201 | 158 | 43 | 79% |
| 8, 2 | 198 | 103 | 95 | 52% |
| 8, 3 | 189 | 74 | 115 | 39% |
| 8, 4 | 202 | 71 | 131 | 35% |

Figure 6.3 Results of second experiment

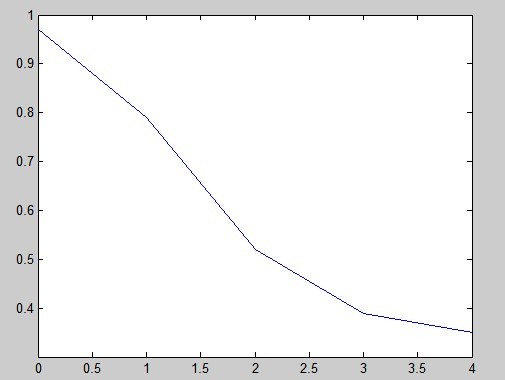


Figure 6.4 The performance affected by different threshold

The results of second experiment show the trend we expected. The threshold (8,0) gives the best result of handling the virtual cluster, the co-scheduler responds almost all request from virtual cluster, the online rate of virtual cluster is 96%. This online rate decreases with the threshold of virtual cluster members being scheduled, which is due to the reason mentioned above. When the threshold is set to (8,4), the poorest performance is given, and the virtual cluster is only scheduled 1 time. The lower bond of the performance is given by the threshold of virtual cluster being schedule ranging from the size of the virtual cluster to the above. The figure 6.4 shows the trend graphically.

The third experiment also contains 5 tests as same as above. The thresholds are adjusted to obtained the CPU usage by utilizing function *xc\_domian\_get\_cpu()*. We calculated the average VCPU usage of virtual machines. As we know the number of VCPUs o each domain, this can be done without a problem. The basic idea in this experiment is to record usages of all VCPUs when a virtual cluster online, and then it's divided by number of VCPUs. The usage is recorded using CPU cycle (trillion). The result of the experiment is shown in figure 6.5.

The results show the CPU usage we expected. When the value of threshold of cluster member being scheduled becomes high, the lower chance of scheduling the virtual cluster online since the co-scheduler counts the VCPU usage when the virtual cluster is brought online. As the result of setting threshold to a high standard, each member of virtual cluster being scheduled online won't be counted, namely, it is considered as idle status. Comparing to the situation of setting a high threshold, setting a low threshold gives a better CPU usage via more frequently scheduled the virtual cluster online, thus, the CPU is being efficiently used. Fact should be noticed, the co-scheduler does not intensively affect the usage of VCPU. the difference between the highest usage and the lowest usage is about 1.17 trillion, which is not quite big in terms of CPU cycle. The figure 6.6 illustrates the experimental results of this experiment.

|  |  |
| --- | --- |
| Threshold | Average VCPU usage (trillion) |
| 8, 0 | 25.61 |
| 8, 1 | 25.24 |
| 8, 2 | 24.70 |
| 8, 3 | 24.52 |
| 8, 4 | 24.54 |

Figure 6.5 The results of third experiment

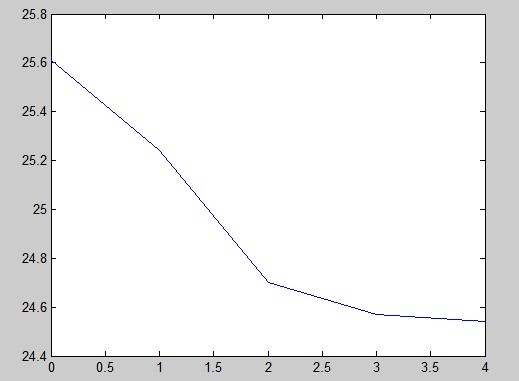


Figure 6.6 The average CPU usage of VCPUs

# 7.Conclusions and Future Work

# 7.1 Conclusions

To improve the performance of Xen in virtual cluster scheduling, a co-scheduler is proposed in this project. This scheduler aims to cooperate with exiting schedulers of Xen to efficiently allocate CPU resource to virtual machines.

There are two big challenges in this project. Firstly, to understand how Xen works is a difficulty. Xen is not like other application program, basically, it is an operating system perhaps even more complicated. To understand the how Xen works, a comprehensive study of Xen has been taken during the project by reading the source code of Xen. Secondly, how to design a co-scheduler which improves the performance of virtual cluster while keeps the efficiency of CPU usage of each VCPU is difficult. This requires a co-scheduler to balance the resource requests from both individual CPUs and virual cluster. Thirdly, this co-scheduler is expected to bring the lowest overhead to the virtual system. A scheduling process should not take too much time, otherwise, the performance will be harmed. This is included into the consideration of designing co-scheduling algorithm.

This co-scheduler is designed to be able to consider both the demands of virtual cluster and the available CPU resource of the virtual system, which gives a comprehensive thought of the present situation of the system. This co-scheduler monitors the virtual system to check if there is enough CPU resource for the virtual cluster through checking the PCPU status. On the other hand, the demands of virtual cluster is also considered by the co-scheduler via counting the times of virtual cluster members being scheduled. Two thresholds are set accordingly so that users have a flexible control of the CPU scheduling process, which means the user can adjust the values to decide if assign more resources to the virtual cluster or to the individual virtual machines.

The experiments in Chapter 6 has shown the performance of proposed co-scheduler. The results of the experiments have shown that there is a significant improvement in handling virtual cluster. There is 30% chance increased in allocating requested CPU resources by using our co-scheduler. By changing the thresholds, the weight of CPU resource allocation can be dynamically adjusted. The relation between the threshold and weight is also stated. The experiments also state the fact that the co-scheduler brings almost 0 effect to the average CPU usage, and the CPU efficiency is improved accordingly by using proposed co-scheduler, which is exactly what we expected our co-scheduler to be.

# 7.2 Further work

The current version of this co-scheduler has successfully achieved the goals that we set for the project. However, some additional works can be done for improving the performance of this co-scheduler.

First of all, this co-scheduler is currently implemented outside the kernel of Xen. This feature benefits from good portability, however, the overall overheads significantly high. This is because the current version of co-scheduler is not directly communicating with the hypervisor. If we want to achieve a better performance of this co-scheduler, the best way is to implement it in the Xen kernel to cut down the overhead by directly manipulating the CPU resource allocating.

Secondly, this co-scheduler uses a strategy that pauses all the individual domains to get enough CPU resources. This also brings waste of CPU resource if requested number of CPU only requires one individual virtual machine offline. Interrupting all individual virtual machines is not necessary. The co-scheduler can be optimized by interrupting the individual domains only if it is necessary. By this optimization, the efficiency of individual virtual machines should be improved.

# Reference

[1] P. H. Gum,"System/370 extended architecture: Facilities for virtual machines", *IBM Journal of Researchand Development*, 27(6):530–544, 1983.

[2] J. E. Smith and R. Nair,"Virtual Machines: Versatile platforms for systems and processes",Elsevier, USA,2005*.*

[3] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn,"Parallel job scheduling - a status report", *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, *pages 1–16, 2004.*

[4] C. Weng, Z. Wang, M. Li, and X. Lu,"The hybrid scheduling framework for virtual machine systems", *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual ExecutionEnvironments (VEE)*, pages 111–120, 2009.

[5] C. A. Waldspurger,"Memory resource management in VMware ESX server",*Proceedings of the 5thsymposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[6] VMware. http://www.vmware.com/.

[7] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A.Warfield, P. Barham, and R. Neugebauer,"Xen and the art of virtualization",*Proceedings of the ACM Symposium on Operating Systems Principles(SOSP)*, pages 164–177, October 2003.

[8] Xen. http://www.xen.org/.

[9] KVM. http://www.linux-kvm.org/page/main page.

[10] Hyper-V. http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx.

[11] Virtualbox. http://www.virtualbox.org/.

[12] *Network-Based Information Systems: First International Conference,* NBIS 2007 ISBN 3-540-74572-6 page 375.

[13]Bader, David; Robert Pennington, "Cluster Computing: Applications", *Georgia Tech College of Computing,* Retrieved 2007-07-13.

[14]"Nuclear weapons supercomputer reclaims world speed record for US",*The Telegraph.* 18 Jun 2012.

[15]Sukanya Suranauwarat, “A CPU Scheduling Algorithm Simulator”*, Milwaukee,WI 37th ASEE/IEEE Frontiers in Education Conference*, October 10-13, 2007*.*

[16]M., Sindhu; R., Rajkamal; P., Vigneshwaran, "An Optimum Multilevel CPU Scheduling Algorithm",*Advances in Computer Engineering (ACE), 2010 International Conference on, vol., no., pp.90,94, 20-21* June 2010 doi: 10.1109/ACE.2010.14.

[17]Sabrian, F., C.D. Nguyen, S. Jha, D. Platt and F. Safaei,"Processing resource scheduling in programmable networks",*Computer communication,* 28:676-687.

[18] Abraham Silberschatz, Peter Baer Galvin, Greg Gangne,”OperatingSystem Concepts”, *edition-6, 2002, John Wiley and Sons, INC.*

[19]Mr. Umar Saleem and Dr. Muhammad Younus Javed, “Simulation ofCPU Scheduling Algorithm”, *0-7803-6355-8/00/ $10.00* @ 2000 IEEE.

[20] Sun Huajin’, Gao Deyuan, Zhang Shengbing, Wang Danghui. ”DesignFast Round Robin Scheduler in FPGA”, *0-7803-7547-5/021/$17.00 @2002 IEEE.*

[21] Md. Mamunur Rashid and Md. Nasim Adhtar, “A New Multilevel CPUScheduling Algorithm”, *Journals of Applied Sciences 6 (9): 2036-2039,*2009.

[22] Neetu Goel; R.B. Garg*.*"A Comparative Study of CPU Scheduling Algorithms",*International Journal of Graphics & Image Processing, Vol 2, issue 4,* November 2012.

[23] Sriram Govindan; Jeonghwan Choi; Arjun R. Nath; Amitayu Das; Bhuvan Urgaonkar;Anand Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Management in Consolidated Xen-Based Hosting Platforms",*IEEE TRANSACTIONS ON COMPUTERS, VOL. 58, NO.* 8, AUGUST 2009

[24] Chia-Ying Tseng1 and Yi-Ling Chung, “An Enhanced CPU Scheduler for XEN Hypervisor to Improve Performance in Virtualized Environment”.

[25] Kenneth J. Duda and David C. Cheriton, "Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler",*in Proc. of the 17th ACM Symposium on Operating Systems Principles, Kiawah Island, SC,* December 1999.

[26] J. C. R. Bennett and H. Zhang,"WF2 Q: worst-case fair weighted fair queueing",*Proceedings of IEEE INFOCOM’96,* pages 120–128, San Francisco, CA, Mar. 1996.

[27] A. Demers, S. Keshav, and S. Shenker,"Analysis and simulation of a fair queuing algorithm",*Proceedings of the SIGCOMM ’89 Symposium,* pages 1–12, Sept. 1989.

[28] A. K. Parekh and R. G. Gallager,"A generalized processor sharing approach to ﬂow control in integrated services networks: the single-node case",*ACM Transactions on Networking, 1(3):344–357, June 1993.*

[29] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton,"A proportional share resource allocation algorithm for real-time, time-shared systems",*Proceedings of the 17th IEEE Real-Time Systems Symposium,* pages 288–299, Dec. 1996.

[30] Kenneth J. Duda and David R. Cheriton,"Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose".

[31] A. Demers, S. Keshav, and S. Shenker,"Analysis and simulation of a fair queueing algorithm",*Proceedings ACM SIGCOMM,* pages 1–12, September 1989.

[32] P. Goyal, X. Guo, and H. Vin,"A hierarchical CPUscheduler for multimedia operating systems",*Proceedings of the Usenix Symposium on Operating SystemDesign and Implementation ’96,* pages 107–122, October 1996.

[33] Andy Bavier; Larry Peterson, “The Power of Virtual Time for Multimedia Scheduling”.

[34] K. J. Duda and D. R. Cheriton. Borrowedvirtual-time (BVT) scheduling. http://wwwdsg.stanford.edu/pub/bvt.html, 2000.

[35] Ludmila Cherkasova; Diwaker Gupta and Amin Vahdat, “Comparison of the Three CPU Schedulers in Xen”.

[36] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications",*IEEE Journal of Selected Areas in Communications*, 1996.

[37] Jinho Hwang; Wood, T., "Adaptive dynamic priority scheduling for virtual desktop infrastructures", *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on, vol.,no., pp.1,9, 4-5 June 2012 doi: 10.1109/IWQoS.2012.6245988.*

[38] Diego Ongaro; Alan L. Cox; Scott Rixner, “Scheduling I/O in Virtual Machine Monitors”.

[39] Chia-Ying Tseng1 and Po-Chun Huang, “An Efficient Virtual CPU Scheduling Algorithm for Xen Hypervisor in Virtualized Environment”.

[40] Zheng, Zhiyun, Ying Zhang, Zhenfei Wang, Xingjin Zhang, Liping Lu,"The multi-processor load balance scheduler based on XEN",2*012 International Conference on Systems and Informatics (ICSAI)*, May. 2012: 905-909.

[41] N. Nishiguchi, “Evaluation and consideration of the credit scheduler for client virtualization”,*Xen Summit Asia*, 2008.

[42] Rajendra Bele; Chitra Desai, "Exploring CPU Scheduling Strategies in Xen Virtualization Technology"*, International Journal of Sciences Research Article (ISSN 2305-3925) Volume 2*, Issue Mar 2013

[43] Zhibo Chang; Jian Li; Ruhui Ma; Zhiqiang Huang; Haibing Guan, "Adjustable Credit Scheduling for High Performance Network Virtualization", *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pp.337,345, 24-28 Sept. 2012

[44] Vidya Rao Suryanarayana, “Credit Scheduling and Prefetching Hypewrvisor Using Hiddeb Narkov Models”.

[45 [Ludmila Cherkasova; Rob Gardner,”Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor”.

[46] Diego Ongaro Alan L. Cox Scott Rixner, “Scheduling I/O in Virtual Machine Monitors”.

[47] Yulong Yu, Yuxin Wangy, He Guo, Xubin He. “Hybrid Co-Scheduling Optimizations for Concurrent Applications in Virtualized Environments”.

[48] Fabrizio Petrini,Wu-chun Feng,"Improved Resource Utilization with Buffered Coscheduling", *Journal of Parallel Algorithms and Applications* , 2000

[49] J.K. Ousterhout, “Scheduling techniques for concurrent systems,” *in Proc. of the 3rdInternational Conference on Distributed Computing Systems*, 1982, pp. 22–30.]

[50] Fabrizio Petriniyand Wu-chun Feng, "Scheduling with Global Information in Distributed Systems".

# Appendix - User Manual

1. Install Ubuntu 13.04

2. Install Xen 4.2.2.

3. Install libvirt package.

4. Extract the co\_sched.zip

5. Compile the source code by following command:

*gcc -o co\_sched co\_sched.c -lvirt -lxenctrl*

6. Launch all the virtual machines to run.

7. Run the co-scheduler by typing command similar to below one:

*./co\_sched -threshold num num -clustervm vm1 vm2 vm3*