International Journal of Multimedia and Ubiquitous Engineering
Vol. 10, No. 3 (2015), pp. 283-298
http://dx.doi.org/10.14257/ijmue.2015.10.3.26

Formal Description of the Virtual Machines System Framework

Xiaodong Liu, Li He and Hating Xu
School of Computer Science and Engineering, Henan Institute of Engineering, Zhengzhou, 451191, China
liuxiaodongxht@qq.com, engineerheli@126.com, 2279883989@qq.com

Abstract

Virtual resources have the duality of hardware and software, which brings new challenges to virtual resources management and manipulation. There is urgent need for formally describing virtual resources in order to effectively reason about virtual resources allocation, scheduling and management. This paper describes the virtual resources and their dynamic behavior. We first give the formal definition of the resources pool and the virtual machine (VM). Then, we describe the dynamic behavior of VMs. In particular, we present a formal description of how virtual resources are used by VMs. This paper reveals the logical laws of virtual resources and their dynamic behaviors and makes the virtualization theory system more rigorous.

Keywords: *virtualization, resources management, formal description, formal semantics*

1. Introduction

Virtualization technology is gaining popularity as a software-based solution for building shared hardware infrastructures. In this emerging technology, each physical server runs a software layer called Virtual Machine Monitor (VMM) that supports the execution of multiple VMs and provides safety and isolation to the overlying VMs. VMM also has the ability to migrate running VMs between physical servers to improve the availability and security of VMs. With the advantage of functional isolation, manageability and live migration, virtualization technology has been widely used. Over the years, a number of VMMs have been proposed. Xen [1] is an x86 VMM and supports both full-virtualization and para-virtualization. The virtualization approach taken by Xen is extremely efficient. It allows operating systems, such as Linux and windows XP, to be hosted simultaneously for a negligible performance overhead. Other VMMs, like KVM [2], are designed to only support full-virtualization.

The virtualization technology converts the physical resources, such as CPU, memory, disk and I/O to “resources pool”. The VMs can be customized according to the requirement on the resources pool. Multiple isolated VMs can be created on the resource pool according to the requirement. The resources of VMs are the software simulation of the hardware resources. They can be created or destroyed dynamically, and they can even migrate from one physical server to another. Therefore, virtual resources have software properties. On the other hand, virtual resources such as virtual CPU, virtual I/O and virtual Memory can perform compute, communication and storage operation. These resources also have hardware properties. Virtual resources have the duality of software and hardware, which brings new challenges to virtual resources allocation, scheduling and management.

The recent research on virtual resources scheduling and management mainly focuses on improving VM performance[3,4], maximizing underlying physical resources utilization [5, 6] or achieving underlying physical resource fairness [7-9]. However, these approaches are not based on a formal description and a formal semantics, or provide only a partial formal semantics. We believe that formal semantics is essential for reasoning about the virtual resources scheduling and management. In order to effectively reason about virtual resources management and manipulation, the virtual resources need to be described in a formal method. Formal semantics is the cornerstone for state space exploration techniques [10] which can be used to build trustworthy, highly reliable systems. More generally, we advocate that formal semantics is essential to describe virtual resources, as it offers the potential to reason about the correctness of virtual resources management and manipulation. However, as far as we know, it has no work on formal methods of virtual resources.

This paper describes the virtual resources and their dynamic behavior. We take xen VMM as the example, because xen VMM supports both para-virtualization and full-virtualization. We first formally define the resources pool, the virtual resources pool and the virtual machine, and we also describe the relationship among them. Secondly, we give the operation semantic of VMs and describe their dynamic behavior, including VM creation, VM destruction, VM clone and VM migration. Thirdly, we describe the virtual resources of VMs and formally describe how these virtual resources are used.

Our major contributions are the following aspects:

- We propose that virtual resources have the duality of hardware and software.
- We give the operation semantics of VMs.
- We describe virtual resources and their dynamic behavior in a formal method.

The rest of the paper is organized as follows: Section 2 introduces the virtual resources management framework. Section 3 introduces the formal definition of the VM. Section 4 introduces the operation semantic. Section 5 introduces the virtual resources of VMs. Section 6 introduces the related work. Finally, Section 7 gives conclusions and future work.

2. Background and Related Work

This section presents an overview of the virtual resources management framework, focusing on the concept of CPU virtualization, I/O device virtualization and memory management.

2.1. Xen Architecture

Xen [1] is an open-source VMM that allows multiple operating systems to share the same physical server in a safe and resource managed fashion. Figure 1 describes the xen architecture. The Xen VMM is located between operating systems and hardware. It provides virtual resources, such as virtual CPU (VCPU) and virtual I/O device, for the overlying operating systems and performs functions such as scheduling CPU and allocating memory among operating systems. There is a privileged domain (Dom0) in the xen VMM which is an auxiliary management domain. It can access to hardware resources directly. Guest operating systems (OS) are not allowed to access to hardware resources directly and they must access hardware resources through Dom0.

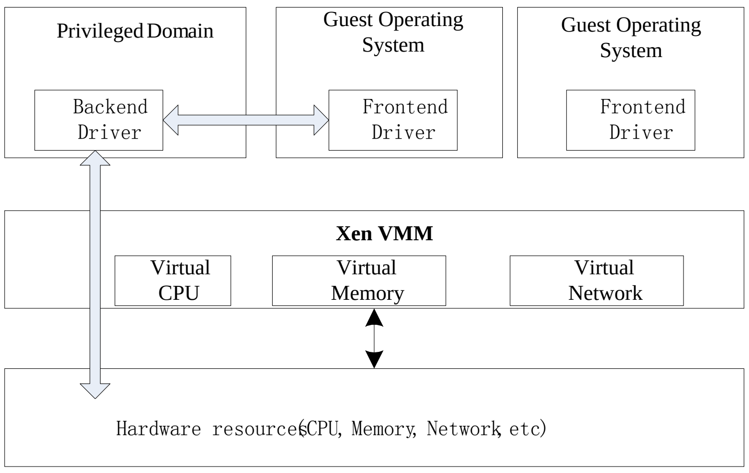


Figure 1. Xen Architecture

2.2. CPU Virtualization

In the xen virtualization environment, Guest OS cannot directly schedule physical CPU. Xen establishes the virtual CPU (VCPU) structure and provides one or more VCPUs for every Guest OS. All VCPUs are time-division multiplexing physical CPU. So, xen need reasonable allocation of time slices for VCPUs and schedule them.

Xen provides multiple schedulers, such as borrowed virtual time (BVT) [11], simple earliest deadline first (SEDF) [12] and credit scheduler [13]. The credit scheduler is xen's default scheduler at present. Its overall objective is to allocate the processor resources fairly. The Guest OS is assigned a weight value when it is created. Each Guest OS is allocated a certain number of credits according to its weight every 30 milliseconds. The credits will be allocated to VCPUs of the Guest OS fairly. As a VCPU runs, it consumes credits. According to the credits of VCPU, a VCPU's priority can be one of the three values: OVER, UNDER and BOOST. If VCPUs are in the OVER state, then they have used up its fair share of CPU resources. If VCPUs are in the UNDER state, then they have CPU resources that can be consumed. The BOOST state provides a mechanism for domains to achieve low I/O response latency. All VCPUs in BOOST state are placed in front of those in UNDER state in the run queue, while those in OVER state are kept in the tail of the queue. Every physical CPU has a run queue of VCPUs. The queue is sorted by the priority of VCPUs and the head of the queue is always selected to run.

2.3. I/O Device Virtualization

Xen adopts split virtual driver model by introducing a privileged driver domain (Dom0). A virtual driver consists of two split sub-drivers, respectively called front-end driver (FE) and back-end driver (BE). The FE driver provides interfaces of the virtual driver to Guest OS, but the FE driver cannot conduct real I/O operations. The Dom0 conducts real I/O operations on behalf of Guest OS. Communications between FE and BE are implemented by the I/O ring mechanism. The I/O ring adopts producer/consumer communication mode. When Guest OS need perform I/O operation, they put read/write requests into I/O ring. The BE will read these requests, process them and put response information into I/O ring. Processed response information can be read by the FE. FE and BE notify each other of an I/O event via an event channel. This I/O model improves the efficiency of the I/O device and enhances the reliability of an entire system.

2.4. Virtual Memory Management

The initial memory allocation for each Guest OS is specified at the time of its creation. But, the allocated memory is not fully utilized by the Guest OS when it is running. To make full use of physical memory, xen implemented balloon driver technology [14] to adjust physical memory of Guest OSes. Xen VMM can reclaim memory by balloon driver inflation and increase the memory of Guest OS by balloon driver deflation.

3. The Formal Definition of the VM

This section gives the formal definition of the VM and resources pool.

Firstly, we introduce some concepts and notations.

- We use ∏for the resources pool. It is the logical abstract of physical resources. The VM can be customized according to the requirement on the resources pool.
- we can form a view VM:∏ to mean that a virtual machine VM is running on the resources pool ∏. The view VM:∏ can also be expressed as ∏(vid), where vid is the ID of the virtual machine which is running on the resource pool ∏.
- Virtual resources pool (VRP). Virtualization technology also allows multiple virtual machines to be created on the virtual machine. So, a virtual machine can be considered as a "virtual resources pool". We use ∏(vid) for the virtual resources pool.
- Res::=CPU|memory|disk|I/O|VCPU|Vmemory|Vdisk|VI/O, where CPU, memory, disk and I/O is the resources of ∏, VCPU is the virtual CPU of the ∏(vid) , Vmemory is the virtual memory of the ∏(vid) , Vdisk is the virtual disk of the ∏(vid) and VI/O is the virtual I/O of the ∏(vid).
- We suppose there are n VMs are running on the same resources pool ∏, we use VM_i for the i-th VM and $VM_i.VC_i$ for the i-th VCPU of the VM_i.
- We suppose $VM_{i1 \ldots i(n-1)}$ is a VM and there are n VMs are running on the $VM_{i1 \ldots i(n-1)}$, we use $VM_{i1 \ldots i(n-1)in}$ for the in-th VM which is running on the $VM_{i1 \ldots i(n-1)}$.

Table 1 gives the list of symbols we use in this paper along with their meaning.

Table 1. Description of Symbols Used in this Paper

∏	resources pool
VC_i	The i-th VCPU of the VM
C_i	The i-th CPU of the resources pool
R	one resource of Res
m	a memory space
v	the value of the data
w_i	The weight of the VM_i
v_i	The number of VCPUs of the VM_i

Definition 3.1. A resources pool is a four-tuple ∏=<C, M, D, B> where:

- C is a finite set of CPU resources. Every element of C can be expressed to {(c, h)| c∈N, h∈R+} where c is the number of CPU and h is the frequency of the CPU.
- M is a finite set of memories.
- D is a finite set of disks.
- B is the size of I/O bandwidth.

Definition 3.2. ∏ is a resources pool and VM1 is a virtual machine that is running on the ∏.

(1) ∏(1) is a virtual resource pool.

(2) If $\prod(v_1, v_2, \ldots v_n)$ is a virtual resources pool, then $\prod(v_1, v_2, \ldots v_n v_{n+1})$ is a virtual resources pool

Only generated by (1) and (2) is a virtual resources pool.

Definition 3.3. A virtual machine is a five-tuple VM=< C_v, M_v, D_v, B_v, S> where:

- C_v is a finite set of VCPU resources. Every element of Cv can be expressed to {(c, h)| c∈N, h∈R+}. Where c is the number of VCPU and h is the frequency of the VCPU.
- M_v is a finite set of virtual memories.
- D_v is a finite set of virtual disks.
- B_v is the size of the virtual bandwidth

S is the state of the virtual machine. S::=r|b|p|s|d where: r represents that the virtual machine is running, b represents that the virtual machine is blocked because of waiting for I/O, p represents that the virtual machine is paused, s represents that the virtual machine is shutdown and d represents that the virtual machine is dying. At any time, the state of the virtual machine can only be one of the S.

4. Operation Semantics

When we want to create a VM, we need to specify the parameters, such as the number of core, the size of memory, the size of disk and the size of bandwidth. When the VM is created, it will be allocated to resources according to the parameters by the VMM.

The CPU resources allocation is maps VCPUs of the VM to CPUs of the resources pool. The memory and disk resources allocation is allocates the specified size of address space.

Definition 4.1. For each $VC_i \in VM.C_v$, the function m (VC_i) $=C_i$ represents that the VC_i is allocated to the run queue of C_i.

Definition 4.2. Let Lm be a set of memory address space, we define m[VM.vm$\mapsto$ Lm] for the virtual memory of VM is allocated to the address space Lm. Let L_d be a set of disk address space, we define m[VM.vd$\mapsto$ L_d] for the virtual disk of VM is allocated to the address space L_d.

When we want to create a VM, we can also specify the bandwidth value. In the xen virtualization, the bandwidth value is converted into a certain number of credits.

Definition 4.3. let T:B→C_b be a transfer function. For a input B, we define the transfer function T:B→C_b by T(B)=C_b.

We use the notation VM_Create(∏(vid$_0$)) to mean that creates a virtual machine on the resource pool ∏. It should be noted that not all creation operation can succeed. When the capacity of the resources pool ∏ can not satisfy the resources requirement of the VM, the VM cannot be successfully created. The formal definition of the creating conditions of VMs as follows.

Definition 4.4. Let S be a finite set, the operation size(S) return the size of the set. Let S1 and S2 are two finite set, we define the operation

S1∪S2={s|s∈S1 or s∈S2},

S1∩S2={s|s∈ S1 and s∈ S2},

S1-S2={s|s∈ S1 but s∉ S2}.

Definition 4.5. The VM=< C_v, M_v, D_v, B_v > can be successfully created on the resources pool ∏=<C,M,D,B> only when the following relationships are satisfied:

1. size(C_v)<=size(C)
2. Size(M_v)<=size(M)
3. size(D_v)<size(D)
4. B_v <=B.

When the specified parameters of the VM satisfy the definition 4.5, the VM can be created. The creation rule can be described as follows:

$$\Pi(vid_0) = VM\_Create(\Pi(vid\ 0))$$
$$= \frac{[m(VC_i), \forall 1 \le i \le m], m[M_v \mapsto L_m], m[D_v \mapsto L_d], [VM.Cb = T(B)]}{[M = M - M_v], [D = D - D_v], [B = B - B_v], \Pi[vid / vid_0]} \quad \text{(Rule1)}$$

We also give the destruction rule which is described as follows:

$$VM\_Destroy(\Pi(vid))$$
$$= \frac{[m[VC_i \mapsto <>], \forall 1 \le i \le m], m[VM.M_v \mapsto <>], m[VM.D_v \mapsto <>], [VM.B_v \mapsto <>]}{[M = M \cup M_v], [D = D \cup D_v], [B = B \cup B_v], \Pi[vid \mapsto <>]} \quad \text{(R}$$

ule2)

Virtual machines are not only can be dynamically created and destroyed, but also can be cloned and dynamically migrate from one resources pool to another.

Definition 4.6. let VM=< C_v, M_v, D_v, B_v , S> is a virtual machine and it is running on the resources pool ∏. We define the clone of the VM for creating a virtual machine VMclone=< C_v, M_v, D_v, B_v , S> on the same resource pool ∏.

Definition 4.7. let VM=< C_v, M_v, D_v, B_v , S> is a virtual machine and it is running on the resources pool $∏_1$. we define the migration of VM from $∏_1$ to $∏_2$ for creating a VM=< C_v, M_v, D_v, B_v , S> on the resources pool $∏_2$ and destroying the VM from resources pool $∏_1$.

Proposition 4.1 the VM can be cloned on the resources pool ∏ only when size(Vmemoy)<=size(M)&size(D_v)<=size(D)&size(B_v)<=size(B).

Proof. If the VM is running on the resources pool ∏, then by definition 4.5, len(VM.c)<=len(∏.c). By definition 4.6, len(VMclone.C)=len(VM.clone). So, len(VMclone.C)<= len(∏.c). By definition 4.5, the clone operation can succeed only when ize(Vmemoy)<=size(M)&size(D_v)<=size(D)&size(B_v)<=size(B).

Definition 4.8. VM=< C_v, M_v, D_v, B_v ,S> can be successfully migrated to the resources pool ∏=<C,M,D,B> only when the following relationships are satisfied:

1. size(C_v)<=size(C)
2. Size(M_v)<=size(M)
3. size(D_v)<size(D)
4. B_v<=B.

Now, we give the clone rule and migration rule as follows.

$$\Pi(vid_n) = VM\_Clone(\Pi(vid))$$
$$\frac{VM\_Create(\Pi(vid + 1))}{[M = M \cup M_v], [D = D \cup D_v], [B = B \cup B_v], \Pi[(vid + 1)/vid_0]} \quad \text{(Rule 3)}$$

$$VM\_Migrate(\Pi_1(vid) \to \Pi_2(vid_0)) = \frac{VM\_Create(\Pi_2(vid_0))}{VM\_Destroy(\Pi_1(vid))} \quad \text{(Rule 4)}$$

5. Virtual Resources of VMs

This section describes the virtual resources of VMs, including virtual CPU, virtual memory and virtual I/O.

5.1. Syntax Definitions

First, we introduce some syntax that will be used later.

(1) $R : (\vee_{i=1}^{n} P_i)$ where R∈ Res and it is shared among $P_1, P_2, ..., P_n$. Any one or multiple of them can access to the R at the same time.

(2) $R : (\|_{i=1}^{n} P_i)$. R is shared among $P_1, P_2, ..., P_n$ and only one of them can access to the R at the same time.

(3) $VC\_Migrate(VC_i, C_i \rightarrow C_i)$. The VCPU VC_i migrates from C_i to C_i.

(4) $VM \xrightarrow{r} R$. The VM can access to the R and the R can only be read by the VM.

(5) $VM \xrightarrow{w} R$. The VM can access the R and the R can only be written by the VM.

(6) $VM \xrightarrow{rw} R$. The VM can access the R and the R can be read and written by the VM.

(7) $VM_i.R \xrightarrow{r} VM_i$. The VM_i permits the VM_i access the R can be read by theVM_i.

(8) $VM_i.R \xrightarrow{rw} VM_i$. The VM_i permits the VM_i access the R and the R can be read and written by the VM_i.

(9) $VM_i.R \xrightarrow{t} VM_i$. The VM_i loses the ownership of the R and the VM_i gets the ownership of the R.

(10) $VM.v \xrightarrow{in} m$ The VM inputs the data v into m.

(11) $VM.v \xleftarrow{out} m$ The VM gets the data v from m.

(12) $VM_i.v \xrightarrow{send} VM_j$. The VM_i sends the data v to the VM_j through network.

5.2. CPU Virtualization

In the xen virtualization system, VCPUs scheduling of the VMM must decide which VCPUs should run on the physical CPU. Given a resources pool ∏ with N physical CPUs, we will distinguish them with an identifier C_i, i=0,1,…,N-1. Every C_i has a run queueCq_i.

Definition 5.1. A status of VCPU can be one of the following:
BOOST: if the VCPU wakes up from a blocked event, its status will be BOOST.
UNDER: if the credits of the VCPU are more than zero.
OVER: if the credits of VCPU are less than or equal to zero.

Definition 5.2. a Cq is a VCPU run queue. The order of the Cq is according to the status of the VCPU. The head of Cq will select to run on theC_i. We form a view VC∈ Cq_i to mean that the VC is in the queue Cq_i and form a view VC: C_i to mean that the VC is running on theC_i.

Definition 5.3. a queue Cq_i is busy if it has one or more VCPUs waiting for scheduling, otherwise it is idle.

Definition 5.4. we suppose there are N vcpu$VC_1, ..., VC_N$, if VCi∈ Cq_i,i=1,…N, then $C_i : (\|_{i=1}^{n} VC_i)$.

Definition 5.5. if VC:Ci and $C_i \xrightarrow{rw} Cache$, then $VC \xrightarrow{rw} Cache$

Definition 5.6. if VC1:Ci ,VC2:Cj, $C_i \xrightarrow{rw} Cache$,$C_i \xrightarrow{rw} Cache$ then $(VC_1 \vee VC_2) \xrightarrow{rw} Cache$

5.3. Memory Resources of VMs

This section describes the memory resources of VMs. When VMs are created, they are allocated a certain size of memory according to their specified size. However, the allocated memory is not fully utilized by VMs when they are running. Firstly, applications which are running on the VMs may lead to insufficient memory, and the VMs need more memory. Secondly, the memory of VMs is idle when their workloads are low, and the VMM will recycle the idle memory of VMs. Thirdly, the memory of one VM may be accessed to by another VM or the ownership of the memory may transfer to another VM in the xen shared memory mechanism. In this paper, we mainly focus on page mapping and page transferring.

Definition 5.7. We suppose VM_i and VM_j are any two VMs which are running on the same resources pool. Page mapping is the memory page of the VM_i is allowed to be accessed to by the VM_j but the ownership is also belongs toVM_i. Page transferring is the VM_j get the ownership of the memory page and the VM_i loses the ownership of the memory page.

The page mapping and page transferring are based on the grant mechanism.

Definition 5.8. A grant entry is a three-tuple GE=<T, ID, Fm> where:

- T is the type of xen grant mechanism. T:: =NULL|M|T where NULL represents that there is no grant, M represents that the type of grant entry is page mapping and T represents that the type of grant entry is page transferring. Each M or T has its subflags, which are the memory page frame status in the page mapping or page transferring process. subflag:: =readonly|reading|writing|transferring|transferred where readonly represents that the shared page can only be read by the destination VM, reading represents that the shared page is reading by the destination VM, writing represents that the shared page is writing by the destination VM, transferring represents that the shared page is page transferring and transferred represents that page transferring has been completed.
- ID is the id of the destination VM.
- Fm is the page frame number of the granted memory.

Each GE is marked by an integer key, which is called grant reference (GR).

5.3.1. Page Mapping

This section describes the interactions that should be performed when a VM_j want to access to the memory page P of VM_i.

The page mapping behaves as follows.

1. $VM_i \xrightarrow{create} GR$,$VM_i.GR \xrightarrow{send} VM_i$

GR=<M-readonly,VM$_j$,P>.

As before, VM$_i$ creates a GR and send it toVM_i.

2. $VM_i \xleftarrow{m} GR.P$

When the VM_i receives the GR, it can map the memory page P, which is granted access, to its memory address space.

3. $VM_i \xrightarrow{access} P$

Once the above two operations are completed, the VM_i can access to the memory page P.

4. $VM_i \xrightarrow{destroy} P$

When the VM_i completed the memory access, it will remove the memory page from its memory address space.

5. $VM_i \xleftarrow{recycle} VM_i.GR, VM_i \xrightarrow{destroy} GR$

Then, the VM_i recycles the GR from the VM_j and destroys the GR.

5.3.2. Page Transferring

This section describes the interactions that should be performed when the ownership of memory page P is transferring from VM_i to VM_j.

The page transferring behaves as follows.

1. $VM_i \xrightarrow{create} GR, VM_i.GR \xrightarrow{send} VM_i$

GR=<T, VM_i ,P>.

The VM_i creates a GR and sends it to the VM_i.

2. $VM_i \xrightarrow{create} T_{trs}$

Ttrs=<mfn,id,gr> where mfn is the grant memory frame number which will be transferred, and id is the VM id of the receiver and gr is the GR provided by the receiver.

When the VM_i receives the GR transferred by VM_i, it creates a Ttrs and completes the page transferring operation.

3. $VM_i \xrightarrow{receive} P$

The VM_i receives the transferred memory page P and maps the memory page P, which is granted to be accessed, to its memory address space.

4. $VM_i \xrightarrow{destroy} GR$

When the VM_i has mapped the memory page P, which is granted to be accessed, to its memory address space, it destroys the GR.

5.3 I/O Resources of VMs

This section describes the I/O virtualization of VMs.

5.4.1. Shared-memory Communication Model

This section describes the behavior when one VM wants to communicate with another VM which is running on the same resources pool. An example scenario would occur when the VM_i tries to send the message v to the VM_j. VMs which are running on the same resources pool can communicate through memory. The shared memory communication between VMs is different from processes. The sharing mechanism is based on the authorization mechanism. The communication behaves as follows.

1. $VM_i v \xrightarrow{in} m$

As before, the VM_i puts the v into the memory space m.

2. $VM_i \xrightarrow{create} GR, VM_i.GR \xrightarrow{send} VM_i$

GR=<M-readonly, VM_i, m>.

The VM_i creates a GR, the grant type is the page mapping, the subflag is readonly, the destination id is the id of VM_i and Fm is the page frame number of m. Then, the VM_i sends the GR to VMj.

3. $VM_i \xrightarrow{receive} GR, VM_i \xleftarrow{m} m$

Once the VM_i receives the GR, it can map the memory m to its memory address space.

4. $VM_i.v \xleftarrow{out} VM_i.m$

The VM_i can read the v from the m of the VM_i.

5. $VM_i \xrightarrow{destroy} m$

When the VM_i has get the data v, it removes the memory page from its memory address space.

6. $VM_i \xleftarrow{recycle} VM_j.GR, VM_i \xrightarrow{destroy} GR$

The VM_i recycle the GR from the VM_j and destroy the GR.

5.4.2 Virtual Device Communication Model

This section describes the behavior when one VM wants to communicate with another VM which is running on the different resources pool. When two VMs are running on the different resources pool, they communicate through the underlying physical I/O device. However, VMs cannot access to physical I/O device directly. Virtual device model is a commonly used model of Para-virtualization. The I/O performance of the model can close to the real physical platform. When the hardware virtual machine (HVM) installed para-virtualization driver [15], it can also use virtual device model. The virtual device model is based on device I/O ring, event channel and grant mechanism.

Definition 5.9. A device I/O ring is a four-tuple $Sring$ =<QP, QC, PP, PC,> where:
QP is the number of request producer;
PP is the number of response producer;
PE is the number of request event;
QE is the number of response event.

Definition 5.10. A front-end ring is a three-turple $Fring$ =< QP, PC, NR > where:
QP is the number of request producer;
PC is the number of response consumer.
NR is the size of the Fring.

Definition 5.11 A back-end ring is a three-turple $Bring$ =< QC, PP, NR > where:
QC is the number of request consumers;
PP is the number of response producer;
NR is the size of the Bring.

Definition 5.12. We define the $Fring$ input I/O request into $Sring$ as $Fring.QP = Sring.QP$. We define the Bring input I/O response into $Sring$ as $Bring.PP = Sring.PP$

Definition 5.13. We define ioreq is the I/O request information which includes the address and length of data which need write to I/O device. We define iorep is the I/O response information which include the id and completed status of the ioreq.

Definition 5.14. An I/O request queue (ReqQ) or I/O response queue (RepR) is a linear storage device which only allows insert I/O request on one end and delete I/O request on the other end.

Proposition5.1. the size of the idle queue of the ReqQ is $|NR - QP + PC|$. A ReqQ is full if and only if $|NR - QP + PC| = 0$

Proof. The size of the ReqQ is NR according to the definition 5.10. The number of requests generated by front-end is QP according to the definition 5.10. When there are no responses from the back-end ring, the size of the idle queue of the ReqQ is $|NR - QP|$. When the number of responses from the back-end is PC, the size of the idle queue of the ReqQ will be $|NR - QP + PC|$.

Because the size of the idle queue of the ReqQ is $|NR - QP + PC|$, if $|NR - QP + PC| = 0$, the ReqQ is full. If $|NR - QP + PC| \neq 0$, the ReqQ is not full. So, A ReqQ is full if and only if $|NR - QP + PC| = 0$.

Proposition5.2. There are outstanding messages to be processed on a ring if and only if $| Sring.PP - Fring.PC | > 0$

Proof. The number of requests inputted into the shared ring by the front-end is $Sring.PP$ according to the definition 5.9. The number of processed requests is $Fring.PC$ according to the definition 5.10. There are outstanding messages to be processed on a ring if $| Sring.PP - Fring.PC | > 0$. There are not outstanding messages to be processed on a ring if $| Sring.PP - Fring.PC | = 0$. So, there are outstanding messages to be processed on a ring if and only if $| Sring.PP - Fring.PC | > 0$.

Example 5.1. An example scenario would occur when the VM_i tries to send the message v to the VM_i which is running on another resources pool. The communication behaves as follows.

(1) $VM_i.v \xrightarrow{in} m$

As before, the VM_i put the message v into its memory m.

(2) $VM_i \xrightarrow{generate} ioreq, VM_i.ioreq \xrightarrow{in} Sring$

Next, the front device of the VM_i generates an ioreq including the id of the request, the address and the size of the message. Then, the VM_i puts the ioreq into the I/O shared ring.

(3) $VM_i \xrightarrow{create} GR, VM_i.GR \xrightarrow{send} VM_i$

GR=<M-readonly, Dom0, m>.

The VM_i creates a GR, the grant type is the page mapping, the subflag is readonly, the destination id is the id of Dom0 and Fm is the page frame number of m. Then, the VM_i sends the GR to Dom0.

(4) $VM_i. \xrightarrow{notify} Dom0$

The VMi notifies the Dom0 that it has sent an ioreq through event channel.

(5) $Dom0 \xleftarrow{out} I/ORing.ioreq$.

When the Dom0 receives the notification, it will remove the ioreq from the I/O shared ring.

(6) $Dom0.ioreq \xrightarrow{in} Rqueue$

The Dom0 not process the ioreq immediately, and it puts the ioreq into the response queue.

(7) $Dom0.ioreq \xleftarrow{out} Rqueue$

The I/O requests in the response queue are processed using first come first serve (FCFS) strategy. When the ioreq is in the head of the response queue, it will be processed by the Dom0.

(8) $VM_i \xrightarrow{receive} GR, VM_i \xleftarrow{m} m$

The Dom0 receives the GR, it maps the memory m to its memory address space.

(9) $Dom0.v \xleftarrow{out} VM_i.m$

Once the Dom0 reads the ioreq, it can acquire the information of the ioreq. The Dom0 can get the message from the m.

(10) $Dom0 \xrightarrow{destroy} m$

When the Dom0 has got the data v, it removes the memory page from its memory address space.

(11) $VM_i \xleftarrow{recycle} VM_i.GR, VM_i \xrightarrow{destroy} GR$

The VM_i recycles the GR from the Dom0 and destroys the GR.

(12) $Dom0.v \xrightarrow{s} VM_i, Dom0 \xrightarrow{generate} iorep$

The Dom0 sends the message v to the VM_i uses the physical network device of the resources pool. Then, it generates an iorep including the id and the status of the iorep.

(13) $Dom0.iorep \xrightarrow{in} I/ORing, Dom0 \xrightarrow{notify} VM_i$

The Dom0 puts the iorep into I/O the shared ring and notifies the VM_i that it has sent an iorep through event channel.

(14) $VM_i \xleftarrow{out} I/ORing.iorep$.

When the VM_i receives the notification, it removes the iorep form the I/O shared ring.

6. Related Work

Our work is motivated by previous work on virtual resources management and scheduling, the description of physical machine resources and the description of the system framework.

6.1 Virtual Resources Management and Scheduling

There has been recent work on improving I/O performance or I/O fairness. HwanjuKim et al. [16] present a virtual machine scheduling technique for transparently bridging the semantic gap between the VMM and Guest OSes in order to improve I/O performance without compromising CPU fairness. Task-aware scheduling [17] and Communication-aware scheduling [4] are both scheduling strategies in order to improve I/O performance. The building of a virtual router platform is investigated to ensure I/O fairness [7]. There are also some work on dynamically allocate CPU resources. Fernando Rodríguez-Haro et al. [18] present a resources management approach to support dynamically change resource requirements of VMs with monitoring and control primitives. XMM [19] is a resources management tool to support real time task.

6.2 The Description of Physical Resources

A number of formal languages, equipped with a formal semantic, have been introduced, like communicating sequential process [20] and π-calculus [21]. C.A.R Hoare [20] first introduces the basic concept and the behavior of the process. Then, the concurrency behavior of processes is introduced, and the algebraic laws which describe the interaction of processes are proposed. Thirdly, a complete mathematical definition of the concept of a nondeterministic process and techniques for avoiding nondeterministic are proposed. Fourthly, the communication among processes and resources which are shared among many processes are described. CSP provides clear assistance to the programmer in his tasks of specification, design, implementation, verification and validation of complex computer systems.π-calculus also formally describe the communication and concurrency behavior among processes. These work is similar to our work. However, their work mainly focuses on effectively facilitates the construction of safe and reliable software. Our work mainly focuses on effectively reason about virtual resources management and manipulation. Formal description of programming languages is mainly used to improve the security and stability of the language. Avinash Malik et al. [22] present a programming language and give its formal model of computation, formal syntax and semantics. Direct memory manipulation through pointers is essential in many applications. The misuse of pointers will cause program errors. Dengping Zhu and Hongwei Xi [23] describe resources use a light-weighted formal method to support safe programming with pointers. R. Shi and H. Xi [24] present an approach to safe multi-core programming in ATS. They formalize a type system that can guarantee safe manipulation of resources on multi-core machines.

6.3 The Description of the System Frame

MarcAiguier et al. [25] define a unified framework for modeling abstract components, as well as a formalization of integration rules to combine their behavior. The authors of [26] present a formal description of an advanced management infrastructure able to provide end user with pseudonymity, identity aggregation, cross-layer singlesign-on and advanced authorization decisions. The authors of [27] establish a formal fuzzy reasoning system and the associated semantics. Ruqayya Abdulrahman et al. [28] construct a formal specification of an OSNRS to help us investigate the feasibility of using Multi Agent System technology in retrieving historical information from Online Social Networks sites.

7. Conclusions and Future Work

This paper presents a formal description of virtual resources system framework. We have defined the resources pool and virtual machine. We then defined the operation semantic of VMs. Next, we described virtual resources of VMs and how they used virtual resources. The formal description of virtual resources can offer the potential to reason about the correctness of virtual resources management and manipulation.

In the future, we plan to describe the virtual resources share behavior to avoid conflict. We also plan to use the operation semantic to describe Qos influence among VMs and provide Qos guarantee mechanism for VMs.

Acknowledgements

This paper is a revised and expanded version of a paper entitled [A Network Resources Scheduling Method for Virtual Machine] presented at [The 3rd International Conference on Next Generation Computer and Information Technology, October 24-26, 2014 at Hochimin, Vietnam].

This work is supported by Doctor Foundation of Henan Institute of Engineering, Foundation of He’nan Educational Committee (13A520148), and a grant from National Science Fund for Young Scholars （No. 61301232).

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, “Xen and the art of virtualization”, ACM SIGOPS Operating Systems Review, vol. 37, **(2003)**, pp. 164-177.

[2] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor”, Proceedings of the Linux Symposium, **(2007)**, pp. 225-230.

[3] H. J. Hwanju Kim and Joonwon Lee, “XHive: Efficient Cooperative Caching for Virtual Machines”, IEEE Transactions on Computers, vol. 60, **(2011)**, pp. 106-119.

[4] S. Govindan, J. Choi, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: communication-aware CPU management in consolidated Xen-based hosting platforms", Computers, IEEE Transactions, vol. 58, **(2009)**, pp. 1111-1125.

[5] W. Lixi, X. Jing, Z. Ming, T. Yicheng, and J. A. B. Fortes, “Fuzzy Modeling Based Resource Management for Virtualized Database Systems,Proceedings of the 2011 IEEE 19th International Symposium on Modelling”, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2011) , **(2011)**, pp. 32-42.

[6] Y. Song, Y. Z. Sun, H. Wang, and X. Song, “An adaptive resource flowing scheme amongst VMs in a VM-based utility computing”, 7th IEEE International Conference on Computer and Information Technology, **(2007)**.

[7] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, “Friendly virtual machines: leveraging a feedback-control model for application adaptation”, Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments, **(2005)**, pp. 2-12.

[8] M. B. Anwer, A. Nayak, N. Feamster and L. Liu, “Network I/O fairness in virtual machines”, Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures, **(2010)**, pp. 73-80.

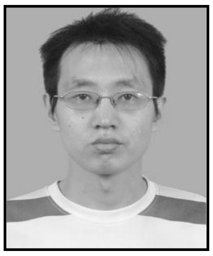
[9] X. Liu, W. Tong and C. Shen, “QoS-aware I/O schedule for virtual machines in cloud computing environment”, The Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, **(2013)**.

[10] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, “Symbolic Model Checking: 10^{20} States and Beyond”, Proc. IEEE Fifth Ann. Symp. Logic in Computer Science (LICS ’90), pp. 428-439, **(1990)**.

[11] K. J. Duda and D. R. Cheriton, “Borrowed-virtual-time (bvt) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler,” Proceedings of the 17th ACM symposium on Operating Systems Principles (SOSP’99) , **(1999)**, pp. 261–276.

[12] I. Leslie, D. Mcauley, R. Black, T. Roscoe, P. 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, vol. 14, pp. 1280–1297, **(1996)**.
[13] Credit scheduler, http://xen.org/files/summit_3/sched.pdf.
[14] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server", OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, **(2002)**, pp. 181-194.
[15] J. Liu, W. Huang, B. Abali, D. K. PandaHigh, "Performance VMM-Bypass I/O in Virtual Machines", USENIX 2006 Annual Technical Conference.
[16] H. Kim, H. Lim, J. Jeong, H. Jo, J. Lee, and S. Maeng, "Transparently bridging semantic gap in cpu management for virtualized environments", Journal of Parallel and Distributed Computing, vol. 71, **(2011)**, pp. 758-773.
[17] H. Kim,H. Lim, J. Jeong, H. Jo, J. Lee, "Task-aware virtual machine scheduling for I/O performance", Proc. of 2009 ACM SIGPLAN/ SIGOPS International Conferenceon Virtual Execution Environment, VEE'09, **(2009)**; Washington, DC, USA.
[18] F. Rodriguez-Haro, F. Freitag, and L. Navarro, "Enhancing virtual environments with QoS aware resource management", Annales Des Telecommunications-Annals of Telecommunications, vol. 64, **(2009)**, pp. 289-303.
[19] B. K. Kim, Y. J. Yoo, C. Yoo, and Y. W. Ko, "Design and Implementation of Resource Management Tool for Virtual Machine, in Computer Applications for Communication", Networking, and Digital Contents, vol. 350, Springer-Verlag Berlin, **(2012)**, pp. 17-24.
[20] C.A.R Hoare, "Communicating sequential processes, the origin of concurrent programming", **(2002)**, pp. 413-443.
[21] R. Milner, "Communication and Concurrency", Prentice-Hall, Inc., **(1989)**.
[22] A. Malik, A. Girault, and Z. Salcic, "Formal Semantics, Compilation and Execution of the GALS Programming Language DSystemJ", Ieee Transactions on Parallel and Distributed Systems, vol. 23, **(2012)**, pp. 1240-1254.
[23] D. Zhu and H. Xi, "Safe programming with pointers through stateful views", Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages, **(2005)**, pp. 83-97.
[24] R. Shi and H. Xi, "A linear type system for multicore programming in ATS", Science of Computer Programming, **(2012)**.
[25] M. Aiguier, F. Boulanger and B. Kanso, "A formal abstract framework for modelling and testing complex software systems", Theoretical Computer Science, vol. 455, **(2012)**, pp. 66-97.
[26] A. Peacuterez, G. Loacutepez, O. Caacutenovas and A. F. Goacutemez-Skarmeta, "Formal description of the SWIFT identity management framework", Future Generation Computer Systems, vol. 27, **(2011)**, pp. 1113-1123.
[27] Z. Zaiyue, Y. Sui, C. Cungen, and W. Guohua, "A formal fuzzy reasoning system and reasoning mechanism based on propositional modal logic", Theoretical Computer Science, vol. 368, vol. 5, **(2006)**, pp. 149-160.
[28] R. Abdulrahman, D. R. W. Holton, D. Neagu, and M. Ridley, "Formal Specification of Multi Agent System for Historical Information Retrieval from Online Social Networks", Agent and Multi-Agent Systems: Technologies and Applications. vol. 6682, Springer-Verlag Berlin, **(2011)**, pp. 84-93.

Authors



Xiaodong Liu, he received his Ph.D. degree from Shanghai University. He is currently an associate Professor of Henan Institute of Engineering. His primary research interests cover virtualization, cloud computing and grid computing

Xiaodong Liu, he received his Ph.D. degree from Sun Yat University. He is currently an Instructor of Henan Instituteof Engineering. His primary research interests cover swarm intelligence, things of internet.

Xiaodong Liu, he is currently an assistant of Henan Institute of Engineering. His primary research interests is cloud computing.