TSAC: Enforcing Isolation of Virtual Machines in Clouds

Chuliang Weng, *Member, IEEE*, Jianfeng Zhan, *Member, IEEE*, and Yuan Luo, *Member, IEEE*

Abstract—Virtualization plays a vital role in building the infrastructure of clouds, and isolation is considered as one of its important features. However, we demonstrate with practical measurements that there exist two kinds of isolation problems in current virtualized systems, due to cache interference in a multi-core processor. That is, one virtual machine could degrade the performance or obtain the load information of another virtual machine, which running on a same physical machine. Then we present a time-sensitive contention management approach (TSAC) for allocating resources dynamically in the virtual machine monitor, in which virtual machines are controlled to share some physical resources (e.g., CPU or page color) in a dynamical manner, in order to enforce isolation between the virtual machines without sacrificing performance of the virtualized system. We have implemented a working prototype based on Xen, evaluated the implemented prototype with experiments, and experimental results show that TSAC could significantly improve isolation of virtualization. Specifically, compared to the default Xen, TSAC could improve the performance of the victim virtual machine by up to about 78 percent, and perform well in blocking its cache-based load information leakage.

Index Terms—Cloud, virtual machine, isolation, performance, access control, scheduling

1 Introduction

With the development of computer and network technologies, cloud computing [1] is considered as a new infrastructure for deploying applications and can provide a variety of resources and services to users on demand. Examples of clouds include Amazon's Elastic Compute cloud, Microsoft's Azure Platform, IBM's Blue cloud. In typical instances of clouds, virtualization technology [2], [3] is an important component of the infrastructure. Virtualization technology can aggregate the functionality of multiple standalone computer systems into a single hardware computer, in order to promote the usage of the hardware while decreasing power consumption. Differing from the traditional system software stack, a virtual machine monitor (VMM) is inserted between the operating system level and the hardware level in the virtualized system. Currently, typical examples of system virtulization include Xen [4], [5], VMWare [6], [7], KVM [8], Hyper-V [9], and VirtualBox [10].

When multiple VMs run simultaneously on a physical server for efficiency, they have to share the common hardware resources, in the manner of time multiplexing or space sharing. State-of-the-art VMMs such as Xen have provided a basic isolation between VMs, in the aspect of CPU and memory, discussed in Section 2. In the ideal condition, applications running in a VM should have no impact on the performance of applications running in other VMs, when they run simultaneously on a hardware server. That is, virtualization should provide a good isolation between VMs. It is not a good practice to allow a VM with a heavy workload to appropriate the resources of other VMs without any limitation.

Applications running on multi-core processors suffer from poor performance isolation [11], [12], [13]. Cache interference becomes a well-known problem and also has been well studied. There are a lot of research efforts on how to deal with cache contention between processes in operating systems. Differing from resource management in operating systems in the non-virtualization scenario, although the VMM is in control of allocating physical resources, it lacks knowledge of processes in the guest operating systems, that is the semantic gap [14]. As a result, eliminating the negative influence of cache interference on isolation between VMs seems to be a non-trivial challenge. Differing from those methods, in this paper, we focus on the virtualized multi-core system, and try to mitigate the negative influence of cache interference on isolation provided by virtualization.

We say that VM V_α is *cache-unfriendly* to VM V_β if V_α could degrade performance of V_β, or V_α could estimate the current load of V_β (i.e., the load information leakage, which could be used to implement a side channel), when they running together. It is the sharing of physical resources without any effective restriction that weakens isolation between VMs. For example, state-of-the-art VMMs such as Xen dynamically map all virtual CPUs (VCPU) to all physical CPUs (PCPU). Two VCPUs from different VMs may be assigned to two cores in the same chip, or the two VCPUs may share a single core by turns. Then the shared cache becomes a carrier in the virtualized system, through which a VM with a specific application could degrade the performance or estimate the load information of the other VM (see Section 3.3 for details), although the default isolation is adopted. It indicates that current isolation methods are helpless to deal

- *C. Weng is with the Shannon Laboratory, Huawei Technologies Co., Ltd. E-mail: wengchuliang@huawei.com.*
- *J. Zhan is with the Institute of Computing Technology., Chinese Academy of Sciences. E-mail: zhanjianfeng@ict.ac.cn.*
- *Y. Luo is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. E-mail: luoyuan@cs.sjtu.edu.cn.*

Manuscript received 1 June 2013; revised 26 Jan. 2014; accepted 16 Mar. 2014. Date of publication 7 May 2014; date of current version 8 Apr. 2015. Recommended for acceptance by J. Cao.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2322608

with the cache-unfriendly problem between VMs, actually this issue had been raised for some time [15].

Isolation is considered as one of important features provided by virtualization; however there exist problems in this kind of isolation. The motivation of this paper is to enforce isolation of virtualization, and we propose a dynamic contention management approach (TSAC) for the VMM to map VCPUs onto PCPUs and allocate memory among VMs. To the best of our knowledge, TSAC is the first to implement low-level access control in the VMM for reinforcing the basic isolation in the virtualized system. The major contributions we make in this paper are as follows.

- We demonstrated that cache contention in multi-core systems could weaken the fundamental feature of isolation provided by virtualization. Specifically, a VM could degrade performance or estimate the load of other VMs, when running together. State-of-the-art VMMs such as Xen are helpless to handle the cache-unfriendly problem between VMs.
- The concept of access control is introduced to the resource management in the VMM, and a time-sensitive access control (TSAC) framework is presented for allocating resources in the VMM, in order to avoid having two cache-unfriendly VMs share these physical resources within a certain time, rather than forbidding them to share them all the time.
- Differing from the existing static method, dynamic contention management strategies are proposed and designed for two kinds of physical resources, PCPU-set and page color, based on the TSAC framework, respectively. The former implements a coarse-grained isolation, while the latter achieving a fine-grained isolation.
- A working prototype is implemented based on the open source Xen, and the modifications to Xen are highly localized and relatively small. We have evaluated the working prototype on standard benchmarks, and it is demonstrated that TSAC could significantly improve isolation of virtualization, not only improve the performance of the victim VM by up to about 78 percent, but also perform well in blocking its cache-based load information leakage.

The rest of this paper is organized as follows. The next section provides the background of this research work. Section 3 analyzes cache-unfriendly problems in virtual machine systems. Section 4 presents TSAC in the VMM. Section 5 provides our detailed design and implementation, and Section 6 evaluates it. Section 7 provides a brief overview of related work, and Section 8 concludes the paper.

2 Background

In this section, we introduce the basic isolation of physical resources between VMs in a virtualized system, while discussing pros and cons of these existing methods.

2.1 Basic CPU Isolation

There are four privilege levels in the ×86 processor architecture, which are numbered from zero (most privileged)

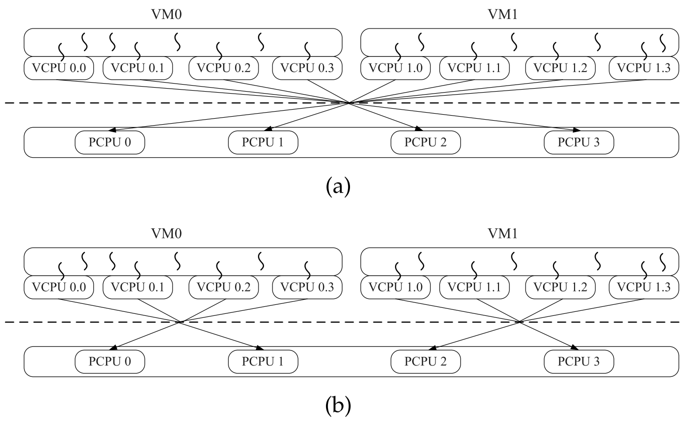


Fig. 1. (a) Time-multiplexing CPU without restriction, (b) a VCPU restricted to a specific set of PCPUs.

to three (least privileged). Generally, operating systems on the ×86 processor architecture are running on top of the hardware in Ring-0, and applications are running in Ring-3. In a virtualized system, a VMM is inserted between the hardware and guest operating systems. Consequently, the VMM runs in Ring-0, and guest operating systems are modified to run in Ring-1 or Ring-2, which have not been used by any well-known x86 operating system since OS/2. This provides isolation and protection to a VM from other VMs and keeps the VM from executing high-privileged Ring-0 instructions, while the guest operating system being kept safely isolated from applications running in Ring-3.

Besides isolation and protection in the aspect of instructions, VCPUs in VMs are carried out by the VMM on the underlying PCPUs. In order to achieve a near-native performance, the CPU scheduling in the VMM should avoid the waste of any processing cycle. Currently, there are two mapping methods of VCPUs on PCPUs, illustrated as Fig. 1. One method is that all PCPUs in the system are multiplexed by all VCPUs from different VMs, which is the default means in Xen. The other is that a VCPU can only be mapped to some particular PCPUs, which is provided by the interface xm vcpu-pin" in Xen.

Intuitively, relatively strong isolation exists in the second method, which is originated from IBM's static LPAR (logical partition) in IBM System/390 [3], and now is used in IBM's DLPAR (dynamic logical partitioning). However, the restriction of a VCPU running on particular PCPUs is statically set by the command or the configure file associated with the VM, which is not conducive to take full advantage of CPU resources. To the best of our knowledge, there is no mechanism in existing VMMs, by which the VMM could dynamically restrict particular PCPUs for a VCPU, in order to enforce isolation between VMs without sacrificing performance of the virtualized system.

2.2 Basic Memory Isolation

Now we turn our attention on memory virtualization and isolation. In a traditional system, there are typically two address spaces, that is the virtual address space and the physical address space. The operating system manages the mapping from the virtual address space to the physical address space by the Memory Management Unit (MMU).

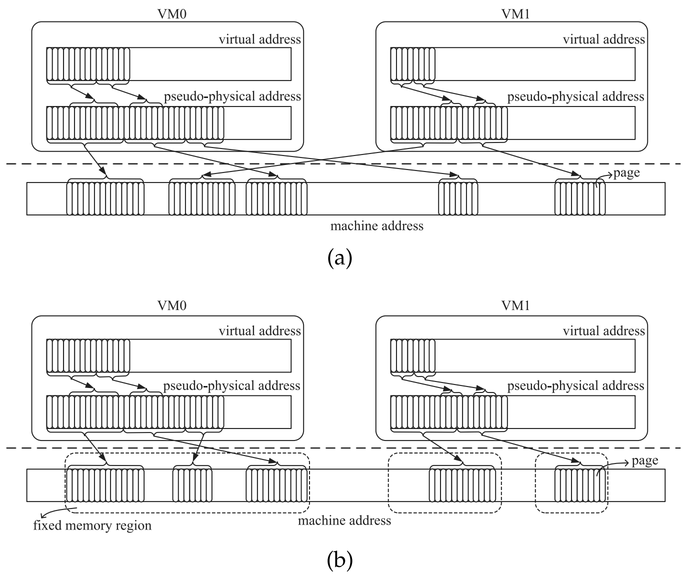


Fig. 2. (a) Memory allocation without restriction, (b) memory allocation restricted to specific memory regions.

The operating system maintains a page table for each process that maps its each virtual page to a physical page. In the virtualized system, another page table is added for translating the physical address space into the machine address space, because the physical address space in the VM is an illusion to the guest operating system provided by the VMM, actually it is the pseudo-physical address space. It is this pseudo-physical-to-machine mapping that implements the virtualization of memory. The VMM must ensure that no two VMs access the same memory area. Each page or directory table update in VMs must be validated by the VMM to ensure that VMs only manipulate their own tables, in order to maintain memory isolation between VMs.

In the current system virtualization, for maximizing memory utilization, generally each VM has a maximum and current consumed physical memory allocation, and can adjust its current memory allocation up to the maximum limit configured. Memory is allocated and freed dynamically at a granularity of the page level, and the VMM cannot guarantee that a VM will receive a contiguous allocation of physical memory to use. There are also two possible ways of allocating memory to VMs in the VMM, illustrated as Fig. 2. One way is that a page in the pseudo-physical address can be mapped to any machine address except that is reserved for the VMM. This kind of memory allocation is adopted in Xen. The other way is that a page in the pseudo-physical address is restricted to a fixed machine memory region as in IBM's LPAR, which is set by the administrator at the creation of a VM.

We have an intuitive sense that there exists potential vulnerability in the first memory allocation way, because of pages from VMs being arbitrarily mapped to the same physical memory. However, the second memory allocation way can overcome vulnerability, with the cost of sacrificing memory utilization. As far as we know, there is also no access control mechanism in existing VMMs, by which the VMM could dynamically allocate memory pages to VMs, in order to inhibit the potential isolation problems without sacrificing the performance.

3 Isolation Issues

In this section, we present measurements on a real system to demonstrate that issues on isolation are possible in the multi-core system with the VMM Xen, which motivate us to present TSAC in the following sections.

3.1 CPU Scheduling in Xen

Before presenting issues on isolation among VMs in Xen, we would like to introduce the existing performance isolation solution in Xen. The default scheduling algorithm in Xen is the *Credit* algorithm [16]. This algorithm is a kind of *proportional share* strategy, featuring automatic workload balancing of VCPUs across PCPUs on a multi-core system. The scheduling algorithm can efficiently achieve global workload balancing on a symmetric multiprocessing (SMP) system.

In Xen, each VM is associated with a *weight* and a *cap*. Each VM will get CPU time in proportion to its assigned weight; for example, a VM with a weight of 256 will get twice as much CPU as a VM with a weight of 128 on a contended host. When the cap is 0, a VM can receive extra physical CPU time; this is called work-conserving mode [17]. A non-zero cap indicates non work-conserving (NWC) mode [17], in which the CPU time obtained by a VM is strictly in proportion to its weight, and it cannot receive any extra CPU time. Therefore, the isolation in Xen is implemented by the NWC mode.

The credits of all runnable VMs are recalculated at an interval of 30 ms, mainly in proportion to weights assigned. The basic unit time (*tick*) of scheduling is 10 ms, and the credit of the running VCPU is decreased every 10 ms. When normal applications run on VMs, this algorithm achieves good isolation between VMs.

3.2 Isolation Issues

In a virtualized system with multi-core processors, VMs usually share memory caches in multi-core processors. As said earlier, VM V_α may be *cache-unfriendly* to VM V_β when running together. If V_α could degrade performance of V_β, we call V_α a Performance Hog VM (PH-VM). Besides, we call V_β a victim VM (victim-VM). With a certain memory access pattern, a PH-VM can occupy shared cache in the chip, and constantly evict useful cache content belonging to other VMs in the shared cache, preventing those VMs from using cache efficiently. The effect is that cache misses frequently occur on those VMs, and they will run with a high cache miss ratio. The similar method is also adopted by the related work [18], [19]. As a result, a PH-VM can severely degrade the performance of other VMs with which it is co-scheduled. A specific pseudo-code for the performance hog application running on a PH-VM is shown as follows, in which array a is far larger than the cache size.

```
//initialize array a[N] by malloc
while(1){
    for (i = 0; i < N; i += M){
        a [i] = i;
    }
}
```

Another kind of isolation issue is that V_α could estimate the current load of V_β when running together, due to the sharing of caches between VMs. Then, in this case we denote V_α as Eavesdropper VM (ED-VM). The ED-VM can measure cache activity on its physical machine, so it can estimate the current load of the machine and infer load information of other co-scheduled VMs. This kind of isolation issue can be used to implement a keystroke activity side channel between VMs [15]. Specifically, cache activity can be measured by the Prime + Trigger + Probe technique [20], [15], and the time (CPU cycles) of step Probe is the load sample.

3.3 Measurement

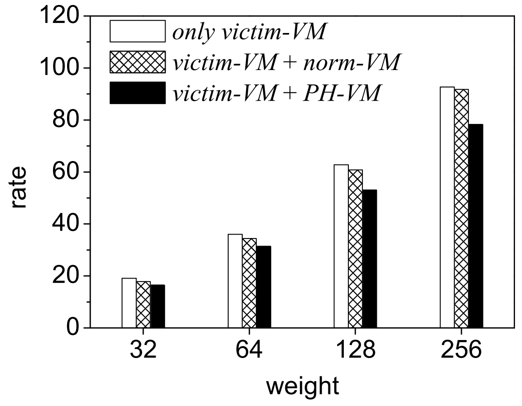
We run a PH-VM or ED-VM simultaneously with a victim-VM to measure its influence, and analyze whether the existing mechanism in Xen can inhibit the negative impact on isolation. All experiments were executed on a Dell Precision workstation, with dual quad-core Xeon X5410 CPUs and 8 GB of RAM. The virtualized system ran Xen 3.4.2, and all VMs ran the Fedora Core 6 Linux distribution with the Linux 2.6.18 kernel.

The victim-VM is configured with 4 VCPUs and 1,024 MB memory. The workload in the victim-VM is SPEC CPU2000 [21], and four copies of benchmarks run on it simultaneously using the rate metric in SPEC CPU2000. The SPEC *rate* metric measures the throughput of a VM by carrying out a number of similar tasks, and a higher rate indicates that the VM takes less time to run the tasks and thus delivers better performance. A PH-VM is also configured with 4 VCPUs and 1,024 MB memory, and four copies of the performance hog application (the pseudo-code shown in Section 3.2) run on it, and its weight is fixed as 256.

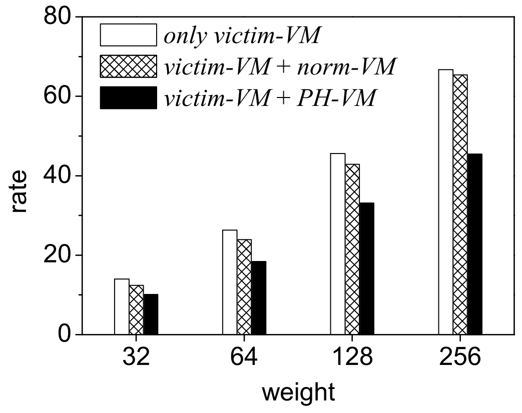
To measure the negative impact of a PH-VM on isolation, we first measure the throughput of a victim-VM when it runs alone in the system, and then we run a victim-VM and a PH-VM simultaneously in NWC mode. We also run a victim-VM and a normal VM simultaneously in the NWC mode, where the normal VM is configured the same as the victim-VM, except for its weight being fixed as 256. The workload in the normal VM (denoted as *norm-VM*) is also SPEC CPU2000 with 4 copies. The result is shown as Fig. 3.

In Fig. 3, we observe that: (1) the rate of the victim-VM running alone in the system increases along with its weight, which indicates that a VM gets CPU time in proportion to its weight; (2) comparing the rate of *only victim-VM* with *victim-VM + norm-VM*, the normal VM has little impact on the performance of the victim-VM when they are co-scheduled in a physical system; (3) the performance of the victim-VM is degraded significantly when it runs simultaneously with the PH-VM, even if the total of running VCPUs equals the number of PCPUs, and the two VMs run in the NWC mode, which is the most stringent isolation mechanism provided by Xen; (4) the greatest performance degradation of the victim-VM induced by the PH-VM is observed when its weight is 256.

Now we consider the load information leakage induced by a ED-VM through sharing cache between two VMs. In the measurement, a ED-VM is configured with one VCPU and 1,024 MB memory, and runs simultaneously with a victim-VM in the NWC mode. First, with no workload on the victim-VM, we measure the time of the Probe step [20] in



(a)



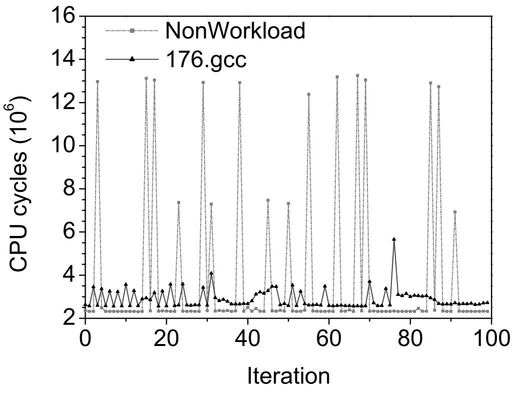
(b)

Fig. 3. The performance of the victim-VM: (a) workloads on the victim-VM and the norm-VM are 176.gcc; (b) workloads on the victim-VM and the norm-VM are 256.bzip2.

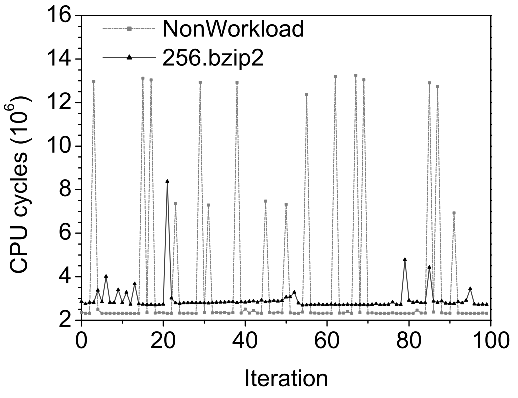
the ED-VM, which takes samples to spy on the load information of the victim-VM. In this case, the 100 continuous samples (denoted as `NonWorkload`) are shown as the dotted line curve in Fig. 4. Second, the ED-VM runs simultaneously with the victim-VM, and the workload in the victim-VM is SPEC CPU2000 (i.e., 176.gcc or 256.bzip2) with four copies. We also measure the time of the Probe step, and the 100 continuous samples tested on the ED-VM are shown as the solid line curve in Fig. 4.

According to the results depicted in Fig. 4, we observe that: (1) in `NonWorkload`, the majority of samples are less than 2.5×10^6 cycles with a steep rise in an interval; (2) the values of samples are larger than 2.5×10^6 cycles when the workload is SPEC CPU2000, which is due to contending for the shared cache between the ED-VM and the victim-VM; (3) according to the characteristics of samples in the two cases, the ED-VM can easily detect the load information of the VM running simultaneously with it on the same physical machine, even though the NWC mode is adopted. Moreover, the steep rise in the value of samples is caused by migration of the ED-VM's VCPU on PCPUs, and its frequent occurrence is because the VCPU will run on different PCPUs for load balancing according to the scheduling manner in Xen.

In summary, the above two measurements on the physical machine show that the basic isolation mechanisms in Xen cannot handle the cache-unfriendly problem between VMs.



(a)



(b)

Fig. 4. Samples on the ED-VM: (a) the workload on the victim-VM is 176.gcc, (b) the workload on the victim-VM is 256.bzip2.

4 Isolation Framework

This section presents an isolation framework and two strategies for VMs in the virtualized system; specifics regarding their implementation are detailed in the following section.

4.1 Design Issues

Current virtualization technology such as Xen provides basic isolation between VMs in a physical machine. However, the shared cache becomes a carrier for the cache-unfriendly operations between VMs in a virtualized multicore system, and we have demonstrated that they are not inhibited in the current VMM. The reason is that caches in the physical CPU are shared by VCPUs from different VMs, with little limitation for isolation. To enforce isolation, we may resort to avoiding the sharing of caches between two VMs if one is cache-unfriendly to the other.

At present, the overhead of virtualization is relatively small. However, additional cost may be unavoidably introduced when a reinforced isolation mechanism is adopted in the virtualized system. Therefore, isolation between VMs should be enforced without sacrificing performance of the virtualized system, and the goal of our work is to design an isolation framework that satisfies both isolation and performance requirements.

To enforce isolation while maintaining performance, differing from the existing methods, a dynamic isolation mechanism will be adopted in this paper. For example, when one VCPU has a potential cache-unfriendly operation on another, a static isolation mechanism, such as the method in the default Xen, fixes each to a different PCPU. Instead, we forbid them to use the same PCPU within a certain time interval. As there are usually a variety of VMs in a virtualized system, it is a large challenge for a VMM with a dynamic contention management mechanism to achieve the goals of isolation, performance and fairness among VMs.

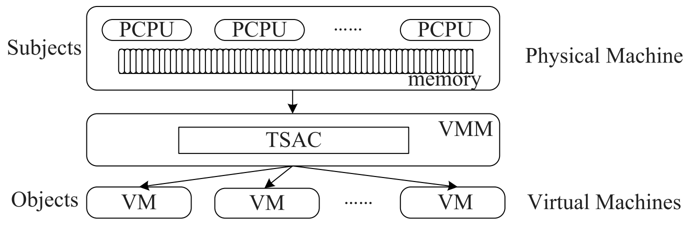


Fig. 5. The isolation framework of virtual machine systems.

4.2 Isolation Framework

In the Chinese wall access control policy [22], access to data is constrained by what data the subject already holds access rights to instead of by attributes of the data in question. All objects concerning the same corporation are grouped into a data set, and data sets belonging to corporations in competition are grouped into a *conflict of interest class*. A subject is allowed to access at most one data set belonging to each such conflict of interest class. It is a simple but effective access control mechanism, which is helpful to maintain the performance, while being also helpful for modification in response to specific requirements. So we design the isolation framework based on it.

In a virtualized system, to enforce isolation between VMs, a physical resource (such as a PCPU) can be allowed to run virtual components (such as VCPUs) from only one of a set of VMs, in which one has negative impact on another. Therefore, we define VMs running various workloads as *object*s, and group those VMs which may be cache-unfriendly to the others into a *conflict of interest class*, and hardware resources (such as PCPUs, memory pages, and I/O adapters) are treated as *subject*s.

Then we extend the Chinese wall policy to resource allocation in the VMM, and present an approach Time-Sensitive Access Control. We consider two kinds of physical resources, PCPUs and memory pages, which may be implicated in cache-unfriendly operations between VMs. First, we define a conflict VM group (CVG) as counterpart to a conflict of interest class; how to classify VMs to different CVGs can be determined according to research efforts [23]. Second, the *contention management strategy* is that a physical resource can work for at most one VM in each CVG within a certain time, with which the assignment of hardware resources to VMs must be made to comply. It should be noted that a shared physical resource will work for at most one VM in each CVG within a period of time, rather than all the time as in the original Chinese Wall policy. Then the isolation framework for virtual machine systems is shown as Fig. 5.

The procedure of subjects accessing objects is represented by the assignment of PCPUs or memory pages to VMs, and it is the VMM that carries out the mandatory strategy in TSAC. Based on the above isolation framework, two VMs in the same CVG are not allowed to access the

TABLE 1
Strategy TSAC-c for Assigning PCPUs to VCPUs

$c_{ij} \dashrightarrow v_{mn}$, if and only if, in the last δ_p ticks:
1) $\exists c_{ij'} \in Ps_i, \exists v_{mn'} \in C(V_i)$: $c_{ij'} \rightarrow v_{mn'}$;
or 2) $\forall c_{ij'} \in Ps_i, \forall V_{m'} \in x_k, \forall v_{m'n'} \in C(V_{m'})$: $c_{ij'} \nrightarrow v_{m'n'}$,
where $V_m \in x_k$.

same cache line in physical chips within a certain period, so cache-unfriendly operations between VMs are inhibited. To achieve this goal, we will turn our attention to the specific description of CPU and memory in following sections.

4.3 Enforced CPU Isolation

In a multi-core system, usually the last level cache (*LLC*, L2 or L3 cache) is shared by multiple cores in the same processor chip. To implement severe cache isolation, besides a PCPU not being allowed to run VCPUs from two or more VMs in the same CVG in turns, multiple PCPUs sharing cache will also not be allowed to run those VCPUs simultaneously. Therefore, we have to refine the definition of subjects from the viewpoint of physical CPU resources, and define PCPUs sharing LLC as a *PCPU-set*, which is actually a *subject* in TSAC.

Formally, a physical computer includes a group of PCPU-sets, denoted by $P = \{Ps_1, Ps_2, \ldots, Ps_{|P|}\}$, and the number of PCPU-sets is $|P|$. The set of PCPUs in the ith PCPU-set is denoted by $Ps_i = \{c_{i1}, c_{i2}, \ldots, c_{i|Ps_i|}\}$, where $|Ps_i|$ denotes the number of PCPUs in PCPU-set Ps_i. The VMs running on the physical computer are denoted by $V = \{V_1, V_2, \ldots, V_{|V|}\}$, and $|V|$ denotes the number of VMs in the system. We refer to X as the set of CVGs, and there are $|X|$ CVGs, and $X = \{x_1, x_2, \ldots, x_{|X|}\}$, and each x_i includes two or more VMs. The set of VCPUs in the ith VM is denoted by $C(V_i) = \{v_{i1}, v_{i2}, \ldots, v_{i|C(V_i)|}\}$, and $|C(V_i)|$ denotes the number of VCPUs in VM V_i.

The assignment of PCPUs to VCPUs is performed by the *scheduler* module in the VMM. According to TSAC, the scheduler can assign PCPU c_{ij} to VCPU v_{mn}, if and only if a PCPU in PCPU-set Ps_i has been assigned to run a VCPU from VM V_m in the last δ_p ticks, or no PCPU in PCPU-set Ps_i has been assigned to run any VCPU from any VMs in CVG x_k in the last δ_p ticks, where $V_m \in x_k$ and δ_p is a time threshold. Formally, the contention management strategy for CPU resources is specified as Table 1 (i.e., TSAC-c), which should be satisfied in the assignment of PCPUs to VCPUs by the scheduler in the VMM.

Where, $c_{ij} \dashrightarrow v_{mn}$ indicates that PCPU c_{ij} can be assigned to VCPU v_{mn}, and $c_{ij} \rightarrow v_{mn}$ indicates that PCPU c_{ij} has been assigned to VCPU v_{mn}, while $c_{ij} \nrightarrow v_{mn}$ indicates that PCPU c_{ij} has never been assigned to VCPU v_{mn}.

According to TSAC-c, two VMs in the same CVG will never share any cache with each other (i.e., *coarse-grained isolation*) within a specified period, so cache-unfriendly operations between the two VMs are inhibited. However, a VM may be blocked from running in the virtualized system because of lack of available PCPUs according to TSAC-c, especially when the number of PCPUs is relatively small. Therefore, we will introduce enforced memory isolation, with which two VMs in the same CVG will never share any cache line in LLC (i.e., *fine-grained isolation*) instead of never sharing the whole LLC in a processor chip, and cache-unfriendly operations are also inhibited.

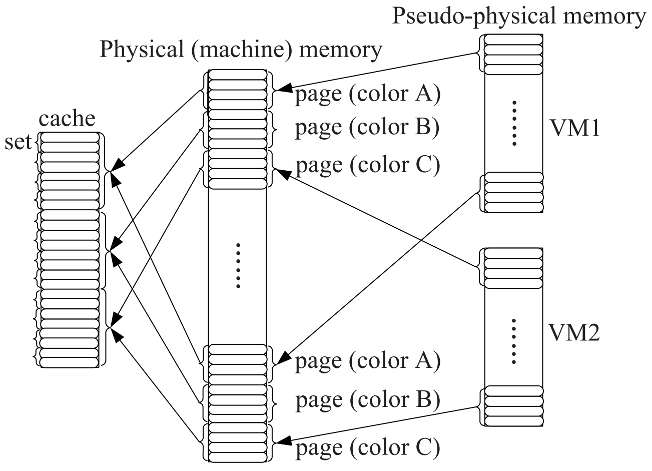


Fig. 6. An illustration of page coloring in the virtualized system.

4.4 Enforced Memory Isolation

To achieve fine-grained isolation between VMs, we will refer to the memory isolation in the presented isolation framework. As the proposed memory isolation is based on memory page coloring, first we briefly discuss the main features of page coloring technology.

In the memory cache of a modern processor, every page in the physical memory has a fixed mapping to a contiguous group of cache lines. Page coloring is a software technique [24], [25], [26] in traditional operating systems, by which memory pages being mapped to the same cache blocks are assigned the same color. Page coloring can also be applied to virtual machine systems, which is illustrated as Fig. 6. The VMM can control the color of pages assigned to a VM in the virtualized system, and cache blocks manipulated by the VMM are at a granularity of the page size times the cache associativity, which is the unit of cache space that can be allocated to a VM. When two VMs in the same CVG have to run on the same PCPU-set, to prevent one VM from degrading isolation through shared cache lines in LLC, the VMM can assign different page colors to them, then the two VMs will never share any cache line in LLC. As the cache-unfriendly operation is mainly related to LLC, it is then eliminated.

Therefore, we refine the definition of the subject from the aspect of physical memory resources, and define page colors instead of memory pages as *subject*s in TSAC. The physical memory MP includes a group of page colors, denoted by $MP = \{Mp_1, Mp_2, \ldots, Mp_{|MP|}\}$, where the number of page colors is $|MP|$. Mp_i denotes the set of memory pages in page color i, and $Mp_i = \{p_{i1}, p_{i2}, \ldots, p_{i|Mp_i|}\}$, where $|Mp_i|$ denotes the number of pages in page color i, and p_{ij} denotes a memory page in this page color. The assignment of memory pages to VMs is performed by the *memory manager* module in the VMM. According to TSAC, the memory manager can assign memory page p_{ij} in page color i to VM V_m, if and only if a memory page in page color i has been assigned to VM V_m in the last δ_m ticks, or no memory page in page color i has been assigned to any VM in CVG x_k in the last δ_m ticks, where $V_m \in x_k$ and δ_m is a time threshold. Formally, the

TABLE 2
Strategy TSAC-m for Assigning Memory Pages to VMs

$p_{ij} \dashrightarrow V_m$, if and only if, in the last δ_m ticks:
1) $\exists p_{ij'} \in Mp_i$: $p_{ij'} \rightarrow V_m$;
or 2) $\forall p_{ij'} \in Mp_i, \forall V_{m'} \in x_k$: $p_{ij'} \nrightarrow V_{m'}$, where $V_m \in x_k$.

contention management strategy for memory resources is specified as Table 2 (i.e., TSAC-m), which should be satisfied in the assignment of memory pages to VMs by the memory manager in the VMM.

According to TSAC-m, two VMs in the same CVG will never share any cache line in LLC with each other within a specified period, consequently cache-unfriendly operations between the two VMs are inhibited. We also recognize that the cache size actually utilized by VCPUs in a VM will be shrunk because memory pages in the VM are fixed to a part of the cache by the page coloring technology. However, the VMM can control the cache usage of a VM by page coloring, which is beneficial to reinforce isolation.

5 Detailed Design and Implementation

We have implemented a working prototype of TSAC, which is based on open-source Xen-3.4.2.

5.1 Isolation Design

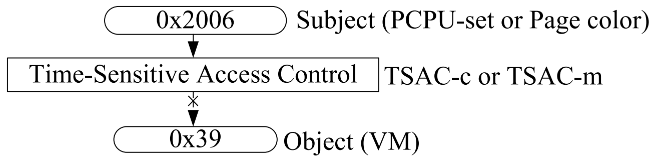
To implement the contention management strategies, the *access records* of subjects (i.e., PCPU-set or page color) should be kept by the VMM. We adopt a 16-bit record for each subject, which is divided into four groups of 4 bits, denoted by a hexadecimal string $g_0g_1g_2g_3$. There are 4 CVGs, and consequently each CVG may include a maximum of 16 VMs, which are indexed by the 4 bits. g_i denotes the ith CVG, and $g_i = 0$ for a subject indicates that the subject has never accessed any object in the ith CVG, therefore each CVG can actually include at most 15 VMs.

We define a 7-bit *label* for an object (i.e., VM), where the highest bit indicates whether the VM belongs to a CVG (0 is true), the next 2 bits specify the CVG, and the lowest 4 bits specify its *ID* in the CVG. For example, if a VM's label is 0 × 3a, it indicates that the VM belongs to the third CVG, and its ID is 10 in this CVG. A VM can belong to at most 4 CVGs, and correspondingly it has four labels.

The procedure for assigning PCPUs or memory pages to a VM is also an accessing procedure of subjects to an object, and the accessing procedure is validated by the scheduler or memory manager module according to TSAC. Specifically, the scheduler or memory manager module checks whether the object's labels and the subject's access record satisfy the contention management strategies in TSAC. For example, a subject (a PCPU-set or page color) with access record being 0 × 2006 cannot be allocated to a VM labeled as 0 × 39, because the subject had been used with another VM, whose ID is 6 in the third CVG, and therefore cannot be used with any other VM in the same CVG. This is shown as Fig. 7.

5.2 TSAC-c Implementation

In the default Xen, the system administrator can use the configuration file of a VM or the command `xm vcpu-pin` to set or change the PCPU affinity of VCPUs in the VM. Then we can control the PCPU affinity of VCPUs to avoid having two VCPUs from the two VMs in a CVG share the same PCPU-set, in order to achieve the goal of enforcing isolation from the aspect of CPU virtualization. However, the PCPU affinity is fixed once it is set, which usually degrades the system performance. Taking two VMs with four VCPUs in the same CVG for example, when they are simultaneously running on a machine with four PCPUs and their weights are equal, each VM has to run on two different PCPUs, which will result in performance degradation for multi-threaded workloads on the VMs [27].


0x2006
Subject (PCPU-set or Page color)
Time-Sensitive Access Control
TSAC-c or TSAC-m
0x39
Object (VM)
Fig. 7. An illustration of an access control for resource allocation in the VMM.

To inhibit cache-unfriendly operations induced by sharing caches, two VCPUs from the two VMs in a CVG are prohibited from running on the same PCPU-set at the same time or in turn. But they can still run on the PCPU-set in a certain interval while guaranteeing isolation, because a VCPU is frequently mapped to different PCPUs for load balancing, and it has little influence on the other through shared caches after a certain interval.

Therefore, we implement a flexible VCPU scheduling procedure illustrated as Algorithm 1 according to TSAC-c. In Algorithm 1, $count[V_m]$ is a counter for VM V_m, it equals δ_p when VCPU v_{mn} runs on PCPU-set Ps_i, and it will decrease one every tick if no VCPU from VM V_m runs on PCPU-set Ps_i. At each tick, the scheduler modifies the access record of PCPU-set Ps_i according to $count$, and also modifies the access record when a VCPU is selected to run.

Algorithm 1 VCPU scheduling for PCPU-set Ps_i

```
1: while each tick do
2:     for each VM $V_m$ do
3:         for each VCPU $v_{mn}$ in VM $V_m$ do
4:             if $v_{mn}$ is a running VCPU in $Ps_i$ then
5:                 $count[V_m] \leftarrow \delta_p$;
6:             end if
7:         end for
8:         $count[V_m] \leftarrow count[V_m] - 1$;
9:         if $count[V_m] = 0$ then
10:            clear the bit in the access record of $Ps_i$ corresponding to the label information of $V_m$;
11:        end if
12:    end for
13:    select VCPUs from run queues of PCPUs in $Ps_i$ according to TSAC-c;
14:    if VMs including the selected VCPUs are not in the access record of $Ps_i$ then
15:        add the label information of those VMs into the access record of $Ps_i$;
16:    end if
17:    schedule selected VCPUs to corresponding PCPUs in $Ps_i$;
18: end while
```

5.3 TSAC-m Implementation

To implement memory isolation with TSAC-m in Xen, we modify `alloc_domheap_pages` function in the memory manager module, which actually invokes `alloc_heap_pages` function. In Xen, the binary buddy memory allocation technique is used; this allocates a group of physically contiguous pages whose sizes are powers of two. To support this, Xen maintains lists of groups of contiguous free pages of size 1 to 2^{20} pages (level 0 to level 20), $free\_page\_list$. When a VM is created, its memory size mem_size is obtained from its configuration file, and pages in $free\_page\_list$ are allocated by `alloc_heap_pages` to the VM repeatedly at the granularity of the page level, until the total is achieved. To allocate a page in the specific page colors, we implement a new function called `pgcolor_get_page`, which is invoked by `alloc_heap_pages` to find and allocate a page in the specific page colors to a VM. The pseudocode of page allocating is shown as Algorithm 2.

Algorithm 2 Page allocating for VM V_m

```
1: $VMp\_index \leftarrow \varnothing$;
2: $VMp \leftarrow \varnothing$;
3: while ($|VMp| < page\_num$) do
4:     Select page color $i$ from $MP$ according to TSAC-m;
5:     $VMp \leftarrow VMp \cup Mp_i^a$;
6:     $VMp\_index \leftarrow VMp\_index \cup \{i\}$;
7: end while
8: $j \leftarrow 0$;
9: while ($j < page\_num$) do
10:    for each $level$ in $free\_page\_list$ do
11:        for each $page\_set$ of $2^{level}$ contiguous pages do
12:            if a page's color in $page\_set$ belongs to $VMp\_index$ then
13:                delete $page\_set$ from $free\_page\_list$;
14:                allocate the page for $V_m$;
15:                insert other pages in $page\_set$ into $free\_page\_list$;
16:            end if
17:        end for
18:    end for
19:    $j \leftarrow j + 1$;
20: end while
```

In Algorithm 2, VMp_index denotes the set of page colors allocated to VM V_m, which is determined by TSAC-m according to access records of page colors and the VM's label information. VMp denotes the set of pages, which can be allocated to VM V_m. Moreover, δ_m in TSAC-m is set to be a runtime of the VM. That is to say, the set of page colors assigned to a VM is fixed until it is shut down. $page\_num$ denotes the number of pages allocated to VM V_m, which could be calculated through mem_size and the size of a page. Mp_i^a denotes the set of free pages in page color i. In addition, in Algorithm 2, when other pages in $page\_set$ are inserted into $free\_page\_list$, the binary buddy allocation technique will also be adopted, to avoid excessive external fragmentation in the physical memory.

6 Evaluation

In this section, we give a comprehensive evaluation and analysis on the prototype system of TSAC. All experiments were executed on a Dell workstation with dual quad-core Xeon X5410 CPUs and 8 GB of RAM. There are three kinds of VMs, victim-VM, PH-VM, and ED-VM, and the configuration details of PH-VM and ED-VM are in Section 3.3. All VMs' weights are 256, and they run in the NWC mode. In these Xeon CPUs, every two cores share an L2 cache, therefore there are two PCPU-sets in a processor according to TSAC-c. Consequently, there are 4 PCPU-sets, [0, 1, 2, 3], in the system with dual processors. In these processors, the capacity of an L2 cache shared by two cores is 6 MB, and the cache associativity is 24, therefore there are at most 64 page colors in the system. Besides, we denote *online ratio* as the percentage of time of a VCPU being mapped to a PCPU. In experiments, the number of VCPUs in all VMs equals 4, so that *online ratio* $= \frac{|P|\times 2}{|V|\times 4} = \frac{8}{|V|\times 4}$, where $|P|$ and $|V|$ are defined in Section 4.3.

6.1 Evaluation with PH-VM

We have demonstrated that the negative impact of cache-unfriendly operations on performance is possible in a virtualized system with multi-core processors in Section 3. In this section, we will examine the enhanced isolation of performance by TSAC in undercommitted systems and overcommitted systems.

6.1.1 *Undercommitted System*

When the total of running VCPUs is not more than the number of PCPUs, we call this virtualized system a *undercommitted* system. In this scenario, (1) a victim-VM runs simultaneously with a PH-VM in the default Xen, and we test its performance (denoted as `Xen`); (2) we test the performance of a victim-VM (denoted as `TSAC-c` when TSAC-c is adopted or `TSAC-m` when TSAC-m is adopted), when it runs simultaneously with a PH-VM in the prototype system of TSAC. When the victim-VM and the PH-VM are running simultaneously in the prototype system, they belong to the same CVG. Besides, (3) we test the performance of a victim-VM (denoted as `vcpu-pin`, when its VCPUs are mapped to two PCPU-sets through `xm vcpu-pin` in Xen, while a PH-VM running on the remaining two PCPU-sets in the system.

First, we test the performance of a victim-VM with SPEC CPU2000 [21]. The SPEC *rate* metric is adopted, and the configuration details are in Section 3.3. Fig. 8 shows that not only TSAC-c but also TSAC-m can mitigate the performance degradation for the majority of benchmarks running on the victim-VM. To be more specific, the rate is improved by up to about 78 percent for 181.mcf, and about 15 percent on average, compared to `Xen`. The reason is that the potential cache-based performance influence can be mitigated by either enhanced CPU isolation or enhanced memory isolation. Fig. 8 also shows that `vcpu-pin` performs well as TSAC, as VCPUs from two unfriendly VMs are mapped to different PCPU-sets, and will never share any cache line in LLC.

Second, the NAS parallel benchmarks [28] run on a victim-VM, and we test their runtimes when the victim-VM and a PH-VM run simultaneously as the above two cases. In order to provide an intuitive comparison, we define the *runtime ratio* of a benchmark in `TSAC-c` or `TSAC-m`, as a ratio of its runtime to the runtime of the same benchmark in `Xen`. As depicted in Fig. 9, both TSAC-c and TSAC-m could

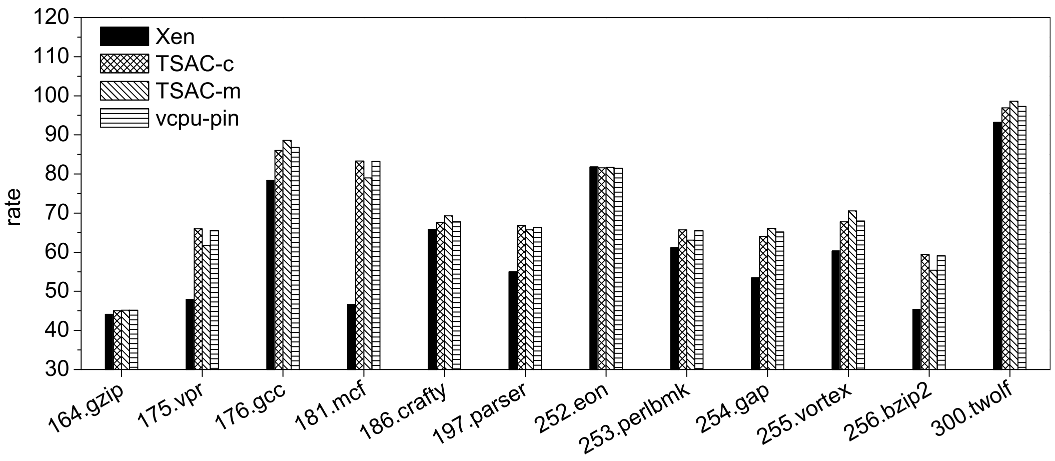


Fig. 8. The performance of a victim-VM, where workloads are SPEC CPU2000, and *online ratio* = 100%.

reduce the runtime of CG by up to about 50 percent, and reduce the runtime of all NAS parallel benchmarks by about 18 percent on average, compared to the default Xen.

Besides, we recognize that an additional overhead is introduced by TSAC-m due to the non-contiguous memory allocation procedure. Algorithm 2 has to perform many times dependent on the memory capacity allocated to a specific VM, which may result in a longer startup time of the VM compared to the default Xen. However, this effect is limited to its startup procedure.

6.1.2 Overcommitted System

To further analyze isolation, we run multiple VMs simultaneously as an *overcommitted* system [29] (i.e., the system is configured to run more VCPUs than the number of PCPUs). First, we run two victim-VMs and a PH-VM, and the workload in each victim-VM is SPEC CPU2000. When the three VMs run in: (1) the default Xen, we test the performance of the two victim-VMs (`Xen`); (2) the TSAC prototype, we also test the performance of the two victim-VMs (`TSAC-c` or `TSAC-m`). When the three VMs are running simultaneously in the prototype system, there are two CVGs, and each victim-VM and the PH-VM belong to a CVG. In addition, (3) with `xm vcpu-pin` in Xen, two victim-VMs are pinned to PCPU-sets [0 ,1] and PCPU-sets [2, 3] respectively, and the PH-VM is pinned to PCPU-sets [1, 2], and we also test the performance of victim-VMs (`vcpu-pin`). As the performance of the two victim-VMs is very similar, one of results is shown as Fig. 10.

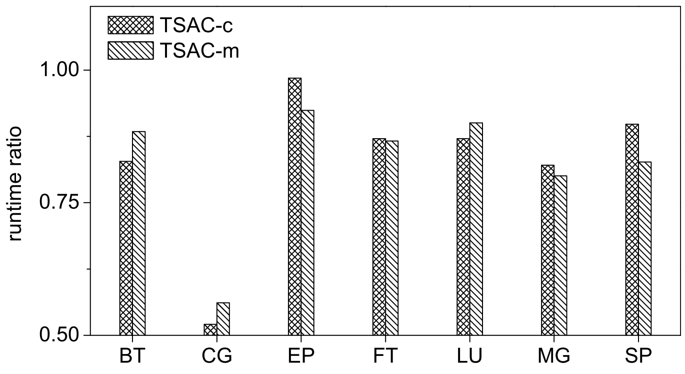


Fig. 9. The performance of a victim-VM, where workloads are the NAS parallel benchmarks, and *online ratio* = 100%.

Fig. 10 shows that TSAC can improve the performance of the victim-VM when it runs in an overcommitted system. The rate of the victim-VM is increased by about 11 percent by TSAC-c and 13 percent by TSAC-m on average, and by up to 45 percent by TSAC-c and 56 percent by TSAC-m for 181.mcf. This is because the number of running VCPUs is larger than the number of PCPUs in the system, and a VM may be blocked for a while due to lack of available PCPUs according to TSAC-c. Therefore, TSAC-m outperforms TSAC-c to some degree in an overcommitted system. Besides, the experiments show that TSAC outperforms `vcpu-pin`, as the static pinning method cannot guarantee ideal isolation as well as fairness at the same time in an overcommitted system, while the dynamic contention management method in TSAC can still work well.

To further investigate the scalability of TSAC, we run four victim-VMs and a PH-VM simultaneously, and each victim-VM and the PH-VM belong to a CVG. The workloads on each victim-VM are the NAS parallel benchmarks. We also test the performance of the four victim-VMs in `Xen`, `TSAC-c` and `TSAC-m`, and the runtime ratio of benchmarks is also adopted. These victim-VMs still have similar performance, and Fig. 11 shows one of results. For each victim-VM, its runtime could be reduced by about 14 percent by TSAC-m and 10 percent by TSAC-c on average, and by up to 28 percent by TSAC-m and 20 percent by TSAC-c for CG. As there is no number limitation of VMs in TSAC, it can be adopted when a variety of VMs run simultaneously in the system.

6.2 Evaluation with ED-VM

As demonstrated in Section 3, the load information leakage is also possible in a virtualized system with multi-core processors. In this section, we first analytically examine the enhanced isolation provided by TSAC. Then, we present experiments with the implemented prototype.

6.2.1 Analysis

As mentioned earlier, avoiding having two VMs share caches in a processor is the most thorough way to prevent cache-unfriendly operations. However, the performance of the virtualized system is also an important issue as well as

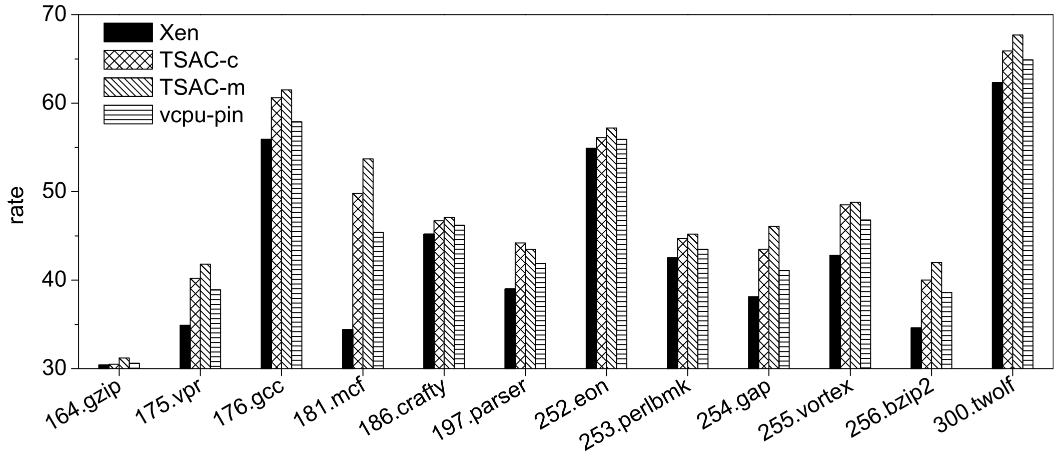


Fig. 10. The performance of a victim-VM, where workloads are SPEC CPU2000, and *online ratio* = 66.7%.

isolation. Therefore, we present TSAC for reinforcing the isolation without sacrificing the performance.

TSAC-c is used to enforce CPU isolation in the virtualized multi-core system. According to TSAC-c, two VCPUs from two VMs in a CVG will not be scheduled to run on the same PCPU-set within a period of time. A VCPU usually run on difference PCPUs for load balancing according to the scheduling manner in Xen, which is illustrated as the steep rise in the value of samples in Fig. 4. Therefore, as long as the interval is an appropriate length, the residual content of the former VCPU will be evicted by other VMs, and then the cache activity information detected by the later VCPU will belong to VCPUs in other VMs, rather than the former. Consequently, the load information of the former VM will not be leaked to the later VM. This guards against the load information leakage in the virtualized system.

Different from TSAC-c, TSAC-m is used to enforce memory isolation in the virtualized system with multi-core processors. With the page coloring technique, two VMs in a CVG use different groups of cache lines, although VCPUs from the two VMs run on the same PCPU-set. Therefore, for the ED-VM, it is impossible to detect the load information of the victim-VM through its own cache activity if they belong to the same CVG.

In summary, to avoid having two VMs in the same CVG share processor caches, TSAC-c guarantees that two VCPUs from the two VMs will not be scheduled to the same PCPU-set at the same time or in turn, and implements a coarse-grained enforced isolation, while TSAC-m guarantees that the two VMs will never share the same cache lines even if they run on the same processor, and implements a fine-grained enforced isolation.

6.2.2 Experiments

To further validate the enhanced isolation of TSAC, we empirically evaluated its effectiveness against cross-VM information leakage illustrated in Section 3. Specifically, an ED-VM and a victim-VM run simultaneously in the NWC mode in the virtualized system, and they belong to the same CVG. The 100 continuous samples are also tested on the ED-VM. There are two cases, one for no workload on the victim-VM, and the other for four copies of the SPEC CPU2000 benchmarks (i.e., 176.gcc or 256.bzip2) on the victim-VM. All tests are performed in the TSAC prototype.

When TSAC-c is adopted to enhance CPU isolation in the VMM, the result is shown as Figs. 12a and 12b. The graphs show that the majority of samples in the two cases have similar values with a steep rise in an interval. Compared to Fig. 4 in Section 3.3, the majority of samples here are not significantly different, which indicates that it is hard for the ED-VM to detect the load information of a VM in the same CVG.

When TSAC-m is adopted to enhance memory isolation in the VMM, the result is depicted as Figs. 12c and 12d. It shows that the characteristics of samples obtained in the two cases are very similar, and indicates that the enhanced memory isolation mechanism performs well in blocking cache-based information leakage.

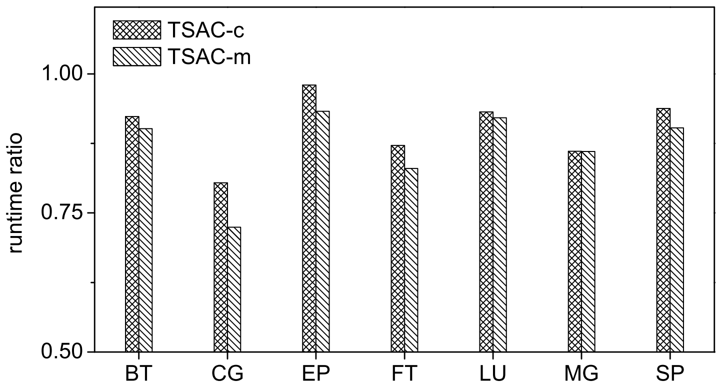


Fig. 11. The performance of a victim-VM, where workloads are the NAS parallel benchmarks, and *online ratio* = 40%.

7 Related Work

Native systems. In the multi-core system, cache interference is a well-known problem, and there are a lot of research efforts focused on its performance improvement. One method is software-based cache partitioning [25], [30], [31], [32], where the cache is partitioned by operating systems using page coloring for workloads, in order to isolate those workloads that are cache-unfriendly to each other. The other method is contention-aware CPU scheduling [33], [34], in which the scheduler tries to predict to what extent threads may hurt each other's performance, and determines whether threads should be clustered or spread across chips. Different from the non-virtualization system, the sematic

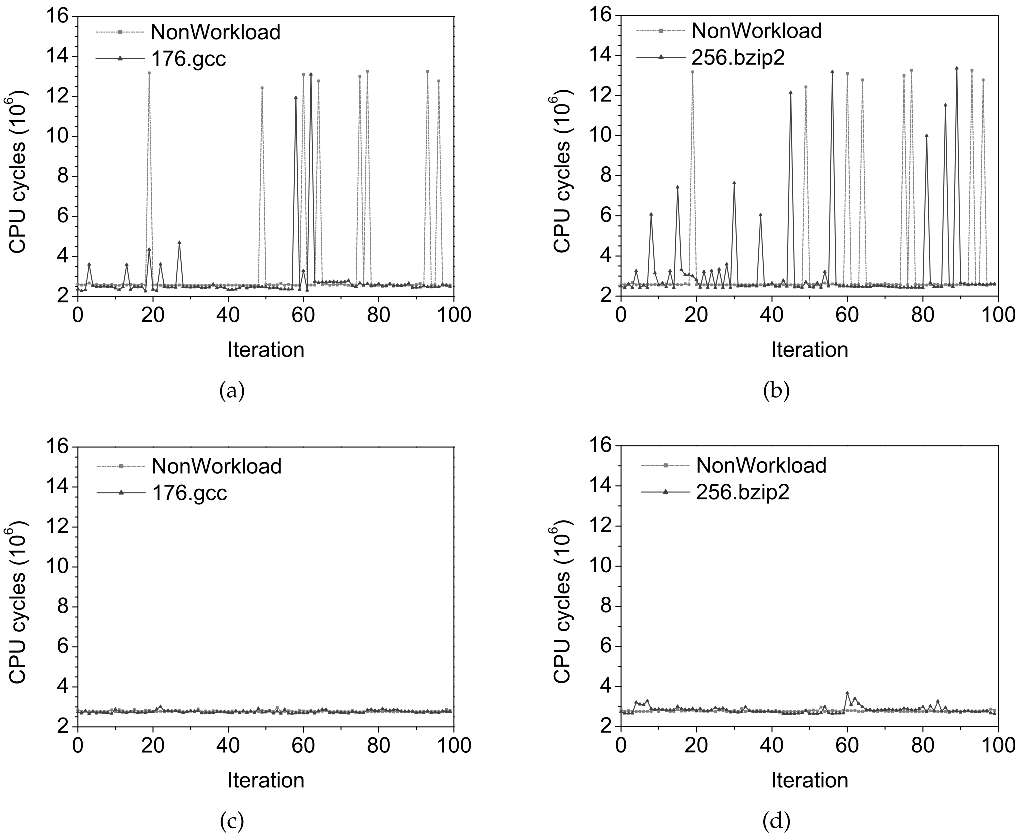


Fig. 12. Samples on the ED-VM: (a) 176.gcc with TSAC-c, (b) 256.bzip2 with TSAC-c, (c) 176.gcc with TSAC-m, (d) 256.bzip2 with TSAC-m.

gap [14] exists in the virtualized system, and the VMM lacks information of processes or threads in the guest operating systems. As a result, these methods cannot be applied directly to virtual machine systems. In this paper, we propose the TSAC framework to enforce isolation between VMs in a CVG. In a cloud system, a VM is usually dedicated to executing a specific application, so that system administrators could classify VMs to the different CVGs through related research efforts [23].

Enforcing isolation. There are some research efforts related to reinforce system isolation in the virtualizated environment. To guarantee the isolation of resource sharing between VMs, sHype [35] is proposed to control the information flow between VMs. The main components in sHype are an enforcement hook and an access control module in the hypervisor, which determine whether two VMs can coreside on a physical machine or two VMs can communicate with each other by sharing memory pages. sHype is focused on sharing inter-VM resources instead of lower-level physical resources such as cache, and its isolation rule is helpless to inhibit cache-based attacks. An analytical model is developed and implemented for sharing policies in the virtualized system [36].

Further, an enforcement method [37] is presented for performance isolation in the virtualized system, where XenMon measures resource consumption, and the SEDF-DC scheduler performs CPU allocation among VMs according to feedback from XenMon, and ShareGuard limits the total amount of resources consumed by VMs based on the specified limits. Differing from our work, the cache issue is not considered in this method, and the resource consumption is dependent on events such as VCPU migration at some control points in the VMM. Another related work is [38], which focuses on fine-grained isolation between VMs. In this work, resource allocation for satisfying various QoS and isolation SLA constraints in the cloud infrastructure is reduced to a constraint satisfaction problem. The resource allocation of a VM is static like the function `xm vcpu-pin`. Our previous work [39] also gives a static method, and a shared physical resource works for at most one VM in each CVG all the time as in the original Chinese Wall policy. Besides, processor and memory page are considered as subjects in the access control policy. In contrast, we adopt TSAC to dynamically prevent caches from being shared by adverse VMs, in order to maintain the efficiency of multiplexing hardware resources while improving the system isolation. Moreover, PCPU-sets and page colors are refined as subjects in the policy.

Attacks based on virtualization. Some kinds of attacks on VMs in a cloud are discussed in [15]. The authors argue that fundamental risks arise from sharing physical infrastructure between mutually distrustful users in the virtualized system, and believe that for security two adverse VMs from different users have to be populated on different physical machines. In this paper, for cache-unfriendly operations, we relax the restriction of avoiding co-residence of adverse VMs on a physical machine, and propose TSAC to eliminate the means of cache-unfriendly operations in the virtualized system, while keeping the performance.

Besides, in default virtualized systems, performance interference [18] is generated using a malicious VM, that is, when the malicious VM exhausts one type of hardware resources, it will bring performance interference to another type of resources. An access-driven side-channel attack is presented in [40], in which a malicious VM extracts fine-grained information from a victim VM co-running on the same machine. TSAC is helpful to defend against those kinds of attacks. In addition, HomeAlone [41] is a system that lets a cloud tenant verify its VMs' exclusive use of a physical machine in the cloud platform. In addition, virtualization may also be utilized to implement virtualization based malwares, and a typical work is SubVirt [42].

Enforcing security by leveraging virtualization. There are also some research efforts, in which virtualization is utilized to enforce the system security. Based on the property of isolation supported by virtualization, we can utilize the trusted VM to monitor the status of guest VMs. One kind is out-of-VM monitoring [43], in which the monitoring mechanism is implemented out of the monitored VMs. The other kind is in-VM monitoring [44], and the hardware virtualization technology is adopted to guarantee the security application, whereas the hardware virtualization technology is not needed in our previous work [45]. Virtualization could also be applied to enhance the isolation of web applications [46] and consumer electronics devices [47].

Performance optimization. As a new computing infrastructure, there are some research efforts related to the performance analysis and optimization of the cloud platform. The cache as a service (CaaS) [48] is presented as an optional service to typical infrastructure service offerings. That is, a large pool of memory are set a sided by the cloud provider, in order to be dynamically partitioned and allocated to standard infrastructure services as disk cache. A novel approximate analytical model [49] is presented for performance evaluation of cloud server farms, which is based on an approximate Markov chain model. DawningCloud [50] is designed and implemented to demonstrate that small-to-medium scale scientific communities can benefit from the economies of scale. As an important testbed for cloud computing research, Open Cirrus [51] offers a cloud stack, which consists of physical and virtual machines, and global services.

8 Conclusion

Virtualization is widely regarded as an important technology for cloud computing, and it can be adopted to customize a variety of VMs on demand. Isolation is considered as an important feature of virtualization, however, in this paper, we show that two kinds of isolation problems exist in the virtualized multi-core system. Consequently, we present TSAC for allocating resources in the VMM, in order to dynamically avoid having two VMs with the cache-unfriendly operation share physical resources. Enhanced isolation for two kinds of physical resources (PCPU-set and page color) is designed and implemented according to TSAC. Our evaluation of the implemented prototype based on Xen demonstrates its practicality and effectiveness. Actually, more contention on physical resources can be considered in the virtualized multi-core system, and enhanced isolation for those physical resources could be easily implemented according to TSAC.

References

[1] R. Buyya, J. Broberg, and A. Goscinski, Ed., *Cloud Computing: Principles and Paradigms*. New York, NY, USA: Wiley, Feb. 2011.

[2] P. H. Gum, "System/370 extended architecture: Facilities for virtual machines," *IBM J. Res. Develop.*, vol. 27, no. 6, pp. 530–544, 1983.

[3] J. E. Smith and R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes. New Yor, NY, USA: Elsevier, 2005.

[4] Xen. [Online]. Available: http://www.xen.org/, 2015.

[5] 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," in *Proc. ACM Symp. Operating Syst. Principles*, Oct. 2003, pp. 164–177.

[6] VMWare. [Online]. Available: http://www.vmware.com/, 2015.

[7] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. 5th Symp. Operating Syst. Des. Implementation*, Dec. 2002, pp. 181–194.

[8] KVM. [Online]. Available: http://www.linux-kvm.org/, 2015.

[9] Hyper-V. [Online]. Available: http://www.microsoft.com/hyper-v-server/, 2015.

[10] VirtualBox. [Online]. Available: http://www.virtualbox.org/, 2015.

[11] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proc. 11th Int. Symp. High Perform. Comput. Arch.*, 2005, pp. 340–351.

[12] F. J. Cazorla, P. M. W. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez, and M. Valero, "Predictable performance in SMT processors," in *Proc. 1st Conf. Comput. Frontiers*, 2004, pp. 433–443.

[13] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *Proc. 13th Int. Conf. Parallel Arch. Compilation Tech.*, 2004, pp. 111–122.

[14] P. M. Chen and B. D. Noble, "When virtual is better than real," in *Proc. 8th Workshop Hot Topics Oper. Syst.*, 2001, pp. 133–138.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 199–212.

[16] Credit. [Online]. Available: http://wiki.xensource.com/wiki/Credit_Scheduler, 2015.

[17] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, 2007.

[18] Q. Huang and P. C. Lee, "An experimental study of cascading performance interference in a virtualized environment," *Perform. Eval. Rev.*, vol. 40, no. 4, pp. 43–52, 2013.

[19] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," in *Proc. 16th USENIX Security Symp.*, 2007, pp. 1–18.

[20] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures," *J. Cryptol.*, vol. 23, no. 1, pp. 37–71, 2010.

[21] SPEC, Standard performance evaluation corporation. [Online]. Available: http://www.spec.org, 2015.

[22] D. F. C. Brewer and M. J. Nash, "The Chinese wall security policy," in *Proc. IEEE Symp. Security Privacy*, 1989, pp. 206–214.

[23] G. Daci and M. Tartari, "A comparative review of contention-aware scheduling algorithms to avoid contention in multicore systems," in *Proc. 3rd Int. Conf. Trends Inform., Telecommun. Comput.*, 2013, pp. 99–106.

[24] R. E. Kessler and M. D. Hill, "Page placement algorithms for large real-indexed caches," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 338–359, 1992.

[25] S. Cho and L. Jin, "Managing distributed, shared L2 caches through OS-level page allocation," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarch.*, 2006, pp. 455–468.

[26] D. Tam, R. Azimi, L. Soares, and M. Stumm, "Managing shared L2 caches on multicore systems in software," in *Proc. Workshop Interaction between Oper. Syst. Comput. Arch.*, 2007, http://www.ideal.ece.ufl.edu/workshops/wiosca07/Paper4.pdf

[27] C. Weng, Z. Wang, M. Li, and X. Lu, "The hybrid scheduling framework for virtual machine systems," in *Proc. Int. Conf. Virtual Execution Environ.*, 2009, pp. 111–120.

[28] NAS Parallel Benchmarks. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html, 2015.

[29] P. M. Wells, K. Chakraborty, and G. S. Sohi, "Hardware support for spin management in overcommitted virtual machines," in *Proc. 15th Int. Conf. Parallel Arch. Compilation Tech.*, 2006, pp. 124–133.

[30] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *Proc. 14th Int. Symp. High Perform. Comput. Arch.*, 2008, pp. 367–378.

[31] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm, “Rapidmrc: Approximating L2 miss rate curves on commodity systems for online optimizations,” in *Proc. 14th Int. Conf. Arch. Support Programm. Lang. Oper. Syst.*, 2009, pp. 121–132.

[32] X. Zhang, S. Dwarkadas, and K. Shen, “Towards practical page coloring-based multicore cache management,” in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 89–102.

[33] S. Blagodurov, S. Zhuravlev, and A. Fedorova, “Contention-aware scheduling on multicore systems,” *ACM Trans. Comput. Syst.*, vol. 28, no. 4, pp. 8–45, 2010.

[34] A. Fedorova, M. Seltzer, and M. D. Smith, “Improving performance isolation on chip multiprocessors via an operating system scheduler,” in *Proc. 16th Int. Conf. Parallel Arch. Compilation Tech.*, 2007, pp. 25–38.

[35] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. Doorn, “Building a MAC-based security architecture for the Xen open-source hypervisor,” in *Proc. 21st Annu. Comput. Security Appl. Conf.*, 2005, pp. 276–285.

[36] S. Rueda, H. Vijayakumar, and T. Jaeger, “Analysis of virtual machine system policies,” in *Proc. 14th Symp. Access Control Models Tech.*, 2009, pp. 227–236.

[37] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing performance isolation across virtual machines in Xen,” in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*, 2006, pp. 342–362.

[38] H. Raj, R. Nathuji, A. Singh, and P. England, “Resource management for isolation enhanced cloud services,” in *Proc. Cloud Comput. Security Workshop, Conjunction with ACM CCS*, pp. 77–84, 2009.

[39] G. Wang, M. Li, and C. Weng, “Chinese wall isolation mechanism and its implementation on VMM,” in *Proc. 3rd Int. DMTF Academic Alliance Workshop Syst. Virtualization Manage.: Standards Cloud*, 2009, pp. 13–18.

[40] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. “Cross-VM side channels and their use to extract private keys,” in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 305–316.

[41] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, “Homealone: Co-residency detection in the cloud via side-channel analysis,” in *Proc. IEEE Symp. Security Privacy*, 2011, pp. 313–328.

[42] S. T. King, P. M. Chen, Y. M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, “Subvirt: Implementing malware with virtual machines,” in *Proc. IEEE Symp. Security Privacy*, 2006, pp. 314–327.

[43] B. Payne, M. Carbone, M. Sharif, and W. Lee, “Lares: An architecture for secure active monitoring using virtualization,” in *Proc. IEEE Symp. Security Privacy*, 2008, pp. 233–247.

[44] M. Sharif, W. Lee, W. Cui, and A. Lanzi, “Secure in-VM monitoring using hardware virtualization,” in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 477–487.

[45] Q. Liu, C. Weng, M. Li, and Y. Luo, “An in-VM measuring framework for increasing virtual machine security in clouds,” *IEEE Security Privacy*, vol. 8, no. 6, pp. 56–62, Nov. 2010.

[46] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy, “A safety-oriented platform for web applications,” in *Proc. IEEE Symp. Security Privacy*, 2006, pp. 350–364.

[47] G. Heiser, “Hypervisors for consumer electronics,” in *Proc. 6th IEEE Consumer Commun. Netw. Conf.*, 2009, pp. 614–618.

[48] H. Han, Y. Lee, W. Shin, H. Jung, H. Yeom, and A. Y. Zomaya, “Cashing in on the cache in the cloud,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1387–1399, Aug. 2011.

[49] H. Khazaei, J. Misic, and V. B. Misic, “Performance analysis of cloud computing centers using m/g/m/m + r queueing systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 936–943, May 2012.

[50] L. Wang, J. Zhan, W. Shi, and Y. Liang, “In cloud, can scientific communities benefit from the economies of scale?” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 2, pp. 296–303, Feb. 2012.

[51] R. Campbell, I. Gupta, M. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Lee, M. Lyons, D. Milojicic, D. OHallaron, and Y. Soh, “Open cirrus cloud computing testbed: Federated data centers for open source systems and services research,” in *Proc. Conf. Hot Topics Cloud Comput.*, 2009, pp. 1–5.

Chuliang Weng received the PhD degree from Shanghai Jiao Tong University in 2004. He is a principal researcher at Huawei Shannon Lab. In 2013, he joined Huawei, and as a technical director, started a research team focusing on building storage systems for virtualized datacenters. Before coming to Huawei, he was an associate professor of computer science at Shanghai Jiao Tong University. From 2011 to 2012, he was a visiting scientist with the Department of Computer Science at Columbia University in the city of New York. His research interests include parallel and distributed systems, operating systems and virtualization, and storage systems. He is a member of the IEEE, ACM and CCF.

Jianfeng Zhan received the PhD degree in computer engineering from the Chinese Academy of Sciences, Beijing, China, in 2002. He is currently a professor of computer science with the State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences). His current research interests include benchmarks and workload characterization, operating systems, big data systems, and parallel and distributed systems. He has authored more than 40 peer-reviewed journal and conference papers in these areas. He was a recipient of the Second-class Chinese National Technology Promotion Prize in 2006, and the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005. He is a member of the IEEE.

Yuan Luo received the PhD degree in probability and mathematical statistics from Nankai University, China, in 1999. From July 1999 to April 2001, he held a postdoctoral position in the Institute of Systems Science, Chinese Academy of Sciences, China. From May 2001 to April 2003, he had a postdoctoral position in the Institute for Experimental Mathematics, University of Duisburg-Essen, Germany. He is currently a full professor of computer science with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His research interests include information theory, coding theory, and computer security. He is a member of the IEEE.