2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

TMemCanal: A VM-oblivious Dynamic Memory Optimization Scheme for Virtual Machines in Cloud Computing

Yaqiong Li[1,2], Yongbing Huang[1,2]
[1] Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, P.R.China; [2]Graduate University of Chinese Academy of Sciences, P.R.China

Abstract—**In current virtualized cloud platforms, resource provisioning strategy is still a big challenge. Provisioning will gain low resource utilization based on peak workload, and provisioning based on average work loads will sacrifice the potential revenue of cloud customers because of bad user experiences. VM-based performance isolation also restrains resource flowing on demand. As to memory, this eventually results in *under-loaded* memory and *over-loaded* memory in the same data center. This paper proposes a VM-oblivious dynamic memory optimization scheme, *TMemCanal*, which leverages under-loaded memory in a data center to accommodate the needs of loaded memory dynamically in a transparent fashion. *TMemCanal* is able to identify the under-loaded memory located in different VMs and reuse it in a way of memory flowing without any modification to their Guest OSs. We implemented *TMemCanal* through extending Xen hypervisor and evaluated using SpecWeb 2005 and LinkPack Benchmarks. Our evaluation shows that *TMemCanal* can efficiently save memory up to 50% with an overhead less than 7%. Our case study of server consolidation also shows *TMemCanal* can promote the performance of memory-intensive services up to 400%.**

Keywords: under-loaded memory; Xen

I. Introduction

Nowadays, lots of enterprises or firms, such as Google [29], Microsoft [26] and Amazon [27, 28], embrace cloud computing, because it can optimize resource utilization for cloud providers and provide elastic economic modules for cloud customers, for example, pay as you go. But, for cloud customers, resource provision strategy is still a new challenge. Provisioning based on peak workload will result in low resource utilization and high cost of ownership. In current data centers, server utilization range from 5% to 20% verifies this point [24, 25]. On the other hand, under-provisioning based on average workload will gain a low service rate, which eventually brings potential revenue sacrifices because of bad user experiences [30]. These demand an elastic resource provision scheme.

Virtualization has been deployed in these cloud platforms to consolidate servers, increase resources utilization and decrease the total cost of ownership (TCO). However, lots of legacy operating systems hosted by virtual machines (VMs) take aggressive strategies of resource utilization. For example, a large amount of memory in Linux is used as buffer cache to promote disk performance, though its speedup of performance mainly depends on access modes of applications. These may eventually hamper the global optimization with varying of applications or fluctuating of workloads. In the same data center, free or underutilized memory, *under-loaded memory* we called, coexists with the unsatisfied desire for more memory from other memory-intensive VMs. This situation will become worse in many-core era, because the average memory capacity per core will decrease with the increase of core number in one chip.

In this paper, we present *TMemCanal*, a VM-oblivious memory optimization scheme, which optimize memory allocation through releasing under-loaded memory in a way of memory flowing without modification to Guest OSs of VMs. *TMemCanal* is able to transparently withdraw under-loaded memory with low performance penalty and reutilize it to promote the performance of other memory-intensive VMs.

Unlike previous efforts, such as Content-based sharing [9, 10] or Ballooning [9, 11], *TMemCanal* is built on memory access frequency and tries to reutilize under-loaded memory in a VM-oblivious approach. The transparency to VMs can be used to construct a unified memory optimization scheme and reduce the technical complexity in a virtualized data center hosting a variety of legacy operating systems, while most of previous works need modifying Guest OSs to insert new drivers [9, 11, 14] or limit themselves to only homogeneous VMs [9, 10].

We implemented *TMemCanal* as an extension of Xen [1, 7], an open source hypervisor. To reduce the performance impact of remote memory access, *TMemCanal* takes a VM-oblivious low-overhead page swapping framework to exchange memory pages amongst different physical servers. This framework proposes a "*mark-copy-validate*" asynchronously processing mechanism to eliminate local cold pages into remote server, which doesn't suspend the execution of the VM. A mechanism of remote memory loading based on *instruction rollback* and *exception replay* is also added into Xen to support remote memory transparent access.

In *TMemCanal*, a multi-layer wide-range latency memory hierarchy is constructed based on memory access frequency. We assign different states to memory pages based on their access latencies and usage modes. The allocated memory of a VM, including under-loaded memory, is dynamically mapped into multiple storage layers of this hierarchy. To implement the memory flowing, we introduce a shared memory pool to leverage memory allocation amongst VMs and an asynchronous paging mechanism to release the under-loaded memory pages of VMs into this pool. These two technologies are built based on the memory hierarchy just mentioned.

To evaluate *TMemCanal*, we deploy *TMemCanal* on our cloud-oriented platform, Rainbow [20, 21, 22]. Our evaluations show that *TMemCanal* can reduce the memory allocation by up to 50% with an overhead less than 7%. In a server consolidation case study, *TMemCanal* promotes the performance of memory-intensive services up to 400%.

The contribution of this paper is three-fold: (1) we observe the under-loaded and over-loaded memory problem in virtual machines in clouds; (2) we propose *TMemCanal* to address this problem by dynamically provisioning of memory; and (3) we have conducted a comprehensive evaluation in the context of Xen-based platforms and show that *TMemCanal* is very promising.

The remainder of this paper is organized as follows. Section 2 will review related works. The architecture and key techniques will be discussed in Section 3 and 4. Section 5 describes the implementation of *TMemCanal*. In Section 6, we evaluate *TMemCanal* and analyze the experimental results. Finally, conclusion remarks and future work are listed in Section 7.

II. RELATED WORK

VMWare ESX Server [9] introduced the content-based page sharing, which can exploit the potential memory sharing opportunities in a virtualized server. However, its better performance depends greatly on the operating system and configuration of the guest VMs [10], such as performing better amongst homogeneous VMs. Ballooning [9, 11] is an efficient approach to leverage active memory allocation of VMs. However, it needs inserting new drivers into Guest OSs to inflate or deflate their memory allocation. These restraints affect active performance and increase technical complexity when deploying these technologies in a virtualized data center hosting a variety of legacy operating systems. Our work can provide a unified solution to leverage memory allocation through efficient paging and maximize memory performance of a distributed system through the multi-layer wide-range latency memory hierarchy.

Gupta et al. [10] implemented three multiplexing techniques, page sharing, delta encoding and memory compressing, in Xen hypervisor [1, 7], which can efficiently save memory with low overhead. It also presented a swapping-based architecture to handle the memory over committing case. This architecture is very similar to our implementation. However it didn't give the efficient paging approach based on under-loaded page detection. Our solution introduces an asynchronous way to swap under-loaded pages into local disk or remote free memory to reduce performance penalty. Based on our memory hierarchy, we also give a memory optimization scheme through memory flowing.

Kinshuk Govil [34] implemented a virtual resource management on shared-memory multiprocessors system. This system introduced a paging scheme to optimize memory performance, which is very similiar with our work. However, our work has special challenges. Because our work focuses on the cluster system instead of shared-memory multiprocessors system, we must implement the remote memory access through software approach without hardware support. So, we introduced an asynchronous approach named with "mark-copy-update" to swap pages and an LRU-Zone based page prefetching to reduce the performance penalty.

Based on Ghost Buffer [23] and Ballooning, Lu et al. [14] and Jones et al. [15] respectively implemented a hypervisor-level exclusive cache system to leverage the memory allocation amongst different virtual machines hosted in the same server. Magenheimer et al. [17] implemented a unified memory resource pool in Xen hypervisor with a new set of APIs. Milso et al. [16] implemented a low-overhead scheme to find the short-lived page sharing. Their works mainly focused on the buffer cache sharing and optimization amongst VMs in one physical server with modification to Guest OSs. They don't also concern the global optimization in a virtualized distributed system.

As to virtualized resource optimization, Padala et al. [31] explored dynamic resource allocation to satisfy service-level objectives (SLOs) of applications. Their research considered only CPU and disk resources. Song et al. [21, 22] exploited memory optimization scheme based on Ballooning by inserting ballooning driver into Guest OSs. Anemone [4] and MemX [5] exploited the usage of remote memory in a virtualized server. In their implementation, remote memory is used as ram disk. Chapman and Heiser [13] implemented a virtual shared-memory multiprocessors system based on Xen in a commercial cluster. It maintained a complex coherency protocol to support a virtual NUMA architecture. Compared with these efforts, our solution is a simple and flexible way to reutilize the distributed memory. In our implementation, remote free memory is thought as the bottom layer of the memory hierarchy of TMemCanal. Through the memory abstract, this memory space can be used as either faster ram disk or a faster cache in shadow cache.

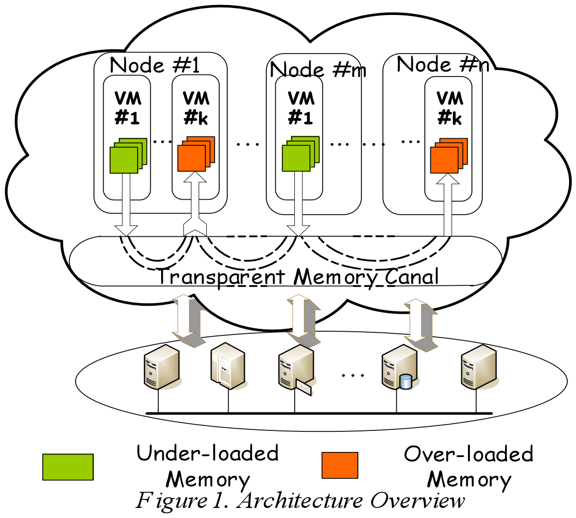


Figure 1. Architecture Overview of TMemCanal

III. SYSTEM ARCHITECTURE

To dynamically reuse the under-loaded memory in different VMs, we construct *TMemCanal* in two steps: first, we exploit the VM-oblivious technology to transparently detect and withdraw the under-loaded memory located in each VM; second, we build the under-loaded memory

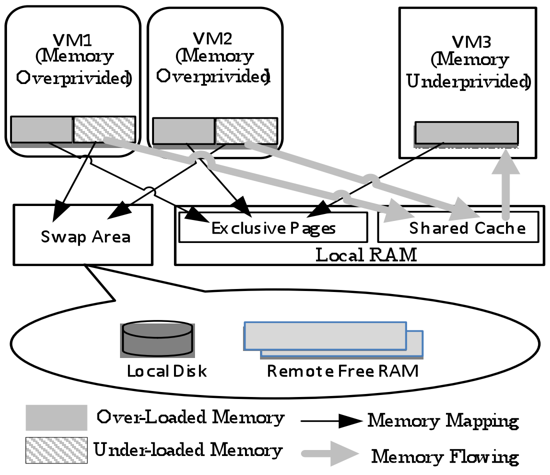


Figure 2. Architecture Overview of TMemCanal Paging

management framework and reutilization scheme from VM's view to promote memory utilization.

Figure 1 illustrates the overview of *TMemCanal*. In *TMemCanal*, we build a virtual memory canal connecting different VMs to optimize the memory allocation across VMs. All memory pages are managed under a reconstructed memory hierarchy based on their usage states and access latencies. Memory pages of a VM are mapped into different storage layers based on their access frequency.

Given the fact that there are many different implementations of the concept of virtual machines, we pick up Xen as the base of our design and implementation since it is open source nature and popular in the research community. TMemCanal extends Xen in two aspects: transparent paging and memory hierarchy reconstruction.

A. Paging Extension

To provide transparent memory flowing mechanism, we extend Xen to support transparent paging. Figure 2 shows the overview of the TMemCanal paging architecture, where VM1 and VM2 are over-provisioned VMs (i.e., they have some under-loaded memory available) and VM3 are under-provisioned VM. In this figure, under-loaded memory pages of an over-provided VM are remapped into the swap areas formed by local disk and remote free memory. Once an under-loaded page is released, it will join the shared cache pool, to be used for other under-provisioned VMs later. This paging mechanism and the shared cache pool form the memory canal amongst VMs hosted in the same physical server.

Figure 3 illustrates the necessary extensions to Xen. In TMemCanal, three extensions are added, including Swap Area Management, Swap Runtime and Swap Domain.

Swap Area Management

Swap Area management includes two parts: *Swap Server* management and *Swap Address Space* management. A *Swap Server* is a page-level storage space provider. The storage space is provided either from local disks or remote free memory.

Swap Runtime

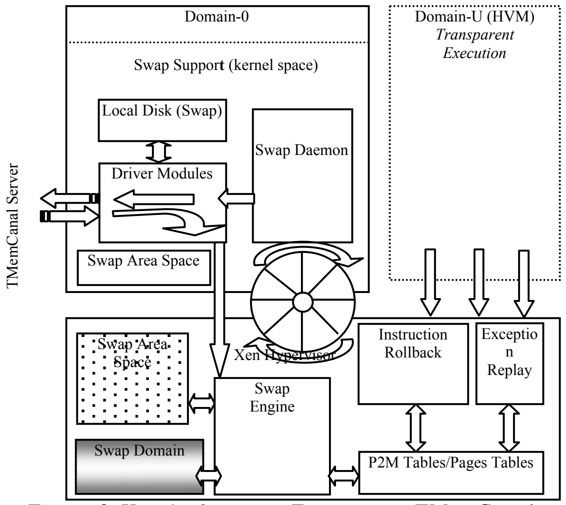


Figure 3. Xen Architecture Extension in TMemCanal.

Swap Runtime is the core of the extensions to Xen. This extension is divided into two parts respectively located in Xen and Domain-0 kernel space. In Xen, the runtime is formed by swap engine, instruction rollback and exception relay which work together to provide efficient paging. The swap support function is added into Domain-0's kernel space, which includes some swap daemons and driver modules to complete page storing. These two parts communicate by FIFO Message Rings illustrated by a ring in Figure 3.

Swap Domain

Swap Domain is special domain created by TMemCanal to manage the released under-loaded memory pages as a shared memory cache pool. Additionally, Swap Domain also provides the memory pages to construct the FIFO Message Rings used by Swap Runtime.

With collaboration of these three extensions, TMemCanal implements two core functions: swapping out and swapping in. To reduce the performance impact of swapping pages out, TMemCanal proposes a "mark-copy-validate" asynchronous approach to release under-loaded pages without suspending the VM. TMemCanal can intercept all memory access to swapped pages and execute the swapping in with instruction rollback and exception replay mechanism. The details of these modules will be discussed further in Section 4.

B. Memory Hierarchy Reconstruction

From a VM's point of view, its memory hierarchy is reconstructed. A multi-layer wide-range latency memory hierarchy is built. In this hierarchy, local memory of a VM is regarded as a huge exclusive cache. Besides the existing memory layers, such as L1, L2, L3 and Local RAM, TMemCanal additionally introduces two extra layers including a shared cache amongst different VMs and a virtual global memory space for external storage devices. Memory pages of a VM are dynamically mapped into different storage layers based on their access frequency. Figure 4 illustrates the multi-layer memory hierarchy.

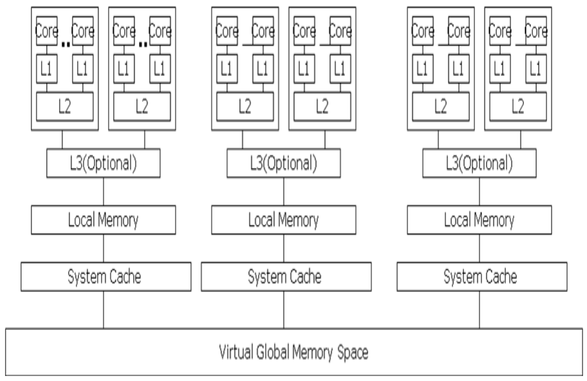


Figure 4. Memory Hierarchy of TMemCanal.

Memory pages in this hierarchy are marked as different usage states based on their usage modes and access latencies. The details are listed as follows:

Local Exclusive: The local dedicated pages of a VM hold local exclusive state. These pages can't be released by the hypervisor, but they can be converted into local shared by paging if they are marked as under-loaded.

Local Shared: The local pages without local exclusive state are all local shared state. They can be used by any VM in this server and transfer into local exclusive state.

Remote Shared: All external storage devices are thought as non-reliable, because TMemCanal assumes that external servers can reclaim the shared storage space at any moment.

Back to the memory hierarchy, the two extra storage layers are discussed in the following.

System Cache

This cache system is a shared memory pool that collects all under-loaded memory and assigns memory to VMs with high memory requirements.

In our paging implementation, this layer is an evict-based shared cache system. When an under-loaded page has been swapped out, it will be sent to the cache pool instead of releasing it immediately. This can efficiently avoid performance thrashing when a page is accessed immediately after swapped out. This memory can also be used to construct shadow cache like in previous works [14, 15, 23].

Virtual Global Memory Space

Virtual global memory space is the abstract of external storage space, including local disk and remote memory. It is managed under a two-tiered address space. In Domain-0, an address space daemon instance maintains a storage device and its address space. Simultaneously, Xen maintains a global external storage space. Domain-0 maps device address spaces into the global address space in Xen. This mechanism can provide redundancy quality through mapping two or more device spaces into the same global address range in Xen.

In our paging implementation, swap area space is built based on this mechanism. To balance reliability and performance, we map both local disk and remote free memory into the same swap area address space.

IV. KEY ENABLING TECHNIQUES

A. Asynchronously Xen Paging

To reduce the overhead of paging, TMemCanal presents an asynchronous paging approach based on mark-copy-validate processing flow.

When an under-loaded page is selected to swap out, TMemCanal firstly allocates a swap area address index in Xen swap area space. After clearing access-bit of the page, TMemCanal will mark the page as "clean and to be copied". In the second step, TMemCanal releases all mappings from Domain-U to this page and then sends a mapping invalidation notification to the devices emulator (qemu-dm) [8] in Domain-0 through buffered-io channel between Xen and qemu-dm. We have modified qemu-dm to handle this invalidation command to release its all mappings to this page.

One FIFO Message Ring is used to transfer paging command from Xen to Domain-0. Krswapoutd, the helper kernel thread, will process all paging requests in Domain-0 kernel space through copying the page content to swap area storage devices. This is the "copy" process. Through a new hypercall, the helper will notify Xen after copying. At this moment, Xen will check whether the page has been accessed during data copying. If not, Xen will set the entry in p2m tables with swap area address index and a new p2m type, p2m_swap_area, and then put the page to the shared cache pool. This process is called "validate". To guarantee correctness, the "validate" process is executed through an atomic operation.

B. Instruction Rollback and Exception Replay

When a swapped page is accessed, a swapping-in processing will be triggered. If the access is happening in Domain-U's context, it maybe is one of the following three situations: Exception/interrupt handling, sensitive instructions emulating and devices emulating DMA. If this page isn't cached in the evict-based cache pool, TMemCanal will issue a swapping in request into another FIFO Message Ring. Simultaneously, a xen_swap_exception is triggered. Swap Runtime will keep transferring this exception to upper callers layer by layer until it reaches the entry point through which domain-U traps into Xen. In this place, TMemCanal will suspend current vcpu after saving its current state (including the blocked instruction or exception) and reschedule immediately. This is called instruction rollback, because most of these happen when Xen emulates the sensitive instructions of CPUs.

When the page is back, TMemCanal will wake the paused vcpu up and reschedule immediately. If a vcpu is selected to occupy current cpu slice, Swap Runtime will check whether it resumes from waiting for the swapped page. If yes, it will resume the blocked instruction or exception handling. This mechanism is call exception replay.

When Domain-0 accesses a swapped page, a swapping in request will also be issued, but it returns from Xen with an error immediately. Domain-0 will check whether this error is triggered by a swapping-in exception. If yes, it will retry until the corresponding page returns.

C. *Under-loaded Memory Detection and Optimiization*

To identify the under-loaded memory pages, TMemCanal first tracks all memory access of a VM. TMemCanal tracks memory access in two ways:

- ✧ *Hook all conversion of pseudo-physical address to machine address.*
- ✧ *Scan the shadow page tables [] or NPT in AMD SVM (EPT in Intel VT-x) periodically.*

Based on these two tracking approaches, TMemCanal implements a memory management scheme based on a modified multi-queues algorithm [18] and an LRU-Zone-based prefetching mechanism.

In our modified multi-queue algorithm, priority-based multiply queues and an out queue are used to manage all memory pages of a VM, in which multiply queues manage recently accessed pages and drop a page into out queue if it has not been accessed for a long time (expire time). We introduced two types of expire time to extend basic multi-queue algorithm. Expire-time-I is used to eliminate pages from high-priority queue to low-priority queue in multiply queues. This parameter is determined by scanning period, mostly by a factor of 10~20. In our implementation, we select 10. Expire-time-II is used to eliminate page from multiply queues to out queue. This parameter is determined by applications. Different access modes of applications use different values. When a page is dropped into out queue, it will be thought as under-loaded memory and be swapped later.

An LRU-Zone is a continuous size-fixed range of pseudo-physical memory and the machine pages mapped into this range are under the management of this zone. The pseudo-physical memory space of a domain-U is divided into several LRU-Zones. These zones are linked into an LRU list. When a page is accessed, the zone it belongs to will be moved to the list head. Mainly this concerns the spatial locality of memory access. Figure 5 shows the LRU-Zone algorithm. In LRU-Zone algorithm, the pages to be prefetched are selected from recently accessed zones. TMemCanal will check the swapped pages from these zones when a prefetching request is issued. Generally, a prefetching request is issued periodically or when a swapped page is accessed.

Name:

Page Prefetching Algorithm

Notations:

dom - struct domain, containing multi-queue policy and a LRU-based zone list
count - number of pages needed to prefetch
range - the range prefetched pages located in
pfn_array - an array containing the pfns of prefetched pages

Description:

Prefetch an amount of pages stored in remote memory within a dom's certain range.
Input: dom, count, range;
Output: pfn_array;

Algorithm:

```
start:
    pfn_index = 0;
    for each zone in dom's zone list:
        if zone belongs to range:
            for each pfn within zone:
                if pfn is in remote memory:
                    pfn_array[pfn_index++] = pfn;
                    if(pfn_index == count)
                        goto end;
end;
```

Figure 5. LRU-Zone Prefetching Algorithm.

V. Implementation

We implemented TMemCanal based on Xen-3.3.1 platform. Because of the assumption that we can't modify Guest OSs, we implement TMemCanal in a full-virtualization environment.

A. *Xen Kernel*

We introduce a new type of p2m table entry, p2m_swap_type. When a page is swapped, its p2m entry will be set with p2m_swap_type and the corresponding index in swap area address space. Through this, TMemCanal can identify a swapped page and get it back when it is accessed in the future.

To support instruction rollback and exception replay, we have modified the handling mechanism of VMExit exception and the code of resuming vcpu. These have been discussed in previous sections.

B. *Domain-0 Kernel*

Because of the lack of communication software stack and device drivers in Xen, we inserted several new modules in Domain-0 kernel to drive the swap devices including local disks and remote memory. Three shared memory-based FIFO Message Rings are introduced respectively to hold swapping in, swapping out and swapping control requests issued by Xen, which are used to transfer data or control information from Xen to Domain-0. A new hypervisor-call is also introduced to notify Xen when paging.

C. *Qemu Issue*

Nowadays, Xen uses an application called qemu-dm to emulate various devices in a full virtualization environment. We introduce an asynchronous buffered-IO event, which tells qemu-dm to invalidate the mapping to under-loaded pages. To assure the success of invalidation, TMemCanal will issue swapping out request until it pasts enough long time after the invalidation event has been issued. Normally, it is a few scheduling cycles.

VI. Evaluations and Analysis

To study TMemCanal, we presented several benchmarks to evaluate it. First, we study the performance impact on a VM when releasing its under-loaded memory. In this group of experiments, we observe the performance varying of VMs with TMemCanal. Second, we study a case of server consolidation using TMemCanal to evaluate the memory optimization in a physical server. For our experiments, we use two physical servers as physical platform, in which Server#1 is used as TMemCanal server providing free memory as swap area. The details of these two servers are illustrated by Table 1.

Server	CPU	Memory	Network	Role
Server#1	2X CPU(AMD Quad- core CPU, 2.0GHZ)	8GB	1Gb Ethernet	Memory consumer
Server#2	2X CPU(AMD quad- core CPU, 2.0GHZ)	8GB	1Gb Ethernet	Memory Producer

Table 1. Physical Servers Configuration

A. *Performance Impact of TMemCanal*

In this group of experiments, we evaluate the performance impact of a VM with TMemCanal to release its under-loaded memory. We select SpecWeb2005 [2] as the benchmark running on the VM hosted in Server#2, because web-based services are prevalent in an enterprise-level data center. To measure the performance impact in detail, we evaluate TMemCanal in different situations. First, we evaluate the performance varying of a VM under different workload pressure. Second, we measure the performance of SpecWeb2005 with different memory size and evaluated the performance impact in this situation.

Figure 6(a) illustrates the performance penalty of a VM with TMemCanal to release its under-loaded pages under different workload pressure. The blue line is the baseline that shows the performance without TMemCanal under different workloads. The measure of the performance is the response rate of the requests under different requests per second. The red line represents the performance curve of the VM that freed its 128MB memory with TMemCanal under different workloads. The results show that TMemCanal almost have no performance impact when the VM is under low workload pressure. Under high pressure of workload, the VM with TMemCanal firstly reaches its turning point, but it can also keep the same downward trend with the base line. Figure 6(b) illustrates the performance of a VM with different remote memory capacity under the same workload. It shows a VM almost can kick off all its free memory with low performance penalty.

Figure 6(c) illustrates the performance of a VM with different memory allocation under the same workload. The results show that kicking pages to the swap areas is vital to a VM in which the main memory dedicated to it is exhausted. But, TMemCanal works well for the VMs with more free memory only with an overhead less than 7%.

Through these experiments, we have evaluated the performance impact to the VMs which under-loaded memory is released by TMemCanal. The results show TMemCanal can be used to leverage the memory allocation with a low overhead when a large amount of free memory exists.

B. *Server Consolidation with TMemCanal*

In this group of experiments, we studied a case of server consolidation with TMemCanal. In our scenario, a full-virtualized VM#1 hosting a HPC-LinkPack benchmark [3] and a para-virtualized VM#2 hosting SpecWeb2005 benchmark are consolidated into the Server#2, in which HPC-LinkPack represents the CPU-intensive services. We deployed TMemCanal in Server#2 to eliminate the under-loaded pages of VM#1. Simultaneously, the released pages are immediately reallocated to VM#2 to promote its performance by ballooning. The reason we selected ballooning as a way of under-loaded memory reutilization is that it provides a efficient approach to maximize the performance of under-loaded memory. We can also construct a transparent way to reutilize the under-loaded memory, such as shadow buffer, but it is not the range of this paper.

Through this group of experiments, we study the actual efficiency in a server consolidation environment. Initially, both of VM#1 and VM#2 have been built with 2048MB memory. VM#2 firstly releases 1536MB memory by ballooning.

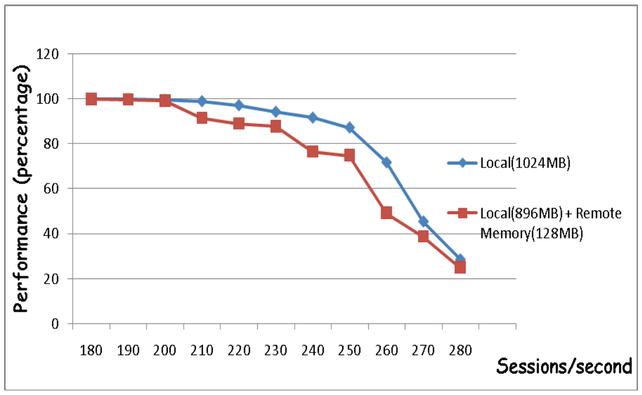
Figure 7 illustrates the performance curves of VM#1 and VM#2. Figure 7(a) presents the performance varying of LinkPack with different released under-loaded memory capacity. From the results, we can observe that using TMemCanal in a CPU-intensive VM nearly hasn't any performance penalty. Figure 7(b) illustrates the performance curve of SpecWeb after inflating its balloon with the memory released by VM#1. The curve shows that the performance of SpecWeb gets high promotion with the inflation of its balloon. Figure 7(c) gives the max sessions of VM#2 with different under-loaded memory reutilization capacity, which can boost its performance by up to 4X.

Our case study shows TMemCanal can help to a global performance promotion via reutilizing the under-loaded memory.

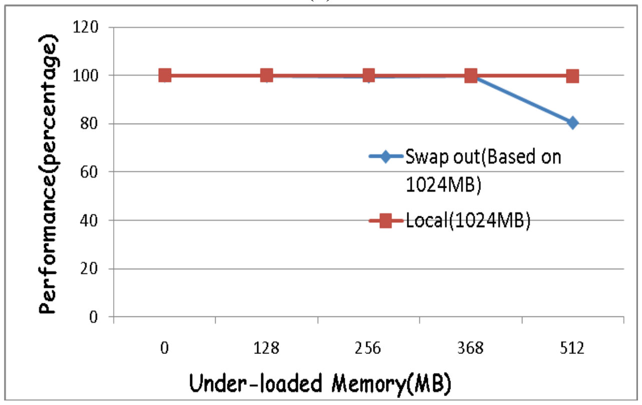
VII. Conclusion

Using TMemCanal, a virtualized distributed system can globally optimize its memory allocation amongst its hosted VMs without any modification to Guest OSs. Through transparent memory canal amongst VMs, TMemCanal is able to leverage memory by a way of memory flowing. Collaborated with reconstructed memory hierarchy, TMemCanal provides a global dynamic memory optimization scheme in cloud platforms. Our evaluations show that TMemCanal can efficiently leverage the memory allocation with low performance penalty to under-loaded memory providers and remarkable performance promotion in a server consolidation case study.

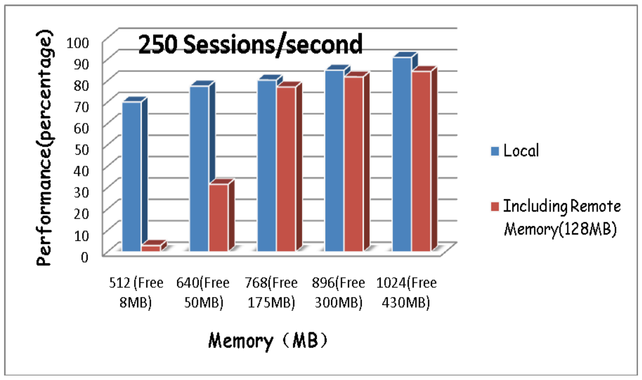
Several future directions focus on applications enjoying such new available under-loaded memory. Thus, we should try to build a global view of distributed under-loaded memory in cloud computing via a light-overhead online



(a)

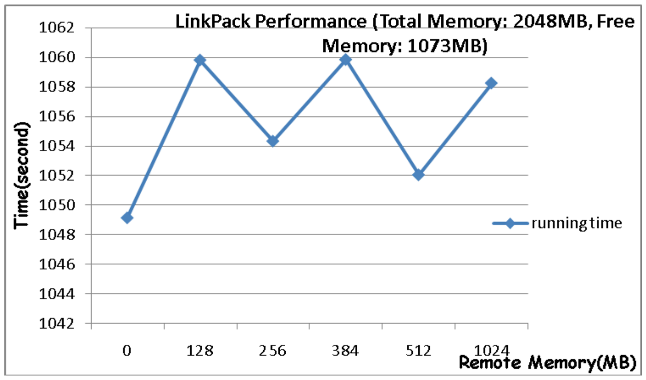


(b)

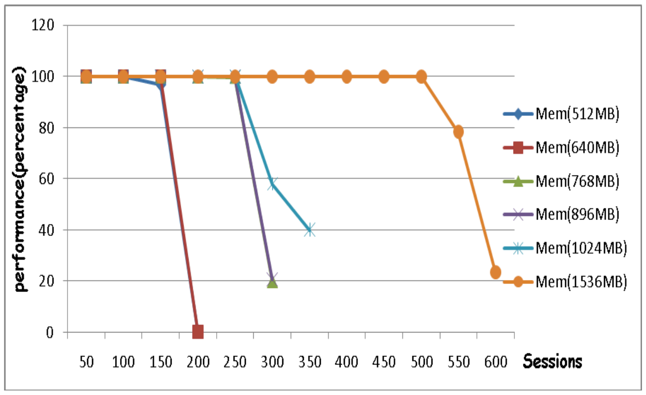


(c)

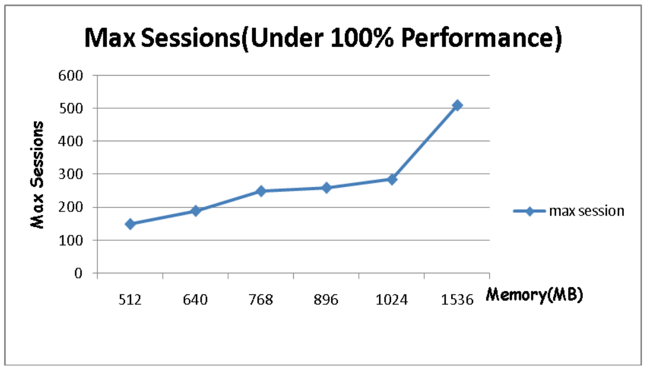
Figure 6. Experiment Results of Performance Impact of TMemCanal. (Larger is better). (a) performance varying of a VM with fixed swapped under-loaded memory; (b) performance varying of a VM with different swapped under-loaded memory; (c)performance impact of VMs with fix-sized swapped memory.



(a)



(b)



(c)

Figure 7. Performance Curve in Server Consolidation Case Study. (a)Performance Curve of LinPack when releasing its under-loaded memory; (b)Performance Curve of Specweb 2005 when inflating its balloon; (c)Max sessions under the situation (b).

efficient detection scheme of under-loaded memory so as to quantify the distribution of under-loaded memory. Finally, a new system software mechanism is needed to build a global virtual space for green cloud computing via greatly increasing utilization per byte/Watt/second of memory.

Acknowledgements

This work was supported in part by the National High-Tech Research and Development Program (863) of China under grants 2007AA01Z119, 2009AA01Z141, 2009AA01Z151, and 2006AA01Z109, and the projects of NSFC under grants 90718040.

Reference

[1] Http://www.xen.org/

[2] Http://www.spec.org/web2005

[3] Http://www.netlib.org/benchmark/hpl/

[4] Michael Hines, Jian Wang and Kartik Gopalan, “Distributed Anemone: Transparent Low-Latency Access to Remote Memory in Commodity Clusters”. Proc. Of International conference on High-Performance computing, 2006

[5] Michael R.Hines, Kartik Gopalan. “MemX: supporting large memory workloads in Xen virtual machines”, Proceedings of the 3rd international workshop on Virtualization technology in distributed computing, Reno, Nevada, 2007.

[6] E. Bugnion, S. Devine, and M. Rosenblum. “Disco: running commodity operating systems on scalable multiprocessors”. In Proceedings of the 16th ACM SOSP, 1997.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. “Xen and the art of virtualization”. In Proceedings of ACM SOSP, 2003.

[8] F. Bellard. “QEMU, a fast and portable dynamic translator”. In Proceedings of the USENIX Annual Technical Conference, 2005.

[9] C. A. Waldspurger. “Memory resource management in VMware ESX server”. In Proceedings of ACM OSDI, 2002.

[10] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. “Difference engine: Harnessing memory redundancy in virtual machines”. In 8th USENIX symposium on Operating System Design and Implementation, 2008.

[11] J. H. Schopp, K. Fraser, and M. J. Silbermann. “Resizing Memory with Balloons and Hotplug”. In Proceedings of Linux Symposium, 2006.

[12] M. Schwidefsky, H. Franke, R. Mansell, H. Raj, D. Osisek, and J. Choi. “Collaborative Memory Management in Hosted Linux Environments”. In Proceedings of the 2006 Ottawa Linux Symposium, 2006.

[13] Matthew Chapman, Gernot Heiser, “vNUMA: A Virtual Shared-Memory Multiprocess”, USENIX’09, Jan.2009

[14] Pin Lu, Kai Shen, “Virtual machine memory access tracing with hypervisor exclusive cache”, USENIX Annual Technology Conference, 2007

[15] Stephen T. Jones, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, “Geiger: monitoring the buffer cache in a virtual machine environment”, ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems. November 2006.

[16] Grzegorz Miło´s, Derek G. Murray, Steven Hand, Michael A. Fetterman, “Satori: Enlightened page sharing”, USENIX Annual Technical Conference, 2009

[17] Project: Transcendent Memory, http://oss.oracle.com/projects/tmem/

[18] Y.Zhou and James F.Philbin, “The Multi-queue Replacement Algoriths for Second Level Buffer Caches”, Proceedings of USENIX Technical Conference, 2001.

[19] Weiming Zhao, Zhenlin Wang, “Dynamic Memory Balancing for Virtual Machines”, VEE’09, 2009

[20] Ying Song, Yanwei Zhang, Yuzhong Sun, Weisong Shi, “Utility Analysis for Internet-Oriented Server Consolidation in VM-Based Data Centers”, IEEE Cluster, 2009

[21] Ying Song, Hui Wang, Yaqiong Li, Binquan Feng, Yuzhong Sun, "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center", 9th IEEE CCGrid, 2009

[22] Ying Song, Yaqiong Li, Hui Wang, Yufang Zhang, Binquan Feng, Hongyong Zang and Yuzhong Sun, “A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing”, HiPC, 2008

[23] D. Thiebaut, H. S. Stone, and J. L. Wolf. Im- proving Disk Cache Hit-Ratios Through Cache Partitioning. IEEE Transactions on Computers, 1992.

[24] RANGAN, K. The Cloud Wars: $100+ billion at stake. Tech. rep., Merrill Lynch, May 2008.

[25] SIEGELE, L. Let It Rise: A Special Report on Corporate IT. The Economist (October 2008).

[26] http://www.microsoft.com/windowsazure

[27] https://s3.amazonaws.com/

[28] http://aws.amazon.com/ec2/

[29] http://code.google.com/appengine/

[30] http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[31] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Shared Singhal, Arif Merchant, “Automated Control of Multiple Virtualized Resources”, 4th ACM Eurosys, 2009

[32] http://hadoop.apache.org/

[33] Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, OSDI, 2004

[34] Kinshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. “Cellular Disco: resource management using virtual clusters on shared-memory multiprocessors”, ACM SOSP, 1999, Kiavah Island, SC