Optimizing Crash Dump in Virtualized Environments

Yijian Huang
Parallel Processing Institute
Fudan University
huangyj@fudan.edu.cn

Haibo Chen
Parallel Processing Institute
Fudan University
hbchen@fudan.edu.cn

Binyu Zang
Parallel Processing Institute
Fudan University
byzang@fudan.edu.cn

Abstract

Crash dump, or core dump is the typical way to save memory image on system crash for future offline debugging and analysis. However, for typical server machines with likely abundant memory, the time of core dump can significantly increase the mean time to repair (MTTR) by delaying the reboot-based recovery, while not dumping the failure context for analysis would risk recurring crashes on the same problems.

In this paper, we propose several optimization techniques for core dump in virtualized environments, in order to shorten the MTTR of consolidated virtual machines during crashes. First, we parallelize the process of crash dump and the process of rebooting the crashed VM, by dynamically reclaiming and allocating memory between the crashed VM and the newly spawned VM. Second, we use the virtual machine management layer to introspect the critical data structures of the crashed VM to filter out the dump of unused memory. Finally, we implement disk I/O rate control between core dump and the newly spawned VM according to user-tuned rate control policy to balance the time of crash dump and quality of services in the recovery VM.

We have implemented a working prototype, Vicover, that optimizes core dump on system crash of a virtual machine in Xen, to minimize the MTTR of core dump and recovery as a whole. In our experiment on a virtualized TPC-W server, Vicover shortens the downtime caused by crash dump by around 5X.

Categories and Subject Descriptors K.6.3 [*Management of Computing and Information Systems*]: Software Management—Software maintenance; D.4.5 [*Operating Systems*]: Reliability

General Terms Management, Performance, Reliability

Keywords Core Dump, Parallel Core Dump, Virtual Machines

1. Introduction

Reliability has been one of the major concerns of modern computer systems, which may be frequently disturbed by software and hardware errors, unsafe software/hardware interactions, as well as operation errors. While restarting the frozen or crashed system is the usual way to recover from failures, it is demanding to diagnose the root cause of the failure for future software and hardware fixes.

VEE'10, March 17–19, 2010, Pittsburgh, Pennsylvania, USA.

Crash dump, or core dump [7], is a typical way to save the crashed or hang context to persistent storage for future offline debugging and analysis [5, 6] before restarting the system. However, for traditional server machines with likely abundant memory, it is usually time-consuming to save the failure context (usually including the whole memory), which delays the recovery of the system and increases the mean time to repair (MTTR). For instance, core-dumping 32 GB memory into a commodity SCSI disk with 70 MB/sec write rate could take more than 400 seconds. Hence, some administrators may choose to directly restart the system without a crash dump, risking recurring crashes on the same problems.

In this paper, we analyze the characteristics of crash dump and propose several techniques to optimize the process of crash dump of virtual machines in consolidated virtualized environments. Unlike that in a native environment, the crash or hang of a virtual machine will not cause an end of the system, as the hypervisor and VM management tools remain fully operational. This opens opportunities to optimize crash dump to minimize the downtime caused by the system crash.

We propose to parallelize the core dump of the crashed virtual machine and the booting of another *recovery virtual machine* to continue the same application services. In order to retain the persistent states, the recovery VM uses the same file system as the crashed VM. As a result, persistent states updated by the crashed VM remain accessible by the recovery VM after the recovery VM is started. The sharing is safe, because the two VMs do not access the file system simultaneously: the recovery VM accesses the file system only after the other VM crashes and stops the access to the file system. Here, we assume that the crash of a virtual machine does not cause irreversible system damages and can be repaired by file system consistency check (e.g., fsck) when booting the recovery VM.

When a VM crashes, its CPU and I/O resources are immediately released. However, due to the long-term memory reservation by the core dump tool, using the memory owned by the crashed VM for recovery is not allowed until the core dump ends. Hence, we propose to concurrently reclaim core-dumped memory and allocate it to the recovery VM. To this end, the pseudo-physical memory of a VM is divided into several chunks of the same size. If the VM crashes, its pseudo-physical memory is core-dumped to disk chunk by chunk, instead of all at once. For each chunk, once it is core-dumped, it is immediately reallocated to another VM which boots the system from the shared storage to recover application services. As core dump continues, the hypervisor eventually releases all memory owned by the crashed VM, and the recovery VM gradually reaches its presumed memory capacity. The recovery VM not only gains memory from the crashed VM as early as possible, but also utilizes the CPU resource which core dump, as an I/O-intensive process, does not fully utilize.

To minimize the time of core dump, we also implement a selective dump strategy that only dumps memory that is in use by

the crashed VM, instead of dumping the whole memory of a virtual machine. To identify which pages are in use, we introspect the crashed VM, extract key data structures of the crashed guest OS and recognize free memory pages not used on system crash, so that only pages used by the guest OS or applications of the VM at the moment of crash are core-dumped. The trustworthiness depends on that the introspected part of memory is not corrupted by system crash, which is a trade-off between the speed and correctness of core dump.

Both the core dump VM and the recovery VM consume the I/O bandwidth. To balance the time of crash dump and quality of services in the recovery VM, we implement a disk I/O rate control mechanism to balance the I/O bandwidth between the two VMs. The bandwidth allocation policy is tuned by user as a trade-off between the speed of reclaiming memory by core dump and the disk I/O bandwidth for recovery. For a none I/O-intensive workload of booting the recovery VM, core dump should be given higher priority of disk I/O. Otherwise, if both core dump and booting the recovery VM are I/O-intensive, the policy should be tuned by user to minimize downtime.

We have implemented a working prototype, called Vicover, based on Xen and Linux. Vicover has a utility program running at the VM management layer. It automatically detects system crash of a VM, and initiates core dump of the VM. It also concurrently boots the recovery VM that shares the same file system with the crashed VM.

Experimental results of core dump and recovery from system crash of a virtualized TPC-W server on the Dell PowerEdge R900 machine show that the downtime is reduced by 5X with concurrent core dump and recovery from 103 seconds to 21 seconds, or is shortened significantly if there are many free pages on crash by selective dump with Vicover. Tuning the allocation policy toward higher I/O priority for I/O-intensive (but not memory-critical) recovery over core dump reduces the downtime of concurrent core dump and recovery by around 50%.

The rest of the paper is organized as follows. Some background information on crash dump is presented in next section. We then describe the overall architecture and optimizations of core dump at the VM management layer in section 3 and detail the implementation issues of our prototype in section 4. Section 5 defines the key metric of evaluation - downtime, introduces our testing framework for evaluation and specifies the testing environment. Section 6 evaluates the effectiveness to shorten the downtime of core dump and recovery by Vicover. Section 7 compares our work with previous efforts to improve recovery from system crash. Finally, we conclude the paper in section 8.

2. Background

To manage various hardware resources and provide rich abstractions of them to applications, modern operating systems have evolved with complex, flexible and rich functionalities, which spoil the stability and result in recurring system crashes. A system crash is a kernel-level crash caused by many factors such as memory corruption and bad drivers [3]. Although OS crashes are rare [3], they are frustrating to users since recovery from them requires rebooting of the whole system.

In order to prevent system crashes from recurring in the future, it is worthwhile saving the states of the crashed system for future offline debugging, which is called "core dump". Core dump is an I/O-intensive process to write system states into persistent storages. On system crashes, CPU and I/O resources are immediately released by the crashed system. After the core dump begins, the memory of the crashed system is fully reserved by the core dump tool, until all the memory is dumped into persistent storage.

Core dump tools are available for commodity OSes as well as for virtualized environments. Microsoft Windows can be set up to save a 64 KB minidump of basic states, a kernel dump of memory in the Windows kernel, or a full dump of the whole RAM on system crash. A debugger tool like WinDbg can be used to analyze the failure data saved in the core dump image.

In Linux, Kdump [7] provides a reliable core dump mechanism. Kdump [7] improves Kexec by fast and automatically booting into a new kernel on system crashes and executing core dump in the context of the new kernel. Kexec is a boot loader which fast boots into a new kernel from the old one without clearing the memory of the old kernel. It pre-loads a new kernel and associated data into a reserved area of memory by the *kexec_load()* system call. The reserved area is configured not to be used by the old kernel. To quickly load the new kernel, Kexec boots into the new kernel without hardware reset, firmware operations or the bootloader stage. The new kernel is directly loaded from the reserved area so that the memory of the old kernel is not cleared. Kdump modifies Kexec to automatically boot into a new kernel on kernel panic. After the new kernel boots, the memory of the old one is accessible through special UNIX device files in the context of the new kernel and can be core-dumped into persistent storage. The output core dump image is analyzable by debugging or analysis tools (e.g. gdb [4] and crash analysis utility [6]).

In virtualized environments, core dump tools save the memory of a VM to persistent storages, whose reliability is guaranteed by the isolation mechanism from the virtualization layer. These tools are available in commodity virtualization products [2, 8]. For instance, to dump the memory of a virtual machine in Xen, the core dump tool of Xen maps the machine frames owned by the crashed VM through the hypervisor, and writes them to an ELF image file in the local file system. A feature of core dump in virtualized environments is live core dump, by which the memory of a virtual machine can be core-dumped while the virtual machine is running.

3. Optimizing Core Dump with Virtualization

In this section, we present an overview on the design of optimizing core dump on the crash of a VM at the VM management layer, to achieve fast core dump and recovery. First, we illustrate the overall architecture. Then we present a detailed discussion on the design concerns of the key components.

3.1 System Architecture

As the VM management layer is operational despite the system crash of a VM, we put complex optimizations of core dump to run at this layer for fast core dump and recovery.

Figure 1 shows the architecture of a virtualized server with the optimizations of core dump applied. The hypervisor runs on bare hardware. It multiplexes hardware for all the VMs running on the hypervisor.

Vicover recovers from the crash of a VM by booting another VM. The crashed VM is shown as the square with solid boundary; the recovery VM that boots the same system and applications is shown as the square with dashed boundary. The crashed VM and the recovery VM share the same file system to retain persistent states. The sharing is safe because the latter VM accesses files only after the former crashes.

Each one of these VMs is allocated memory resource by the hypervisor and forms their own pseudo-physical memory space, represented by the rectangles labeled "Pseudo-Physical Memory Space" in the VMs.

The virtual machine management tools, shown as the rectangle labeled "Management Tools" in the management VM, interacts with the hypervisor to manage VMs.

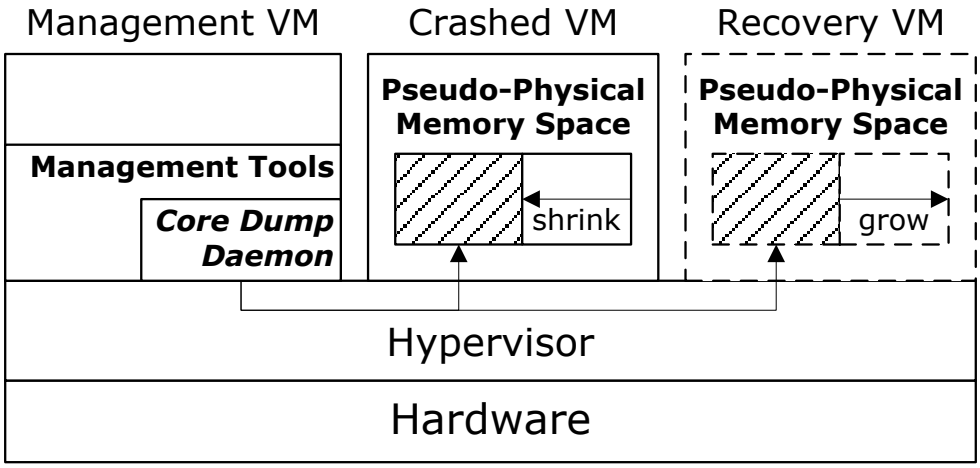

Figure 1. System architecture of Vicover: the core dump daemon is located in the management VM, which creates the recovery VM upon detecting the crash of a VM; the daemon controls the reallocation of resource from the crash VM to the recovery VM.

Vicover consists of a core dump daemon and a hypercall. The core dump daemon is a utility program as part of the VM management tools, shown as the smaller rectangle inside the rectangle of "Management tools". The daemon detects crash and then initiates core dump and recovery automatically. It optimizes core dump by modifying the default core dump tool. In addition, the hypercall is a slight extension to the hypervisor by which the core dump daemon interacts with the hypervisor to concurrently reclaim the memory of the crashed VM and increase the memory of the recovery VM, as is shown by the two arrow lines passing through the hypervisor.

The VM management layer monitors the state of a VM. When a VM crashes, it is notified. Then, the core dump daemon of Vicover is in turn notified of the crash by the VM management layer. It automatically initiates the core dump of the crashed VM and boots the recovery VM. Optimizations of core dump are applied to shorten the time of core dump and recovery, which are explained as follows.

3.2 Concurrent Core Dump and Recovery

Vicover realizes concurrent core dump and recovery, so that hardware resources are better utilized and the overall downtime it takes to finish core dump and recovery is shortened.

Long-term reservation of the entire memory of the crashed VM prevents part of it from being used for recovery earlier, which is the main bottleneck to realize concurrent core dump and recovery. To resolve it, Vicover concurrently reclaims the memory of the crashed VM, and allocates it to the recovery VM. Hence, the pseudo-physical memory owned by the crashed VM shrinks, and that owned by the recovery VM grows concurrently.

To realize it, we divide the memory of the crashed VM into fixed-sized chunks. Once a chunk of memory is core-dumped and reclaimed, it is allocated to the recovery VM. Therefore, memory of the crashed VM is gradually reallocated to the recovery VM chunk by chunk.

Ideally, if the rate of core dump is constant, the rate of memory reallocation from the crashed VM to the recovery VM is also constant. It is expected that the amount of pseudo-physical memory owned by the crashed VM shrinks linearly to zero, and the memory owned by the recovery VM grows linearly to the original memory size of the crashed VM. The linear shrinking and growing are indicated by arrows labeled "shrink" and "grow" in Figure 1. However, there are several issues that prevent such ideal linearity from being achieved in practice:

- *Cache during Core Dump* The memory written by the core dump tool is by default buffered by cache in the operating system, to improve I/O throughput. However, caches of the operating system may defer flushing memory to disk. The caching mechanisms affect the constant rate of reclaiming memory, during concurrent core dump and recovery. To solve this problem, the modified core dump tool of Vicover caches memory read from the crashed VM in its own application-level buffer, and flushes the memory to disk in fixed-sized batches. The core dump tool of Vicover achieves more stable rate of memory flushing and reallocation to the recovery VM than the default one.
- *Selection of Chunk Size* Chunk size is the granularity of memory reallocation from the crashed VM to the recovery VM. A small chunk size incurs more CPU overhead due to more frequent interaction with the hypervisor for memory reallocation. On the other hand, a large chunk size reduces the degree of concurrency between reclaiming memory by core dump and allocating it to the recovery VM and lowers down memory utilization.
- *Minimum Memory Requirement of the Recovery VM* During concurrent core dump and recovery, the pseudo-physical memory owned by the recovery VM grows to the original size of pseudo-physical memory in the crashed VM. Yet, the recovery VM does not start to boot once core dump begins, until a minimum amount of memory required to hold the OS kernel and basic services is gained from the crashed VM.

  We define a threshold of memory to represent the minimum requirement, depending on the guest OS and services to run in the recovery VM. The recovery VM starts once enough memory is gained from core dump by concurrent core dump and recovery.

  Inside the recovery VM, it is configured to make heavyweight services that consume much memory start at the end of the whole boot process. This configuration prevents the recovery VM from greedily consuming too much memory at early stages of the boot process. Thus more boot time is given for core dump to reclaim memory from the crashed VM, before the heavyweight services start in the recovery VM, in the hope that the moment heavyweight services start in the recovery VM, there is already adequate memory reallocated from the crashed VM.

3.3 Selective Dump

Instead of dumping the whole memory of the crashed VM, selective dump only core-dumps the part of memory useful for the specific debugging purpose of the user, saving both time and disk space for core dump. The selection relies on user knowledge about useful selection heuristics in the system to be debugged.

For instance, the VMM can introspect the crashed VM to extract specific data structures indicating the part of memory worth core-dumping. The trustworthiness of introspection relies on that the introspected memory in the crashed VM is not likely to be corrupted during the crash. At the VM management layer, implementing introspection of the crashed VM involves mapping its memory in the virtual address space of the core dump tool, which will be detailed in the next section.

As an example of selective dump, Vicover introspects the crashed VM to extract the page descriptor array of the Linux operating system, access the reference count field of each descriptor, and skip not-referenced free pages during core dump. Since Linux is open-sourced, digging the page descriptor array out of the raw memory data is easy with understanding of its structure by reading the source code in advance. Vicover traverses the array to collect free pages not used at the moment of a system crash. During core dump, these free pages are ignored. The digging implementation is OS-version-dependent, as different versions may vary in the structure of the page descriptor array. Since the page descriptor array counts for less than 1.5% of total memory in the VM, we believe it is not likely to be corrupted.

3.4 Disk I/O Rate Control

Vicover allocates disk I/O bandwidth between the core dump process in the management VM and the recovery VM according to user-defined policies. Core dump is essentially I/O-intensive. If recovery is not I/O-intensive, core dump can be given higher I/O priority. Otherwise, the I/O allocation policy should be tuned by user, to minimize downtime.

Vicover can utilize built-in I/O QoS solutions in commodity virtualization products to balance the I/O between concurrent core dump and recovery. Otherwise, a third-party I/O scheduling tool is applied. The detailed implementation of I/O scheduling for I/O rate control depends on the solution of I/O virtualization, which is described in the next section with Xen as an example.

4. Implementation

We have implemented a working prototype, Vicover, based on the Xen hypervisor with XenLinux as the guest operating system. It consists of 698 lines of code to implement the optimization of core dump at the VM management layer, and 136 lines of code to add a hypercall for Vicover in Xen.

This section describes the implementation of Vicover. First, the architecture of Xen, its VM management tool Xend, including its default core dump tool, are introduced. Second, the way to detect system crash of a VM and initiate core dump automatically by Vicover is described. Next, we explain in detail the implementation of optimizing core dump one by one.

4.1 Core Dump in Xen

The Xen hypervisor runs on bare hardware and multiplexes hardware resources for VMs. There are two kinds of virtual machines (or "Domains", in Xen's terminology) - Domain0 and DomainU. DomainUs are VMs that run applications to provide services, and thus consume most hardware resources. The single Domain0 is the privileged management VM. Domain0 accesses physical devices on behalf of DomainUs to handle I/O requests from DomainUs, according to the front-backend driver model.

The VM management tool of Xen is Xend. It runs as an application in Domain0. Xend can be used to monitor status of VMs, such as the crash event, and to allocate hardware resources among them.

Typically, on detecting a system crash of a VM by Xend, the administrator can invoke the core dump tool, which is a utility program of Xend, to core-dump the crashed VM. The invocation can either be done by command line or by a programming API.

The core dump tool requests the hypervisor to setup a mapping to the pseudo-physical memory pages of the crashed VM from its own process address space. It then writes the memory to an ELF image file in persistent storage. After core dump completes, the administrator destroys the crashed VM and starts the recovery VM with the memory reclaimed from the crashed VM.

The core dump daemon of Vicover automates the above process to initiate core dump and recovery on crash. To share the file system between the crashed VM and the recovery VM, the function that checks exclusive access to the virtual disk image file in Xend is modified to allow sharing.

4.2 Initiation of Core Dump

In DomainU, when a fatal error in the guest OS or one device driver occurs, the kernel panics. The panic handler in the kernel invokes a hypercall to Xen so that the Xen hypervisor intercepts the panic. The hypervisor updates the status of the DomainU as crashed. To forward the crash event to Domain0, the hypervisor fires a virtual interrupt request (VIRQ) to Domain0 through an event channel.

Domain0, which is the listener of the event channel, notices the crash event through the VIRQ, and then notifies Xend. Xend in turn fires an application-level XenStore "watch event" to any registered applications who are interested in the crash event. The core dump daemon of Vicover is such a registered application. It registers with Xend to listen for the watch event of system crash by a watcher API function. On detection of the watch event, it invokes the main routine of the core dump tool in Xend to core-dump the crashed VM. The core dump tool is a modified one for optimized core dump. Next, it boots the recovery VM with the memory reclaimed by core dump from the crashed one. If concurrent core dump and recovery is enabled, the recovery VM is started once the reclaimed memory satisfies the minimum requirement, while core dump is still in progress; otherwise, it is started after core dump completes.

Vicover is a utility program (*xcutils*) of Xend tools. It runs as a background process in Domain0 and is started by the command line in the console. It applies the specified optimizations of core dump according to the command line parameters, including concurrent core dump and recovery, selective dump and disk I/O rate control.

4.3 Concurrent Core Dump and Recovery

The main effort to realize concurrent core dump and recovery is to break the full reservation of the crashed VM's memory by the core dump tool, and have it reallocated to the recovery VM as early as possible. The memory is saved by core dump, and reallocated from the crashed VM to the recovery VM chunk by chunk. Figure 2 shows the pseudo-code to implement concurrent core dump and recovery by modifying the default core dump tool of Xend.

There are 9 steps in Figure 2, which are explained in the following:

1. At Line 1, the core dump tool gets hypervisor-level references to all pseudo-physical memory pages of the crashed VM through a hypercall.
2. At Line 2, the core dump tool invokes a hypercall to relinquish all the hypervisor-level references held by the crashed VM to its pseudo-physical memory pages. To relinquish them, the hypervisor traverses the list of memory pages owned by the

```
1.    Get references to all pseudo-physical pages of the crashed VM.
2.    Make the crashed VM relinquish all its references to its
      pseudo-physical pages
3.    for each page of the crashed VM
      {
            /* Core-dump the current page */
4.          write_to_disk(page)

            /* Release the core dump tool's reference to page, so that it
            is reclaimed by the hypervisor */
5.          put_page(page)

            /* If enough memory has been released */
6.          if (the recovery VM has not started &&
                  released pages >= minimum threshold)
            {
7.                Start the recovery VM.
            }
            /* Every chunk of memory is released, add it into the
            recovery VM */
8.          else if (the recovery VM is running &&
                  another memory chunk is core-dumped and
                  reclaimed)
            {
9.                Add the chunk of memory into the recovery VM.
            }
      }
```

Figure 2. Pseudo-code to implement concurrent core dump and recovery

crashed VM, and drops all the references to each of these pages from the crashed VM.

Line 1 and Line 2 together assure that after Line 2, the core dump tool becomes the only referrer to any memory page of the crashed VM. It enables the core dump tool to have accurate control over when to release a page and to reallocate it to the recovery VM, and also prevents the memory from being accidentally released by the hypervisor in the background.

3. Line 3 begins the iteration over each page of the crashed VM.
4. At Line 4, the core dump tool writes the current page to disk. Due to the Page Cache mechanism of Linux, it is possible that some pages are buffered by the management VM in which the core dump tool runs, and the constant speed of memory reclaiming is affected. In order to achieve linearity of reclaiming memory from the crashed VM, we modified the default caching behavior, so that every batch of memory is core-dumped, it is committed to disk by invoking fsync() over the core dump image file.
5. At Line 5, the core dump tool finally releases the last reference to a page of the crashed VM, after it is core-dumped at Line 4. As a result, the reference count in the corresponding hypervisor-level machine frame descriptor for that page drops to zero. Consequently, the hypervisor reclaims this page from the crashed VM. Then, the page is free to be allocated to the recovery VM.
6. Next, if the minimum memory required has been reclaimed since the 1st iteration, the recovery VM begins to boot at Line 7. After Line 7, core dump and the recovery VM run concurrently. The optimization of concurrent core dump and recovery begins. The default threshold of minimum memory is 128 MB.
7. At some iterations of Line 8, if another chunk of memory is core-dumped and reclaimed, it is allocated to the recovery VM at Line 9. The memory size parameter in the configuration of the recovery VM is set to be larger by the chunk size. The actual allocation of this chunk of memory to the recovery VM will be done by the hypervisor as long as the current memory size is smaller than the maximum allowed. The chunk size is configurable, and is 128 MB by default.

As the main logic of concurrent core dump and recovery, the for loop from Line 3 to Line 9 repeats, until all the memory of the crashed VM is core-dumped and reallocated to the recovery VM. Then, core dump ends; the recovery VM continues to boot with full memory reclaimed from the crashed one, or fortunately it may have already finished booting before core dump ends, depending on the specific behavior of the boot process.

We disable ballooning to prevent the recovery VM from grabbing the free memory of Domain0, in which the VM management tool Xend and Vicover run. Yet, our concurrent memory reclaiming and allocation is inspired by ballooning, in which case a VM reserves its free memory and releases it to the hypervisor by a balloon driver. Ballooning enables dynamic memory reallocation between two running VMs, while Vicover enables that between core dump and the recovery VM.

With regard to Line 1, 2 and 5, we added a dedicated hypercall to the hypervisor for Vicover to interact directly with the hypervisor. The hypercall involves acquiring or releasing references to memory from the core dump tool or from the crashed VM.

4.4 Selective Core Dump

Vicover introspects the crashed VM, extracts the page descriptor array "mem_map" of the guest Linux, identifies free pages indicated by the array, and finally ignores them during core dump. As a result, only pages used by the VM at the moment of crash are core-dumped, which shortens the latency of core dump.

The trustworthiness of this optimization relies on the integrity of the page descriptor array after the system crash of the VM. In our version of Linux, the size of each page descriptor of a 4096-byte page is 56 bytes. So the memory used by the page descriptor array is less than 1.5% of the total. We believe it to be rarely possible that the array is corrupted by the crash.

Vicover follows these steps to introspect the crashed VM at the VM management layer, in order the extract the page descriptor array of the guest operating system:

1. Obtain the virtual address of the page descriptor array in the virtual address space of the crashed VM. It is OS or application-dependent how a data structure can be located in the virtual address space. For our purpose, the symbol info of the page descriptor array in Linux is included in the kernel image file at compile time. We export the virtual address of the page descriptor array out of the kernel image with the "nm" tool at the VM management layer.
2. Convert the virtual address in the space of the guest system to the pseudo-physical address of the crashed VM. This depends on how OS maps the virtual address to the pseudo-physical one. Linux linearly maps data in the kernel space to physical memory. Therefore, the pseudo-physical address of the page descriptor array equals the virtual address obtained in the previous step minus the linear mapping offset.
3. Convert the pseudo-physical address to the machine frame address of the physical host machine. This is virtualization-solution-dependent. As a key data structure of memory virtualization, Xen maintains a pseudo-to-machine ("p2m") table that maps the contiguous pseudo-physical memory of a VM to the probably non-contiguous machine memory in the physical host. In Xend, the p2m table of a DomainU can be accessed in the context of Domain0. Vicover takes advantage of this access

to convert the pseudo-physical address to the machine frame address by the p2m table.

4. Set up mapping to machine frames of the page descriptor array in the core dump process. The running process of Vicover invokes a hypercall to modify its page tables, in order to set up mapping to the machine frames of the page descriptor array. After that, Vicover is able to access the page descriptors in its own virtual address space.

The modified core dump tool of Vicover maps the page descriptor array page by page in its own virtual address. For each mapped page, all the page descriptors stored in that page are accessed with the same mapping. For a page descriptor that spans two pseudo-physical pages, two mappings are setup to access and concatenate the 1st and 2nd halves of the page descriptor.

If the "_count" field of a page descriptor in the guest Linux is zero, the corresponding pseudo-physical page is not used by the guest operating system or applications, and is thus free at the moment of crash. Vicover maintains a bitmap to indicate if each page of the crashed VM is free or not. During core dump, free pages indicated by the bitmap are ignored.

4.5 Disk I/O Rate Control

Disk I/O rate control between concurrent core dump and recovery is realized by I/O scheduling over the core dump process in Domain0 and the virtual disk I/O of the recovery VM, according to the allocation policy tuned by user.

The open-source version of Xen does not provide built-in support of disk I/O rate control among VMs. Vicover introduces a third-party per-process I/O scheduling tool, ionice, to schedule I/O over the core dump process and kernel threads handling I/O for different virtual disks, which run in Domain0. With concurrent core dump and recovery enabled, when the recovery VM is started, Vicover invokes ionice to start scheduling with parameters specifying the priorities of core dump and recovery. When core dump ends, ionice is terminated to cancel the scheduling. After that, disk I/O request is handled in the default round-robin fashion.

For enterprise virtualization products [1, 2], there is built-in support to schedule disk I/O among VMs. Vicover, if transplanted onto these platforms, can take advantage of such built-in support to realize disk I/O rate control between core dump and recovery with slight effort.

5. Experimental Setup

This section first introduces the key metric - downtime. Then, it describes the testing framework to measure downtime, and to monitor resource usage of VMs to understand the optimizations by Vicover intuitively. Finally, it specifies the testing environment of evaluation.

5.1 Downtime as the Key Metric

Downtime is the period during which services are not available to the users. Vicover focuses on the recovery of system-wide crash of a VM. To be application-independent, downtime is measured as the time of core dump and recovery, at the server side.

On the other hand, downtime can also be measured at the client side, as the period during which client requests are not responded by the server. This measurement is more close to semantics of service availability from the perspective of a service user. However, it depends on the application-level behavior of the interaction between the client and the server, which is variant and contrary to our focus on the system-level crash.

As a result, we apply the former measurement: the downtime is measured at the server side, as the time it takes to complete core dump and recovery.

5.2 The Testing Framework

The core dump daemon of Vicover runs constantly in the background in Domain0. The workload benchmark provides services to clients in a virtualized server (which is a DomainU) that will crash, be core-dumped and get recovered. To crash this VM, we insert a Linux kernel module whose initialization function references a null pointer into its guest Linux. The core dump daemon in Domain0 detects the crash, and initiates the optimized core dump, as well as recovery.

To calculate downtime, the moments when important events happen are fetched from the server through SSH, and are logged in an *event log* file in the client. These events include the start and end of testing, core dump, recovery and etc. Note that core dump starts immediately at the moment the virtualized server crashes.

The resource consumptions of the VM to crash, the recovery VM and Domain0 are monitored and recorded at the VM management layer by the testing framework. The framework samples the resource consumption of each VM every second through Xend, and records the statistics in the client.

After core dump and recovery, statistics is parsed to draw a chart which shows the resource consumptions of VMs intuitively.

5.3 The Testing Environment

The Xen hypervisor, DomainUs providing application services, and Domain0 (including the core dump daemon of Vicover) run on a Dell PowerEdge R900 machine equipped with 4 Intel Xeon E7310 1.6 GHz 4-core processors, 32 GB memory, a Seagate SCSI disk and an Intel 82571EB Gigabit Ethernet adapter. The version of Xen is 3.3.0; guest operating systems in all VMs are 64 bit Debian Linux with kernel 2.6.18.8. Domain0 has 512 MB machine memory; the VM to crash has 4 GB machine memory; after core dump and recovery, the recovery VM uses no more than the 4 GB memory reclaimed from the crashed VM. The remaining machine memory is occupied by another idle VM to simulate a consolidated virtualized environment with little free memory.

6. Evaluation

In this section, we compare the downtime to complete core dump and recovery with optimized core dump by Vicover and the downtime of the baseline case, and give our analysis. We begin by showing the evaluation of the baseline case to core-dump and recover from a crashed virtualized TPC-W [25] server. Next, the evaluation of concurrent core dump and recovery by Vicover for the same server is given, with different chunk sizes applied. Thirdly, that of selective dump for this server is given. Then, we evaluate concurrent core dump and recovery combined with selective dump. Finally, we show the effect of disk I/O rate control by Vicover with I/O-intensive workload.

6.1 The Baseline Case

This subsection shows the baseline case of sequential core dump and recovery on system crash of a virtualized TPC-W server. The TPC-W benchmark is the TPC-W-lycos [26] implementation.

Table 1, Figure 3 and Figure 4 are the event log, CPU utilization and memory consumption of the baseline case respectively. All the server applications of TPC-W, such as the Web server and the database server, are deployed in the same VM. They are configured to always start as the last stage of booting the VM. 160 clients (or "Remote Emulated Browsers", in TPC-W's terminology) concurrently access the on-line bookstore simulated by the TPC-W benchmark.

According to the event log in Table 1, the virtualized server crashes at timestamp 27s; core dump and recovery of the TPC-W server complete at timestamp 130s. Therefore, downtime is around

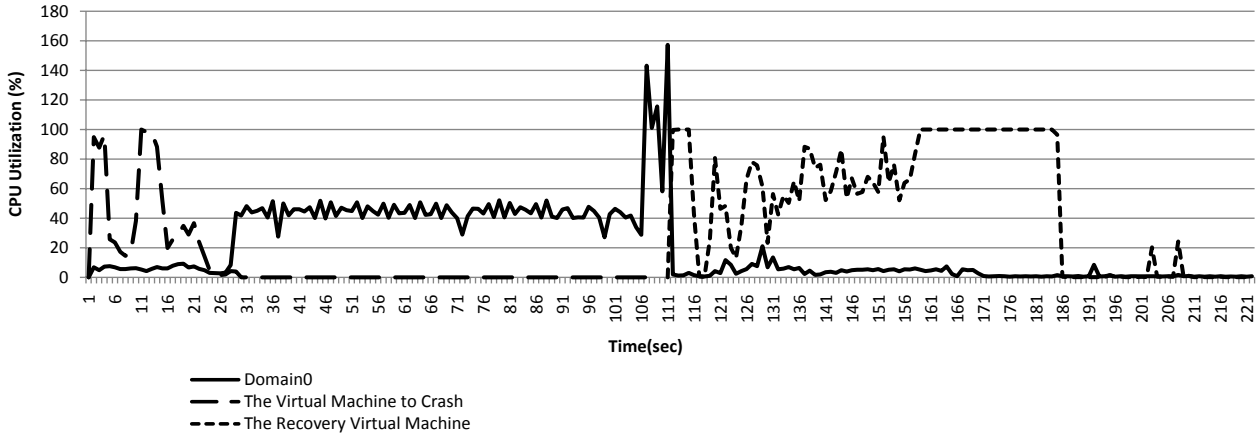

Figure 3. CPU Utilization of the Baseline Core Dump and Recovery of a Virtualized TPC-W Server

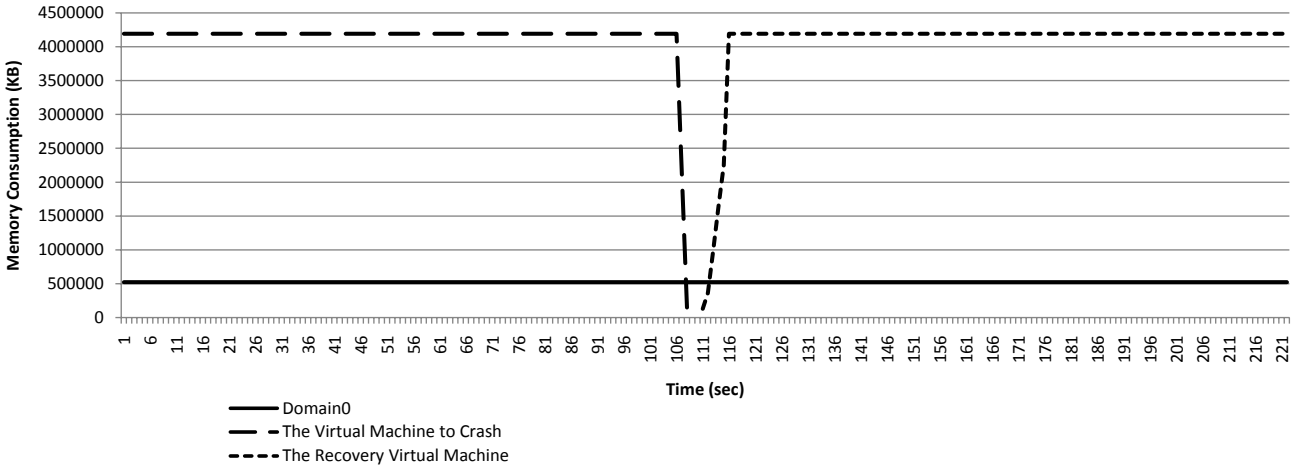

Figure 4. Memory Consumption of the Baseline Core Dump and Recovery of a Virtualized TPC-W Server

Event	Time (sec)
Start of Testing	0
System Crash & Start of Core Dump	27
End of Core Dump & Start of Recovery	108
End of Recovery	130
End of Testing	224

Table 1. Event Log of the Baseline Core Dump and Recovery of a Virtualized TPC-W Server

103 seconds, which is the difference of the two timestamps. Downtime consists of the 81-second core dump time between timestamp 27s and timestamp 108s, and the 22-second recovery time between timestamp 108s and timestamp 130s.

In Figure 3, the curve of CPU utilization of the crashed VM ends on system crash, followed by core dump during which Domain0 is the only consumer of CPU resource. After core dump, the recovery VM begins to boot and thus consumes CPU resource.

With regard to memory utilization, Figure 4 shows that 4 GB memory is reserved by the crashed VM until timestamp 108s, which is the end of core dump and the beginning of recovery, according to the event log. It is not until this moment that the 4 GB memory is reclaimed from the crashed VM, and is reallocated to the recovery VM, which leads to low memory utilization.

6.2 Concurrent Core Dump & Recovery

Table 2 and Figure 5 show the event log and memory consumption of concurrent core dump and recovery of the same virtualized TPC-W server by Vicover, with the 128-MB chunk size.

Event	Time (sec)
Start of Testing	0
System Crash & Start of Core Dump	28
Start of Recovery	35
End of Recovery	49
End of Core Dump	127
End of Testing	191

Table 2. Event Log of Concurrent Core Dump and Recovery

According to the event log in Table 2, core dump takes around 99 seconds, from timestamp 28s to timestamp 127s; and recovery takes 14 seconds. Concurrent core dump and recovery overlaps the 99-second core dump time and the 14-second recovery time. Hence, the downtime is only 21 seconds from timestamp 28s to timestamp 49s, not the sum of core dump time and recovery time. The downtime is only around 1/5 of the baseline one.

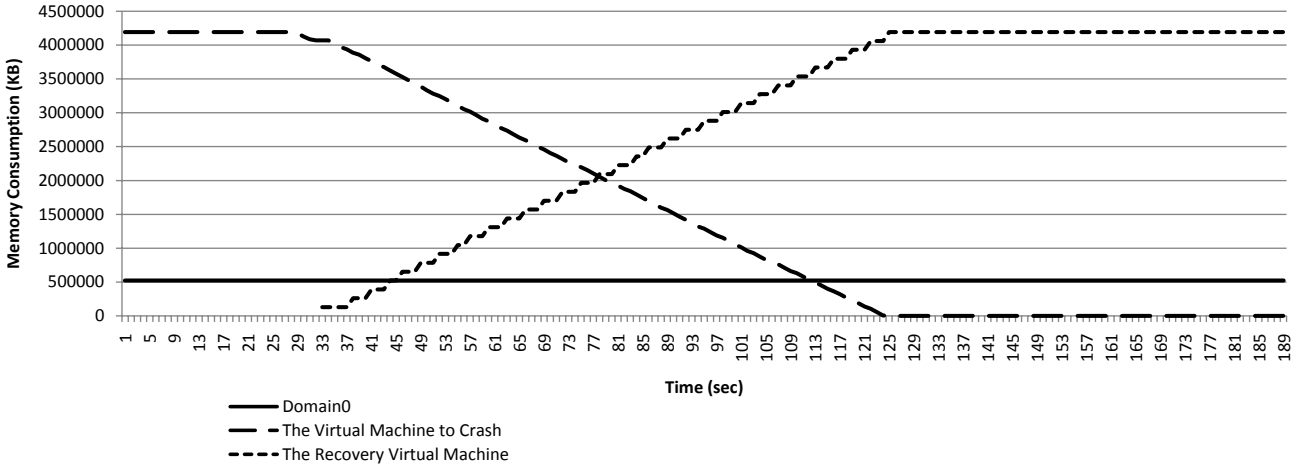

Figure 5. Memory Consumption of Concurrent Core Dump and Recovery (Chunk Size = 128 MB)

The memory consumption shown in Figure 5 illustrates such overlapping intuitively. The recovery VM is created shortly after core dump of the crashed VM begins, at around timestamp 35s, because 128 MB has been core-dumped and reclaimed in total from the crashed VM. After that, core dump and recovery progress concurrently. As a result, the memory consumption of the crashed VM shrinks linearly, while that of the recovery VM grows linearly. It continues until the former drops to zero, and the later grows to 4 GB, at the end of core dump.

The concurrency also improves CPU utilization. The core dump tool and the recovery VM consume CPU resources concurrently, adding to a total of around 84% average CPU utilization during the parallel period; while the exclusive CPU utilization by either core dump or recovery alone in the baseline case is less than 52% on average.

6.3 Evaluating Different Chunk Sizes

To show the effect of different chunk sizes on shortening downtime, we further evaluate concurrent core dump and recovery with 10-MB and 1024-MB chunk sizes, in addition to the previous 128-MB chunk size, as is shown in Figure 6 and Figure 7.

Comparing Figure 5 with Figure 6 and Figure 7, although the core dump tool always releases memory smoothly by the code of Line 5 in Figure 2, the granularities by which the recovery VM gains memory differ. The linearity of allocating memory to the recovery VM with the 10-MB chunk size is almost perfect, while that with the 1024-MB chunk size is coarse-grained.

Chunk Size (MB)	CPU Utilization of Core Dump (%)	Downtime (sec)
10	75.82	21
128	57.58	21
1024	50.57	36

Table 3. Average CPU Utilization of Core Dump and Downtime with Different Chunk Sizes

Table 3 shows the resulting CPU utilization of the core dump tool and downtime with these chunk sizes. Each pair of CPU utilization and downtime is the average of 3 same tests.

With 10 MB as the chunk size, it does not help to further reduce the 21-second downtime already achieved by the 128-MB chunk size, which means the concurrency degree with the 128-MB chunk size is high enough to allocate memory to the recovery VM early. However, the CPU utilization becomes higher with the 10-MB chunk size, due to more frequent interaction with the hypervisor to enlarge the memory of the recovery VM.

With 1024 MB as the chunk size, the overhead of interaction with the hypervisor is the smallest, leading to the lowest CPU utilization. But the memory reallocation is not fine-grained and not in time. Therefore, the downtime increases to 36 sec.

In a word, the 128-MB chunk size balances between CPU utilization and memory reallocation granularity well, and leads to the shortest downtime in our testing scenario.

6.4 Selective Dump

Table 4 shows a summary of 3 trials of selective dump by ignoring pages whose reference count is zero in the corresponding page descriptor of guest OS at the moment of crash. Selective dump is evaluated alone, not combined with concurrent core dump and recovery in this section.

	# Pages Core Dumped	Downtime (sec)
Baseline	1048576	103
Selective Dump 1st Trial	87306	32
Selective Dump 2nd Trial	82813	32
Selective Dump 3rd Trial	67366	30

Table 4. Summary of Selective Dump by Ignoring Free Pages on System Crash

There are 1048576 pseudo-physical pages in total for the 4 GB memory of the crashed VM. The result shows that in all trials, most pages are free - neither used by a process nor used for storing kernel data structures, at the moment of crash. By ignoring these pages, time of core dump as well as the overall downtime is shortened. Comparing the 3 trials, the more pages are skipped, the shorter the downtime is.

6.5 Combining Concurrent Core Dump & Recovery with Selective Dump

In our testing scenario, because both concurrent core dump & recovery and selective dump shorten downtime significantly, we evaluate the combination of the two, to see if the downtime can be further shortened. The 128-MB chunk size is used in this case.

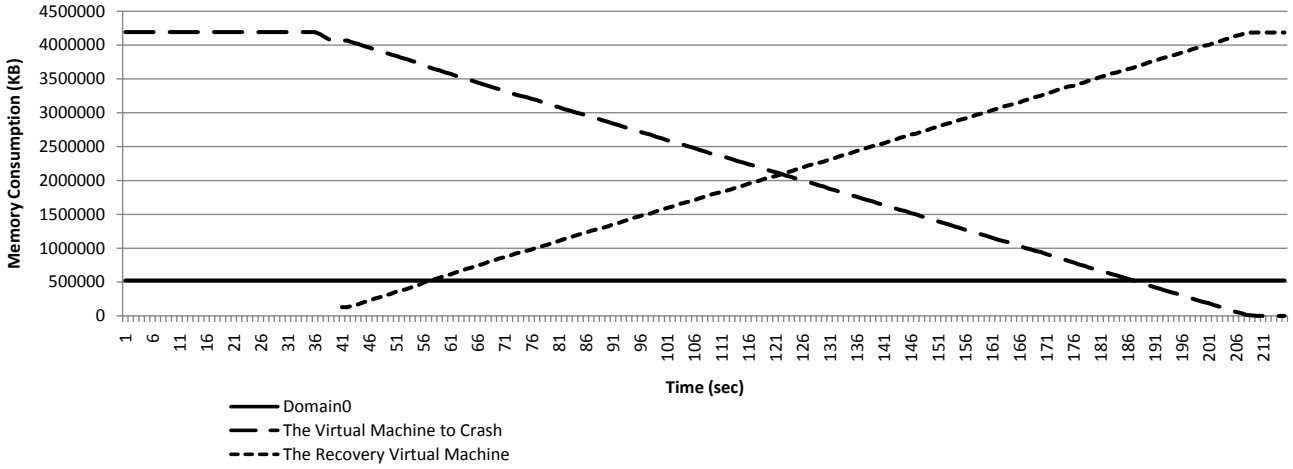

Figure 6. Memory Consumption of Concurrent Core Dump and Recovery (Chunk Size = 10 MB)

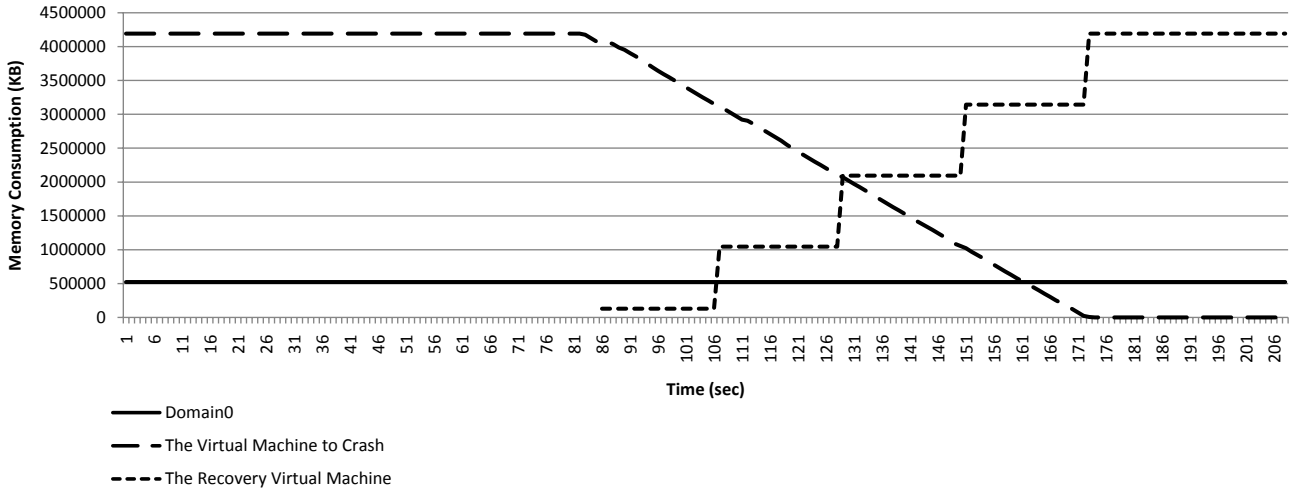

Figure 7. Memory Consumption of Concurrent Core Dump and Recovery (Chunk Size = 1024 MB)

Figure 8 shows that selective dump further accelerates memory reallocation from the crashed VM to the recovery VM, as the slope of curve is more steep compared with that by the concurrent core dump and recovery alone in Figure 5.

Concurrent Core Dump & Recovery: Alone or Combined with Selective Dump?	Alone	Combined
Downtime (sec)	21	24
Core Dump Time (sec)	98	28
Recovery Time (sec)	14	15
CPU Utilization of Core Dump (%)	58	95

Table 5. Concurrent Core Dump & Recovery: Alone v.s. Combined with Selective Dump

Table 5 compares the downtime, core dump and recovery time as well as CPU utilization of core dump between concurrent core dump & recovery alone and its combination with selective dump. The result is the average out of 3 same tests.

The downtime with selective dump combined is 24 seconds, a bit longer than the 21-second downtime achieved with concurrent core dump and recovery alone. This is because the faster memory allocation by the combination leads to more frequent interaction with the hypervisor to reallocate memory; hence, the CPU utilization of core dump becomes much higher in combined concurrent core dump & recovery and selective dump. Consequently it takes 1 second longer for recovery to complete with selective dump combined.

Therefore, Table 5 indicates that the rate of memory reallocation with concurrent core dump and recovery alone is already high enough, and combination with selective dump does not help to further reduce the downtime in our scenario.

6.6 Disk I/O Rate Control

To evaluate the effectiveness of disk I/O rate control when there is disk I/O contention between concurrent core dump and recovery, we make the booting of the recovery VM I/O-intensive but not memory-critical. To realize, a micro-benchmark repeatedly copies a 200 MB file in the local file system of the recovery VM when it is booting. The file is small and therefore the recovery VM does not need much memory for copying when it is booting during concurrent core dump and recovery. Obviously, with this additional workload during booting, it takes more time to boot until the TPC-W Web and database servers become ready. Both core dump and

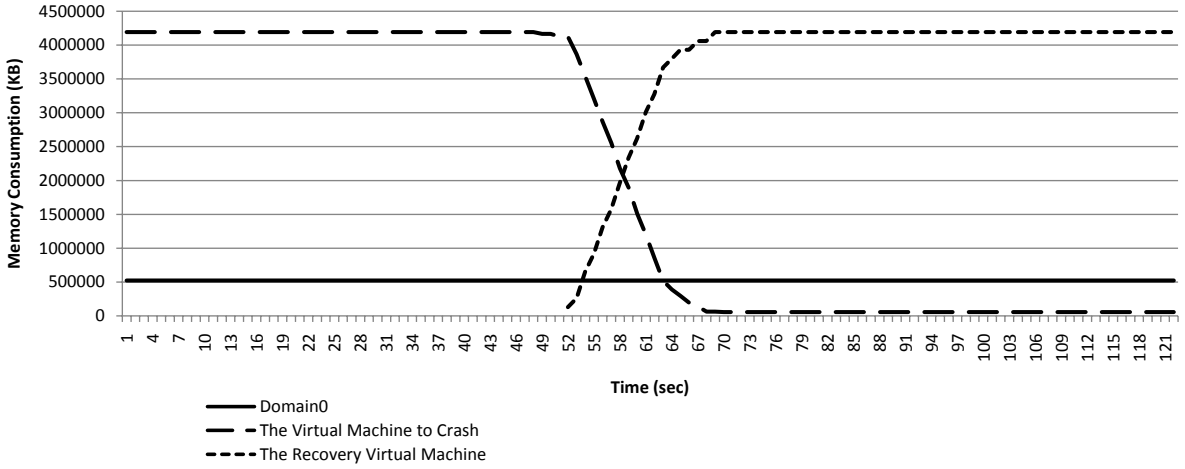

Figure 8. Memory Consumption of Concurrent Core Dump and Recovery Combined with Selective Dump (Chunk Size = 128 MB)

booting the recovery VM now contend for the limited physical disk I/O bandwidth.

Table 6 gives two sample policies of I/O allocation between concurrent core dump and recovery, and the resulting average downtime out of 3 same tests.

	Core Dump I/O Priority	Recovery I/O Priority	Downtime (sec)
Case 1	low	high	44
Case 2	high	low	89

Table 6. Disk I/O Rate Control Policies and the Resulting Downtime

In Case 1, the recovery VM is given higher priority of disk I/O while in Case 2, core dump is given higher priority. Downtime of Case 1 is significantly less than that of Case 2. This indicates that it is better to give the recovery VM higher I/O priority to boot faster, because its booting process is I/O-intensive but not memory-critical, and is therefore more eager to process I/O than to grab more memory from core dump.

The memory consumption graphs for both cases, Figure 9 and 10, further illustrate the effect of disk I/O rate control.

After the recovery VM starts with the I/O-intensive micro-benchmark, the slope of curve is more gentle in Figure 9 than in Figure 10. It indicates that memory reallocation in Case 1 is slower than that in Case 2. This is because in Case 1, core dump tool has lower I/O priority and thus reclaims memory more slowly.

The comparison between Case 1 and 2 reveals that for I/O-intensive and not memory-critical recovery, explicitly setting higher priority to recovery over core dump leads to smaller downtime of concurrent core dump and recovery. Yet, the trade-off is that memory reallocation to the recovery VM is slower, which leads to less available memory to process application requests with, after recovery ends, and before core dump finishes reclaiming the memory.

7. Related Work

Core dump is essential for debugging to achieve robustness in the long term, while fast recovery is important to achieve high availability on system crash [9–11]. Current research works of reboot-based recovery exist in both applications and in systems to recover from system crash quickly without knowing its root cause.

At the application-level, Microreboot [12] proposes rebooting well-isolated stateless application components, rather than the whole application, separating data recovery from application recovery. Crash-only software [13] is the design pattern to build microrebootable systems.

At the system-level, the Recovery Box [14] tries to recover from system failure with backup application and OS data stored in the recovery box, rather than from scratch, which relies on the integrity of the recovery box. Vicover can't backup and restore in-memory application session state as it is application-agnostic, as a limitation. Kexec boots into a new kernel from a reserved memory area, skipping hardware reset, firm operations and the bootloader stage. Kdump [7] modifies Kexec to boot into a new kernel fast and automatically on kernel panic without corrupting memory of the crashed one. Then, core dump is done reliably in the context of the new kernel. In Vicover, the isolation enforced by virtualization serves the purpose of rebooting the system with the recovery VM without corrupting the memory of the crashed VM. [15] analyzes the boot process of Linux, and optimizes it by system configurations such as ROM-based kernel loading, manually setting loop delay value, avoiding redundant probing and etc. Starting heavy-weight services later in the recovery VM helps to leave more time to reclaim memory by core dump, before the recovery VM consumes too much memory concurrently. Yet, it is not a must-to-do in order to realize concurrent core dump and recovery. Vicover optimizes core dump at the virtual machine layer; so it does not rely on the system configuration inside a VM.

At the virtualization level, Vicover is the first to optimize core dump of a VM on system crash to minimize downtime, borrowing existing virtualization techniques developed for other purposes.

To realize concurrent core dump and recovery, Vicover adopts dynamic resource allocation among VMs, which is borrowed from other research works about server consolidation to improve throughput of virtualized servers. AutoControl [16], VMware ESX [17], and MEmeory Balancer [18] dynamically allocate memory among VMs for better throughput; meanwhile, memory reallocation by Vicover aims at minimizing downtime.

Vicover introspects the crashed VM to extract the page descriptor array of the crashed guest OS, identifies free pages indicated by it, and does not save them during core dump. Virtual machine introspection and related techniques are proposed in various research works to build security tools and etc, including Liveware [19], Overshadow [21], Antfarm [20], Lycosid [22], VIX [23] and FVM [24].

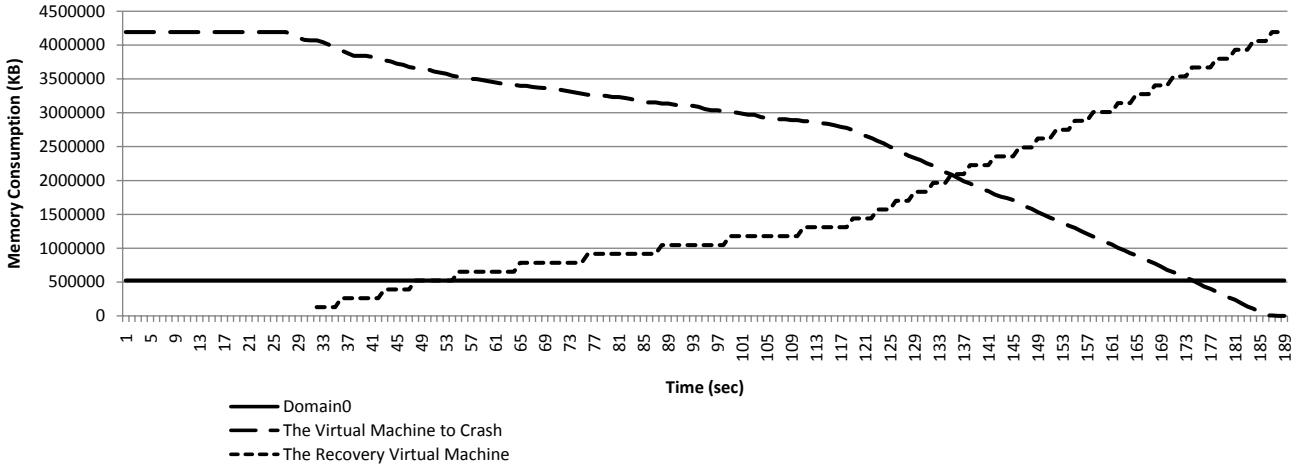

Figure 9. Memory Consumption of Case 1: Higher I/O Priority Assigned to Recovery (Chunk Size = 128 MB)

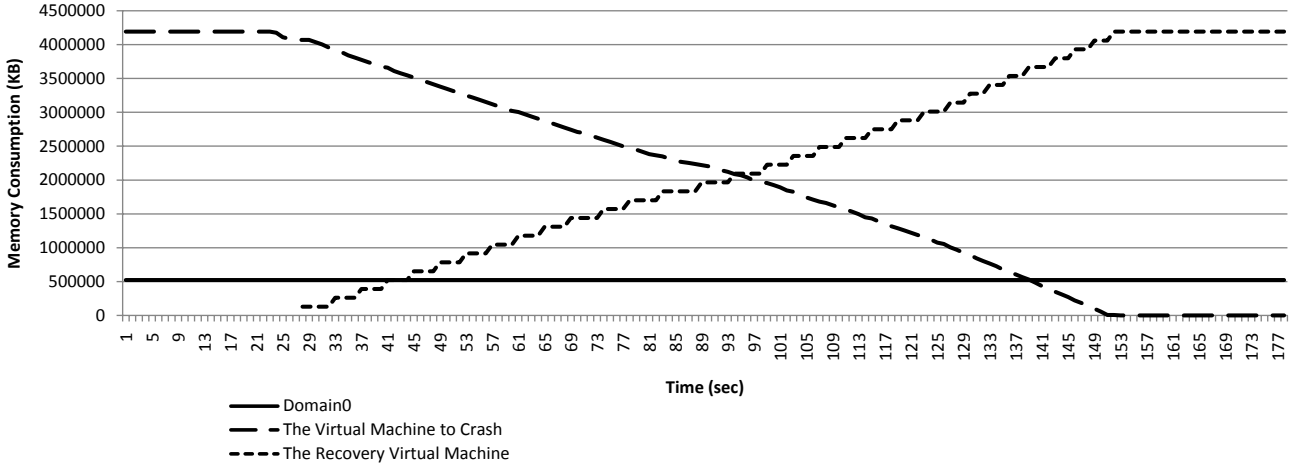

Figure 10. Memory Consumption of Case 2: Higher I/O Priority Assigned to Core Dump (Chunk Size = 128 MB)

8. Conclusion

We propose optimizing core dump at the VM management layer on system crash of a VM, to minimize the downtime it takes to complete core dump and recovery. The recovery is based on rebooting services in the recovery VM. The demonstrated optimizations include concurrent core dump and recovery through concurrent memory reallocation from the crashed VM to the recovery VM, selective dump by introspecting the page descriptor array of guest OS to identify and ignore free pages, and disk I/O rate control between concurrent core dump and recovery. The prototype, Vicover implements these optimizations in the VM management tool of Xen. Experimental results show that Vicover shortens the downtime to core-dump and recover a crashed virtualized TPC-W server by 5X.

Acknowledgments

We thank Ping-Hui Kao for his initial input that motivates this work, Rong Chen for his efforts in discussing and refining the paper and the anonymous reviewers for their valuable comments and suggestions. This work was funded by China National High-tech R&D Program (863 Program) under grant numbered 2008AA01Z138, China National 973 Plan under grant numbered 2005CB321905, Shanghai Leading Academic Discipline Project (Project Number: B114), and a research grant from Intel numbered MOE-INTEL-09-04.

References

[1] Critix Inc. XenServer. http://www.citrix.com, 2009.

[2] VMware Inc. VMware ESX Server. http://www.vmware.com/products/esx/index.html, 2009.

[3] Ganapathi, A. and Patterson, D. Crash Data Collection: A Windows Case Study. In *Proceedings of Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 280–285, 2005.

[4] GNU. The GNU Project Debugger. http://www.gnu.org/software/gdb/, 2009.

[5] Ganapathi, A. and Ganapathi, V. and Patterson, D. Windows XP kernel crash analysis. In *Proceedings of Usenix Large Installation System Administration Conference*, 2006.

[6] David Anderson. White Paper: Red Hat Crash Utility. http://people.redhat.com/anderson/crash_whitepaper/, 2008.

[7] Goyal, V. and Biederman, E. and Nellitheertha, H. Kdump, A Kexec-based Kernel Crash Dumping Mechanism. In *Proceedings of Annual Ottawa Linux Symposium*, pages 169–180, 2005.

[8] Barham, P. and Dragovic, B. and Fraser, K. and Hand, S. and Harris, T. and Ho, A. and Neugebauer, R. and Pratt, I. and Warfield, A. Xen

and the Art of Virtualization. In *Proceedings of ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[9] Mauro, J. and Zhu, J. and Pramanick, I. The system recovery benchmark. In *Proceedings of Pacific Rim International Symposium on Dependable Computing*, pages 271–280, 2004.

[10] Patterson, D. and Brown, A. and Broadwell, P. and Candea, G. and Chen, M. and Culter, J. and Enriquez, P. and Fox, A. and Kcman, E. and Merzbacher, M. and Oppenheimer, D. and Sastry, N. and Tetzlaff, W. and Traupman, J. and Treuhaft, N. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. Computer Science Division, U.C. Berkeley, UCB//CSD-02-1175, 2002.

[11] Fox, A. and Patterson, D. When Does Fast Recovery Trump High Reliability? In *Proceedings of 2nd Workshop on Evaluating and Architecting System Dependability*, 2002.

[12] Candea, G. and Kawamoto, S. and Fujiki, Y. and Friedman, G. and Fox, A. Microreboot - A Technique for Cheap Recovery. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, pages 31–44, 2004.

[13] Candea, G. and Fox, A. Crash-only Software. In *Proceedings of Workshop on Hot Topics in Operating Systems*, pages 67–72, 2003

[14] Baker, M. and Sullivan, M. The Recovery Box: Using Fast Recovery to Provide High Availability in the UNIX Environment. In *Proceeding of Summer USENIX Technical Conference*, 1992.

[15] Bird, T. Methods to Improve Bootup Time in Linux. In *Proceeding of Ottawa Linux Symposium*, pages 79–88, 2004.

[16] Padala, P. and Hou, K. and Shin, K. and Zhu, X. and Uysal, M. and Wang, Z. and Singhal, S. and Merchant, A. Automated Control of Multiple Virtualized Resources. In *Proceedings of European Conference on Computer Systems*, pages 13–26, year 2009.

[17] Waldspurger, C. Memory Resource Management in VMware ESX Server. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, pages 181–194, 2002.

[18] Zhao, W. and Wang, Z. Dynamic Memory Balancing for Virtual Machines. In *Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 21-30, 2009.

[19] Garfinkel, T. and Rosenblum, M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of Annual Network & Distributed System Security Conference*, pages 191–206, 2003.

[20] Jones, S. and Arpaci-Dusseau, A. and Arpaci-Dusseau, R. Antfarm: Tracking Processes in a Virtual Machine Environment. *Proceeding of Usenix Annual Technical Conference*, pages 1–14, 2006.

[21] Chen, X. and Garfinkel, T. and Christopher Lewis, E. and Subrahmanyam, P. and Waldspurger, C. and Boneh, D. and Dwoskin, J. and Ports, D. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. In *Proceeding of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[22] Jones, S. and Arpaci-Dusseau, A. and Arpaci-Dusseau, R. VMM-based Hidden Process Detection and Identification Using Lycosid. Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pages 91–100, 2008.

[23] Nance, K. and Bishop, M. and Hay, B. Virtual Machine Introspection: Observation or Interference. In *Proceeding of IEEE Symposium on Security and Privacy*, pages 32–37, 2008.

[24] Zhang, Y. and Bestavros, A. and Guirguis, M. and Matta, I. and West, R. Friendly Virtual Machines: Leveraging a Feedback-control Model for Application Adaptation. In *Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 2–12, 2005.

[25] Transaction Processing Performance Council. The TPC-W Benchmark. http://www.tpc.org/tpcw/default.asp, 2009.

[26] Bezenek, T. and Cain, T. and Dickson, R. and Heil, T. and Martin, M. and McCurdy, C. and Rajwar, R. and Weglarz, E. and Zilles, C. and Lipasti, M. Characterizing a Java implementation of TPC-W. In *Third Workshop On Computer Architecture Evaluation Using Commercial Workloads*, 2000.