Coordinate Memory Deduplication and Partition for Improving Performance in Cloud Computing

Gangyong Jia, *Member, IEEE,* Guangjie Han, *Member, IEEE,* Joel J.P.C. Rodrigues, *Senior Member, IEEE,* Jaime Lloret, *Senior Member, IEEE,* Wei Li, *Member, IEEE*

***Abstract*—Both limited main memory size and memory interference are considered as the major bottlenecks in virtualization environments. Memory deduplication, detecting pages with same content and being shared into one single copy, reduces memory requirements; memory partition, allocating unique colors for each virtual machine according to page color, reduces memory interference among virtual machines to improve performance. In this paper, we propose a coordinate memory deduplication and partition approach named CMDP to reduce memory requirement and interference simultaneously for improving performance in virtualization. Moreover, CMDP adopts a lightweight page behavior-based memory deduplication approach named BMD to reduce futile page comparison overhead meanwhile to detect page sharing opportunities efficiently. And a virtual machine based memory partition called VMMP is added into CMDP to reduce interference among virtual machines. According to page color, VMMP allocates unique page colors to applications, virtual machines and hypervisor. The experimental results show that CMDP can efficiently improve performance (by about 15.8%) meanwhile accommodate more virtual machines concurrently.**

***Index Terms*—Main memory, memory deduplication, memory partition, virtualization, performance.**

I. INTRODUCTION

WITH the ability to scale computing resources on demand and provide a simple pay-as-you-go business model for customers, cloud computing is emerging as an economical computing paradigm, and has gained much popularity in the industry [1]. Currently, a number of big companies such as Netflix and Foursquare [2] have successfully moved their business services from the dedicated computing infrastructure to Amazon Elastic Computing Cloud (EC2) [3]. Undoubtedly, more customers and enterprises will leverage the cloud to maintain or scale up their business while cutting down the budget, as reported by the International Data Corporation (IDC) that the business revenue brought by cloud computing will reach $1.1 trillion by 2015 [4].

In cloud computing, multiple virtual machines (VMs) can be collocated on a single physical server, and they can operate independently with virtualization technology [5, 6], which provides flexible allocation, migration of services, and better security isolation. In the virtualization environment, physical resources (such as main memory) are managed by a software layer called hypervisor (or Virtual Machine Monitor, VMM), and the primary goal of a hypervisor is to provide efficient resource sharing among multiple co-running virtual machines [7].

However, with the number of VMs keep increasing on one physical server (it will be up to 8 VMs on one physical core in desktop cloud environment), meanwhile the interference among different VMs is more and more serious, virtualization has placed heavy pressure on memory system for both larger capacity and better independence. The demand for memory capacity is much advance to the increasing speed, therefore, both memory size and reducing interference are two of the major bottlenecks to improve performance of the whole server. Memory deduplication detects and reduces page duplication to alleviate the memory demands; memory partition divides memory resource among threads/VMs to reduce interference for improving performance. Both techniques have demonstrated great opportunities in improving memory performance respectively.

Kernel Samepage Merging (KSM) [8], which is the implementation of memory deduplication and adopted by Linux kernel, is running transparently in the hypervisor layer and requires none modification to guest operating systems. KSM scans memory pages of guest VMs periodically to detect all identical pages of the same content. All guest VMs which have the identical pages share one single physical page, therefore, all the other redundant physical pages can be reclaimed to the hypervisor. In this way, KSM can efficiently reduce the memory demands of the VMs. Difference Engine [9] reported memory deduplication can save memory 50% across VMs absolutely, and VMware [10] reported about 40% memory savings. Although memory deduplication can save memory, it exists two problems. The first one is that the overhead of page comparing is too costly to use. The second one is that memory deduplication can't solve the memory contention problem. In the virtualization, VMs running concurrently contend with each other for the shared memory, especially memory. Hence, VM can be slowed down compared to when it runs alone and

This work was supported by “Qing Lan Project”, “the National Natural Science Foundation of China under Grant 61572172, 61401147 and 61572164” and “the Natural Science Foundation of Jiangsu Province of China, No.BK20131137 and No.BK20140248”, “Zhejiang provincial Natural Science Foundation No. LQ14F020011 and No. LQ12F02003”.

G. Jia is now with the Department of Information & Communication Systems, Hohai University, Changzhou, China and the Department of Computer Science, Hangzhou Dianzi University, Hangzhou, 310018, China (Email: gangyong@hdu.edu.cn)

G. Han is now with the Department of Information & Communication Systems, Hohai University, Changzhou, China (Email: hanguangjie@gmail.com)

J. J.P.C. Rodrigues is with Institute of Telecommunications, University of Beira Interior, Covilha, Portugal and Department of Informatics of University ITMO, St. Petersburg, Russiathe (email: joeljr@ieee.org)

J. Lloret is now with the Integrated Management Coastal Research Institute, Universidad Politecnica de Valencia, Valencia, Spain (Email: jlloret@dcom.upv.es)

W. Li is now with the Department of Information & Communication Systems, Hohai University, Changzhou, China (Email: liweihhu@gmail.com)

entirely owns the memory system, which causes system unfairness and overall system performance degradation. Moreover, memory streams of different VMs are interleaved and interfere with each other at DRAM memory, which destroys original spatial locality and bank level parallelism of individual VMs, thus severely degrading system performance [11-14]. As the both number of cores and VMs in the server continue to grow, the contention and interference will be more serious [15].

Memory partition, containing channels/rank/bank partition, divides memory resource among VMs/threads to reduce interference. MCP [16], map memory intensive and non-intensive threads onto different channel(s), speeds up non-intensive threads for preventing interference from memory intensive threads. However, system unfairness is more serious for physically exacerbating intensive threads contention. Bank partition [11, 13, 14], map cores onto different memory banks,isolates memory access streams of different threads to eliminate interference. Although memory partition solves the interference issue, the improvement of system throughput and fairness is limited.

In this paper, we propose a coordinate memory deduplication and partition approach named CMDP to reduce memory requirement and interference simultaneously for improving performance in virtualization. CMDP contains memory deduplication and memory partition techniques simultaneously to play their respective advantages and avoid shortcomings. Therefore, CMDP can improve system performance and reduce interference simultaneously. Moreover, a virtual machine based memory partition called VMMP is added into CMDP to reduce interference among virtual machines. VMMP dynamically maps hypervisor, VMs and applications running on VMs onto different memory banks. Hypervisor, different VMs and different applications running on different VMs use independent memory banks to eliminate interference. Also, CMDP adopts a lightweight page behavior-based memory deduplication approach named BMD to reduce futile page comparison overhead meanwhile to detect page sharing opportunities efficiently. In BMD, pages are grouped into different classifications based on both page behavior and belonged banks. Pages belonged to memory banks of VMs are more possibility with the same content, moreover, pages with similar behavior, especially access behavior, are suggested to have higher possibility with same content. So pages belonged to memory banks of VMs and with similar behavior are grouped into the same classification. Therefore, performing page comparisons is restricted into the same classification, never exceeding to different classifications, which will do many futile comparisons. In this way, BMD can reduce overhead of futile comparisons to make our CMDP be more efficient.

In summary, the paper aims to make the following contributions through the proposal of CMDP:

(1) Coordinate memory deduplication and partition to improve performance and reduce interference simultaneously.
(2) CMDP adopts VMMP, which maps hypervisor, VMs and applications running on VMs onto different memory banks. In this way, hypervisor, different VMs and different applications running on different VMs use independent memory banks to eliminate interference.

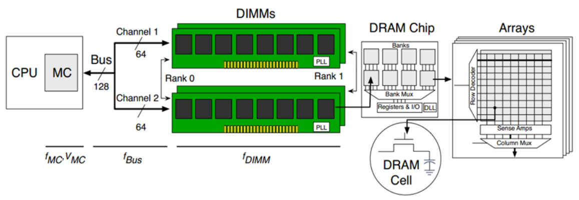


Fig. 1. organization of a modern memory subsystem

(3) Because pages belonged to memory banks of VMs are more possibility with the same content, moreover, pages with similar behavior, especially access behavior, are suggested to have higher possibility with same content, so pages belonged to memory banks of VMs and with similar behavior are grouped into the same classification for reducing comparison range. In our proposed BMD, page comparisons is restricted into the same classification to reduce overhead of futile comparisons.

The rest of this paper is organized as follows. Section 2 elaborates on essential background and research motivations. Section 3 explains our coordinating memory deduplication and partition platform. Section 4 describes experimental methodology and section 5 presents the results of our experiments. We discuss the related work in section 6. Finally, section 7 concludes this paper.

II. BACKGROUND & MOTIVATION

We provide a brief overview of modern memory subsystems, then profile interference and memory sharing, finally analyze the comparison overhead of KSM.

A. DRAM Organization

Figure 1 illustrates the multiple levels of organization of the memory subsystem. To service memory accesses, the memory controller (MC) sends commands to the DIMMs on behalf of the CPU's last-level cache across a memory bus [29]. As shown, recent processors have integrated the MC into the same package as the CPU. To enable greater parallelism, the width of the memory bus is split into multiple channels. These channels act independently and can access disjoint regions of the physical address space in parallel [17].

Multiple DIMMs may be connected to the same channel. Each DIMM comprises a printed circuit board with register devices, a Phase Lock Loop device, and multiple DRAM chips. The DRAM chips are the ultimate destination of the MC commands. The subset of DRAM chips that participate in each access is called a rank. The number of chips in a rank depends on how many bits each chip produces/consumes at a time. Each DIMM can have up to 16 chips, organized into 1-4 ranks.

Each DRAM chip contains multiple banks (typically 8 banks nowadays), each of which contains multiple two-dimensional memory arrays. The basic unit of storage in an array is a simple capacitor representing a bitłthe DRAM cell. Thus, in a x8 DRAM chip, each bank has 8 arrays, each of which produces/consumes one bit at a time. However,

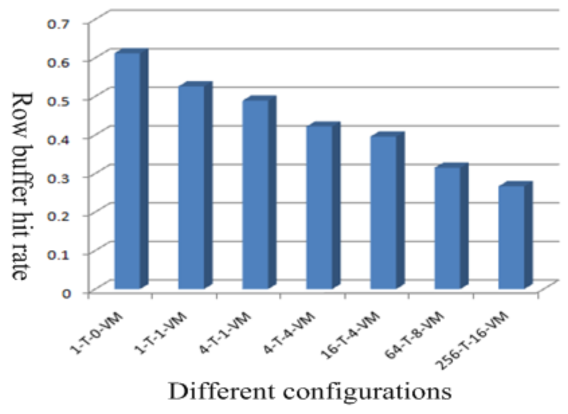


Fig. 2. row buffer hit rate under different running configurations

each time an array is accessed, an entire multi-KB row is transferred to a row buffer. This operation is called an "activation" or a "row opening". Then, any column of the row can be read/written over the channel in one burst. Because the activation is destructive, the corresponding row eventually needs to be "pre-charged", that is, written back to the array.

B. Profiling of Interference

With the core number and VM number keep increasing in the server, one of major challenges on memory system is interference. Usually a single thread's memory requests have good locality and can exhibit good row buffer hit rate. But the locality is significantly reduced in a multi-core server with running many VMs parallel. Therefore, row buffer hit rate decreases sharply, leading to poor overall system performance. Figure 2 demonstrates that the row buffer hit rate decreases significantly with the thread number and VM number increased. The y-axis shows the row buffer hit rate, and the x-axis presents the different running configurations. The n-T-m-VM represents there is m VMs with n threads in the server, and every VM has n/m threads.

The interference mainly derives from three aspects:

1) **Hypervisor interfere VMs.** The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources. The hypervisor takes responsible for the resources management. Therefore, VMs need to request the hypervisor to allocate physical resources, like the memory resource, during their lifetime. In the process of memory allocation, hypervisor may disturb the origin memory access stream of VMs, which is one aspect of the memory interference.
   Above situation has been shown in the figure 2. 1-T which represents one thread running on operating system without hypervisor is better than 1-T-1-VM which represents one thread running on one VM with hypervisor in row buffer hit rate. The advantage is mainly from the hypervisor interference-free.
   Moreover, in order to detailed analyze the interference effect from hypervisor, we count the proportion caused by hypervisor to the VM row buffer misses. Figure 3 demonstrates the VM row buffer misses proportion caused by hypervisor in different configurations. The x-axis presents the different benchmarks from PARSEC running on VM. This figure has proven hypervisor contributes great row buffer misses.
2) **VM interfere applications running on it.**In order to get the service of the guest operating system (OS), applications need to invoke system calls frequently during their lifetime. To access guest OS's address space, applications have to switch to the kernel state. After the service finished, the state returns back to the user state, which is the address space of applications originally. For a simple system call, kernel only uses a small part of a page, while it has to complete the above steps, which may lead to two additional row-buffer misses. Although guest OS invocations are usually short-lived, they are invoked frequently [18], which leads to the frequently switches between kernel state and user, intensifying interference. Figure 4 demonstrates row buffer misses proportion caused by guest OS to different benchmarks. In the figure, we run different benchmarks on VM to count the misses proportion caused by invoking system calls and other guest OS interference. This figure has clearly shown guest OS contributes most row buffer misses to applications.
3) **Interference among VMs and threads on one VM.**VMs and threads on one VM running concurrently contend shared memory in both CMP systems and virtual CMP (VCMP) systems. Therefore, memory streams of different VMs and threads are interleaved and interfere with each other at DRAM memory and virtual memory address space respectively. The results of figure 2 have proven the interference among VMs and threads on one VM. 1-T-1-VM which represents one thread running on one VM is better than 4-T-4-VM which represents four threads running on four VMs and each VM has one thread in row buffer hit rate. Briefly, one VM is better than four VMs simultaneously running in row buffer miss rate. Moreover, as the threads number of one VM increased, such as 1-T-1-VM, 4-T-1-VM, 64-T-8-VM and 256-T-16-VM, from one thread to 16 threads on one VM, the row buffer miss rate decreases seriously. Interference among VMs and threads on one VM needs to alleviate for memory performance improvement.

C. Profiling of Memory Sharing

Basically, memory deduplication of reducing memory requirements is based on the assumption that a system has many identical contents. However, in the virtualization, a physical server mostly hosts multiple VMs to run simultaneously for different services. The software as well as the data used in VMs can be similar [19]. Therefore, through merging those identical pages, the physical system can release additional free pages.

Moreover, the guest OS running on each VM also have many identical contents. Figure 5 demonstrates the identical pages proportion between two VMs without applications running on them. In the figure, the x-axis presents the two guest

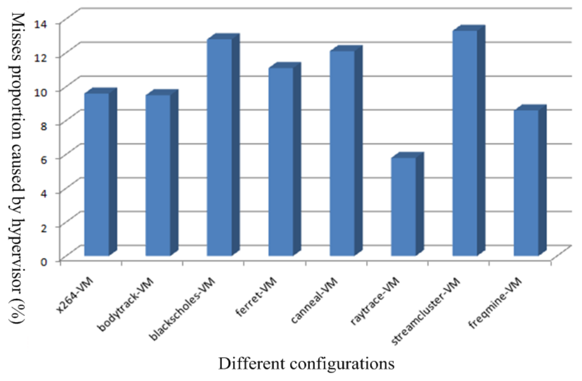


Fig. 3. VM row buffer misses proportion caused by hypervisor in different configurations

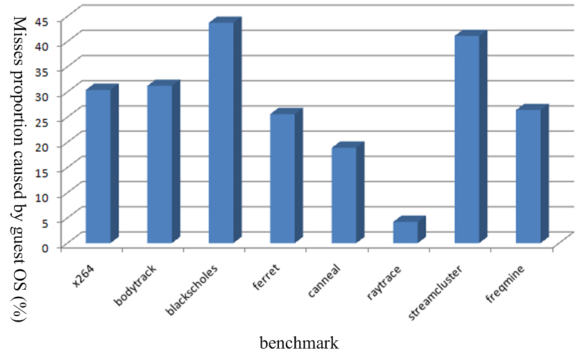


Fig. 4. row buffer misses proportion caused by guest OS

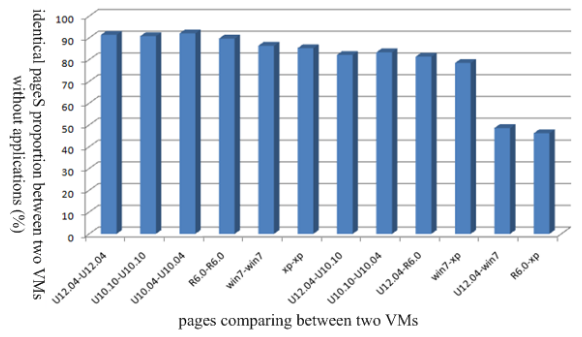


Fig. 5. identical pages proportion between two VMs without applications

OS which compares the identical pages, and U12.04 presents Ubuntu 12.04 version, R6.0 presents Redhat Linux 6.0 version, win7 presents windows 7 version, xp presents windows xp version. The results of the figure 5 show two VMs of the same guest OS and same version have a large proportion of the identical pages, even more than 90%. Two VMs of the same guest OS but with different versions also have a good proportion of the identical pages, more than 75%. Although two VMs of the different guest OS have not as many identical pages as above situations, more than 40%, each server can only one kind of guest OS for service. Therefore, there is good opportunity to alleviate memory request using memory deduplication.

D. *Comparison Overhead Analysis of KSM*

KSM is the implementation of Content Based Page Sharing (CBPS) using in the Linux kernel, which is base on scanning. KSM not only targets kernel virtual machines (KVMs) but also processes running on the host Linux kernel. This scheme uses two red-black trees to detect identical pages, one is named stable tree and the other is named unstable tree. Stable tree is used for recording shared pages, and unstable tree is used for recording single used pages. In each scan round, a candidate page is firstly compared with pages in the stable tree. If there is a match, the candidate page will be merged and shared times plus one. Otherwise, compare with the unstable tree: If there is a match, the single used page is changed to shared and inserted into stable tree; if there is no match in the unstable tree, the candidate page is inserted into the unstable tree.

As the increasing capacity of main memory, the size of these two global trees expands proportionally. One candidate page needs to be compared content with a large number of pages, but those pages have low possibility to share with the candidate page. For example, pages of hypervisor have low possibility to share with VM and applications. In order to reduce useless comparisons, we can take sharing possibility into account. For a candidate page, it only compares with pages of high possibility pages sharing with it. The figure 5 has shown VMs of the same guest OS have high possibility to share pages. Therefore, only candidate pages of VMs need to be compared with pages belonged to VMs, which can reduce much useless comparisons.

Moreover, identical pages always have the same behavior, especially access behavior. One page is used for storing instructions is usually for reading, never for writing. Otherwise, the data page contains a large proportion of writing. So, different access behavior pages are hardly identical. Before comparing page contents, check whether they have different behaviors, give up comparing if their behaviors are different for impossible identical. In this way, the comparisons are reduced further.

III. COORDINATE MEMORY DEDUPLICATION AND PARTITION (CMDP)

In this section, we firstly introduce the overview of coordinate memory deduplication and partition (CMDP) in subsection 3.1. Then we introduce a virtual machine based memory partition called VMMP to allocate unique page colors to applications, virtual machines and hypervisor in subsection 3.2. Finally, we introduce a lightweight page behavior-based memory deduplication approach named BMD to reduce futile page comparison overhead in subsection 3.3.

A. *Overview of CMDP*

To reduce memory requests and interferences simultaneously, we propose coordinate memory deduplication and partition (CMDP). It contains two parts of the CMDP. First, to reduce interferences among VMs, VMMP of the CMDP dynamically maps hypervisor, VMs and applications running on VMs onto different memory banks instead of accessing all memory banks. Through isolating their memory requests

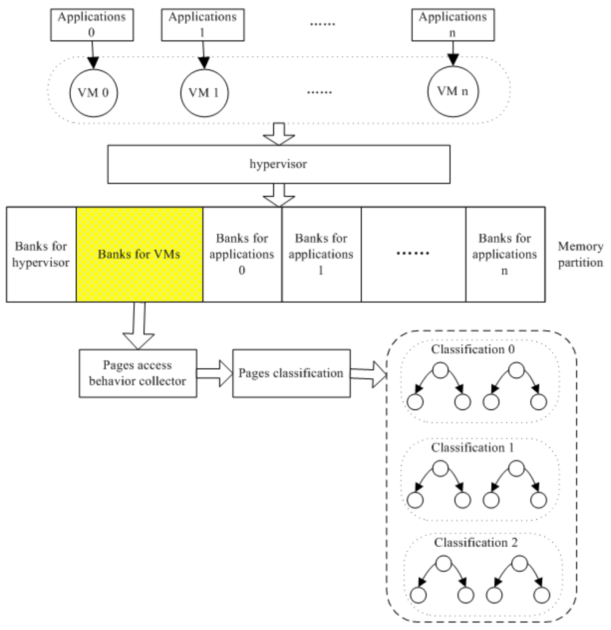


Fig. 6. the CMDP framework containing memory partition and memory deduplication

from others, the spatial locality can be reserved. Second, to reduce memory requests, BMD of the CMDP groups pages into different classifications based on both belonged banks and page accessed behavior. Pages belonged to memory banks of VMs are more possibility with the same content, moreover, pages with similar behavior, especially access behavior, are suggested to have higher possibility with same content. So pages belonged to memory banks of VMs and with similar behavior are grouped into the same classification. Therefore, performing page comparisons is restricted into the same classification, never exceeding to different classifications, which will do many futile comparisons.

Figure 6 demonstrates our CMDP, which contains memory partition and memory deduplication simultaneously. In CMDP, we implement memory partition based on our proposed VMMP, which partitions memory banks into group, each group is mapped to hypervisor, VMs or applications of one VM. In this way, hypervisor, VMs and applications access unique memory banks to prevent interference with each other; similarly, we implement memory deduplication based on our BMD, which restricts page sharing into banks for VMs, because the identical pages mainly in the this memory domain. Moreover, BMD adds the page access behavior to reduce comparisons, which partition pages into different classifications according to their behavior and compare page content within each classification to further reduce comparisons.

B. Virtual Machine based Memory Partition (VMMP)

To partition memory banks, we need to understand the memory request of each part. The memory request is mainly defined by three components: memory intensity, row buffer locality, and bank level parallelism [14].

Memory intensity. Memory intensity is the frequency of an application generating memory requests.

Row buffer locality. In current DRAM memory systems, each bank contains a row buffer for providing temporary data storage of a DRAM row. If continuous accesses in the same row, which called row-buffer hit, the only column accesses need to occur; otherwise, row-buffer conflict occurs, then memory controller precharges the opened row, activate another row, finally to perform column access.

Bank level parallelism. Memory bank is a set of independent memory arrays inside a DRAM device, which can be accessed parallel. In modern out-of-order processors, due to the high latency of off-chip DRAM, more and more memory accesses need to be performed parallel to improve system performance. If threads generate multiple memory accesses to different banks, they can proceed in parallel in memory system.

According to the three components of each application, allocate each application the suitable memory banks to reduce interference while improving performance. Applications of low memory intensity are no performance changes with the bank amount. Memory intensive applications are sensitive to the number of banks, but not more banks better performance. The request of the bank number is limited [14].

In this paper, we assume all VMs have the same guest OS and each VM provide one single service in one server. In cloud computing center, there are many servers. If one server hosts VMs of the same guest OS, which will increase the possibility of sharing pages. Nowadays, each VM is used for special service, so one VM provides one single service normally. Threads for one service are usually sharing address space.

Based on above hypothesis and the memory requests of applications, we partition memory banks into bank groups. Hypervisor is allocated one bank group, which reduces interference to the VMs; all VMs of the same guest OS is allocated another one bank group, which can reduce interference to the applications running on VMs and restrict the share space to reduce comparisons; each applications of one VM is allocated one bank group, which can reduce interference among applications from different VMs. Moreover, each bank group has different bank number. The bank number of bank groups for applications running on VM is mostly 16 banks, because all thread of the application running on one VM is mostly sharing address space and 16 banks is enough at most times, moreover, the total banks of the memory is limited. Similarly, the bank number of bank groups for hypervisor is mostly 16 banks too. The other banks are for VMs. Algorithm 1 shows the pseudo code of our VMMP.

In the server, there will be too many VMs to have enough memory banks for allocating. Above partition is used for parallel running VMs, VMs scheduled in turn can share one memory group.

C. Page Behavior-based Memory Deduplication (BMD)

Since the KSM simply maintains two global comparison trees for all memory pages of a hosting server. To detect page sharing opportunities, each candidate page needs to be compared with a large number of uncorrelated pages in the global trees repeatedly, which will induce massive futile comparisons [7]. The key innovation to reduce futile comparison

Algorithm 1 our VMMP algorithm

```
Definition:
N: the total cores in the server
M: the total memory banks in the server
AVG: AVG=M/(N+2)
RM: RM=(M-32)/N
VMMP:
If AVG <16
    Allocate 16 banks for hypervisor;
    Allocate 16 banks for VMs;
    Each two applications of one VMs share RM banks;
Else
    Allocate 16 banks for each applications of one VMs;
    Allocate 16 banks for hypervisor;
    Allocate remainder banks for VMs;
```

is to reduce the comparison memory domain and break the comparison trees into multiple small trees simultaneously. The most possibility to have same content is the different VMs. In the VMMP, we allocate one memory bank group for all VMs. Therefore, we only need to compare page content within the memory domain for all VMs called comparison domain to reduce futile comparison.

Moreover, pages within the comparison domain are grouped into multiple classifications, with dedicated local comparison tree in each page classification. A candidate page which is belonged to the comparison domain, needs only to be compared with pages in its local comparison tree of its classification, which contains less page nodes. But the pages in its local tree are having much higher probability to have same content with the candidate page, thus it can reduce futile comparisons meanwhile detect page sharing opportunities efficiently.

In order to partition pages into different classifications to reduce comparisons, the page classification approach needs to consider below problems: 1) pages with high probability to have the same content should be partitioned into the same classification, which can detect page sharing in the local tree. 2) pages with low probability to be shared should be partitioned into different classification, which prevents futile comparisons occurring in the local tree. 3) the overhead of the page classification needs to be low.

The lower half of the figure 6 has shown our BMD, which contains a memory access behavior collector and a page classification manager. The memory access behavior collector captures the access behavior of all pages within comparison domain. And the page classification manager groups pages within comparison domain into different classifications based on page access behavior, pages with similar access behavior are grouped into the same classification. The page classification are performed in each scan round, which means that the access behavior of pages captured during the last scan round are used to guide page classification in this scan round. And the memory access collector continues to capture access behavior of pages during this scan round, which will be used in the next scan round.

Page access behavior collector. In this paper, in order to reduce the overhead of collecting page access behavior, we capture page access behavior within comparison domain based on the page table. The page table is mainly used to transfer the virtual address into physical address, and at the same time, it will show the access behavior of the physical page. Every entry in the page table shows the information of the corresponding physical page, containing the read, modify and so on of the physical page.

Page classification. In this work, we use the read and modify information in the entry of page table. We partition all pages within comparison domain into 4 classifications: the first classification is not read and not modified; the second classification is read but not modified; the third classification is not read but modified; the last classification is read and also modified. In this way, we neither modifying the hardware nor collecting additional information. It is easy to realize and the overhead is low.

IV. EXPERIMENTAL SETUP

We carried out our experiments with two 2.00GHz Intel Xeon E5504 processors with EPT enabled. Each E5504 processor has 4 physical cores and we have disabled the Hyper-Thread. There are 3-level caches in each processor, the L1 instruction and data caches are 32KB each and the L2 cache is 256KB, both the L1 and L2 are private in each core. The L3 cache is 16-way 4MB and shared by all four cores in each processor. The cache block size is 64-Byte for all caches in the hierarchy. The total capacity of physical memory is 8GB with one dual-ranked of DDR3-800MHz. The host server runs Ubuntu-12.04 with Linux kernel 3.6.0. We implement CMDP based on KSM of Linux 3.6.10. We use QEMU [20] with KVM [21] (qemu-kvm-1.2.0) to support guest VMs. Each guest VM is configured with 1 virtual CPU, we boot 4 VMs in parallel as our default configuration. We also boot 8VMs in parallel for further evaluation. The guest VMs are running 64-bit Linux-10.10 with Linux kernel 2.6.32. We choose to run the following workloads inside guest VMs:

Kernel Build: we compile the Linux kernel 3.6.10 in guest VMs. We begin this benchmark after the VMs are fully booted and static sharing opportunities are detected.

Apache Server: we run the ab [22] benchmark on Apache httpd server. We test a local web site in guest VMs with 24 of concurrency requests.

MySQL Database: we run the SysBench [23] with MySQL database in guest VMs. We test database with 1-thread and the oltp-table-size is configured as 1500000.

V. EXPERIMENTAL RESULTS

We first evaluate the impact of our proposed CMDP on system performance. Then analyze the effect of our CMMP on interference reduction. Finally, evaluate the advantages of our BMD in reducing both memory request and futile comparisons.

A. System Performance of CMDP

We define system performance using system throughput, which measures by weighted speedup, which is shown in the

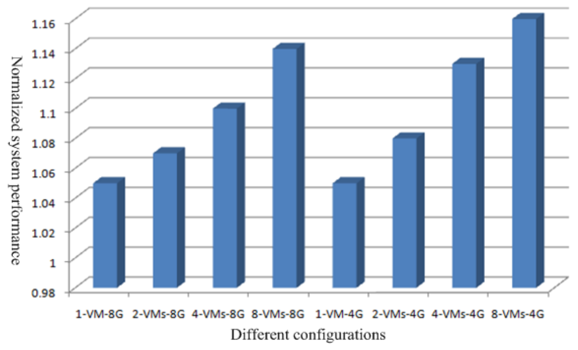


Fig. 7. the normalized performance improvement in different configurations

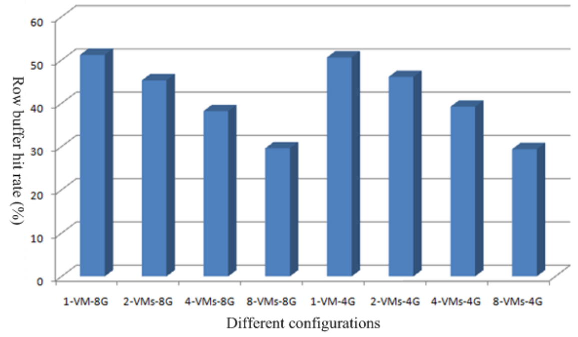


Fig. 8. the row buffer hit rate in different configurations of the default system

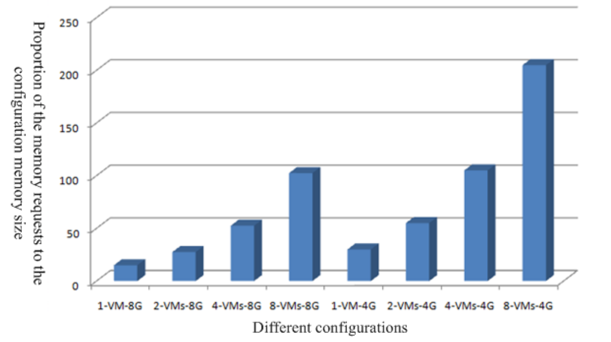


Fig. 9. the proportion of the memory requests to the configuration memory size in different configurations of the default system

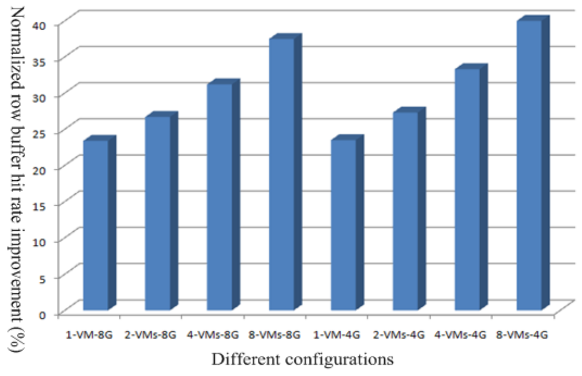


Fig. 10. the improved row buffer hit rate of our VMMP normalized to default system in different configurations

equation (1). The IPC$_i$ represents the IPC of VM$_i$.

$$weighted\_speedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} \quad (1)$$

Figure 7 demonstrates the normalized performance improvement in different configurations with our CMDP, from booting one virtual machine to 8 virtual machines with 4G or 8G memory. The x-axis presents the different configurations, mainly containing the parallel running VMs number and the total physical memory size. The n-VMs-mG represents that n VMs parallel running on the m G memory size server.

The results of the figure show the more VMs parallel running on less configured memory size the better system performance of our CMDP will realize. This is because the more VMs parallel running on less configured memory size makes both more interference and more memory requests, which give better chances to bring our CMDP into play.

Figure 8 shows the row buffer hit rate in different configurations of the default system, from booting one virtual machine to 8 virtual machines with 4G or 8G memory. The results of this figure have proven the more interference along with booting more VMs in one server, from one to eight. The row buffer hit rate decreases serious. This phenomenon is the same in both configured 8G memory size and 4G memory size for the server.

Figure 9 shows the proportion of the memory requests to the configuration memory size in different configurations of the default system, from booting one virtual machine to 8 virtual machines with 4G or 8G memory. The higher proportion is, the memory request contend is more serious, which means the more urgent to use memory deduplication to share the same content pages to reduce memory request. Briefly, the higher proportion is, the better chances for our BMD to reduce memory request. The results of this figure also have proven the more interference along with booting more VMs in one server, from one to eight. Moreover, the server with 4G memory size demonstrates higher proportion than the configuration with 8G memory size, which is the same with our original thinking. In this experiment, each VM is configured with 1GB main memory.

B. Interference Reduction of VMMP

VMMP dynamically maps hypervisor, VMs and applications running on VMs onto different memory banks instead of accessing all memory banks. Through isolating their memory requests from others, the spatial locality can be reserved.

Figure 10 demonstrates the improved row buffer hit rate of our VMMP normalized to default system in different configurations. The results have shown the more interference, the better improvement our VMMP will realize. The improved row buffer hit rate of our VMMP is mainly from the reduction interference from hypervisor to VMs, from VM to applications running on it, and among applications of different VMs.

Reduce interference from hypervisor to VMs. In the section 2.2, we have elaborated one of the interference is

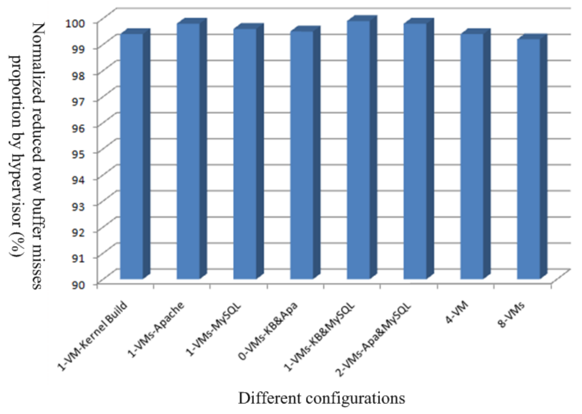


Fig. 11. the reduced row buffer missed proportion by hypervisor of our VMMP normalized to default system

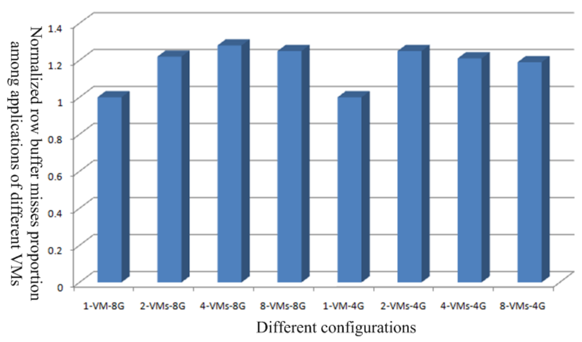


Fig. 12. the reduced row buffer missed proportion among applications running on different VMs of our VMMP normalized to default system

from hypervisor. In our VMMP, we allocate one memory bank group for the hypervisor to isolate it from VMs, which can reduce the interference from hypervisor to VMs.

Figure 11 demonstrates the reduced row buffer missed proportion by hypervisor of our VMMP normalized to default system. The results show in all configurations the reduced buffer misses proportion by hypervisor is more than 99%, which means hypervisor hardly interferes VMs. Therefore, our VMMP almost can prevent the interference from hypervisor to VMs.

Reduce interference from VM to applications running on it. Similarly with the reduction interference from hypervisor to VMs, the interference from VM to applications running on it is almost prevented. Our experimental results show in all configurations of running different applications on VM the reduced buffer misses proportion by VM is more than 98%, which means VM hardly interferes applications running on it. Therefore, our VMMP almost can prevent the interference from VM to applications running on it.

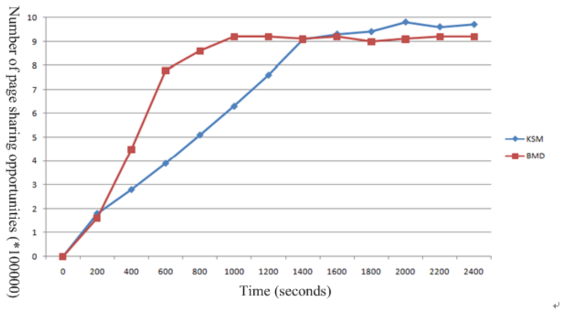
Reduce interference among applications of different VMs. In our VMMP, we allocate one memory bank group for each applications running on one VM to prevent interference among applications of different VMs. Figure 12 demonstrates the reduced row buffer missed proportion among applications running on different VMs of our VMMP normalized to default system. In the figure, we can see the 8GB memory size configuration is better than 4GB memory size configuration in general. This is because the 8GB memory size has more memory banks for partition than 4GB memory size. In the 8GB memory size configuration, the best effect is configured 4 VMs parallel running, which is because when more VMs running parallel there is no enough banks for each applications running on one VM, two need to share one memory bank group. So, the effect is not as well as 4-VMs. Similar results can get in 4GB memory configuration.

C. Both Memory Request and Futile Comparisons Reduction of BMD

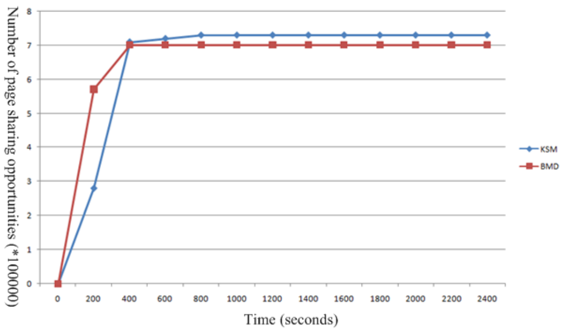
Figure 13 shows the page sharing opportunities of different workloads with 4VMs. For Kernel Build workload in Figure 13(a), we can see that the KSM can detect the most page sharing opportunities. But since it maintains pages into large global comparison trees, candidate pages needs to be compared with a large number of uncorrelated page nodes, thus it takes a little longer time to reach its maximum page-sharing state. On the other hand, our BMD is about 93.8% of the KSM running kernel build in detecting page sharing opportunities. This is mainly because our BMD restricts the comparison within the comparison domain, which is the memory bank group for VMs. Other memory domain is not to compare. And there is one other reason for detecting not all page sharing opportunities: all pages are partitioned into different classifications according to the access behavior, and this partition may reduce the opportunities. Some not read and not write pages may have the identical content with the pages of being read and not write. Similar results have shown in figure 13(b) and 13(c) for Apache and MySQL respectively. Our BMD detects page sharing opportunities about 94.3% of the KSM running Appache. And our BMD detects page sharing opportunities about 91.2% of the KSM running MySQL.

Figure 14 shows the number of pages comparisons of different workloads with 4 VMs. For Kernel Build workload as shown in figure 14(a), we can see that the KSM with large global comparison trees induces the most number of page comparisons. While our BMD has the less number of page comparisons, it is about 31.2% of the KSM, since our BMD both restricts comparison within comparison domain and local classification. For the restricting comparison domain, BMD restricts comparison within 25% of the whole memory, the comparison domain only occupies the 25% of the whole memory. For the restricting local classification, BMD classifies the single tree of the KSM into four trees based on the behavior of both read and modify, so the comparison is reduced to 25% on average. Similarly, for Appache workload as shown in figure 14(b), it is about 30.6% of the KSM. And for MySQL workload as shown in figure 14(c), it is about 35.4% of the KSM.

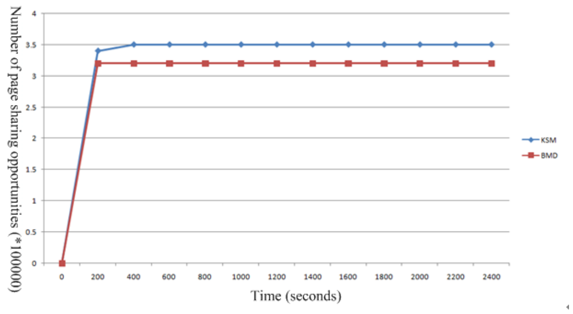
Figure 15 shows the percentage of futile rate reduction with 4 VMs, where the baseline is with the KSM approach. We can see that our BMD can reduce futile rate 18.7% on average. These futile comparisons are mainly both in the domain exceed the comparison domain and local classification, so they can hardly be detected the same content pages. Therefore, its well



(a) Kernel build

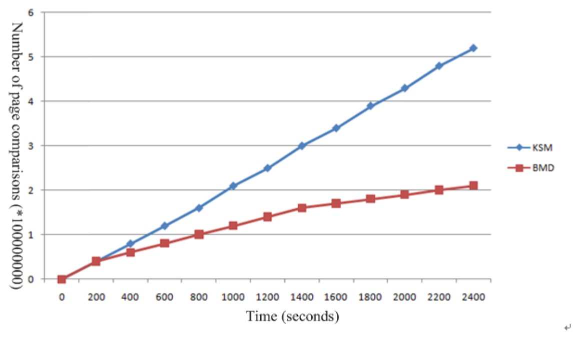


(b) Apache

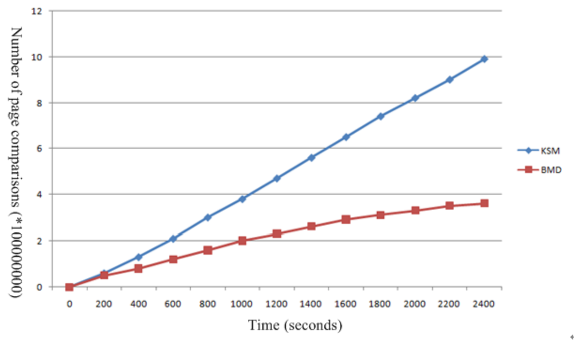


(c) MySQL

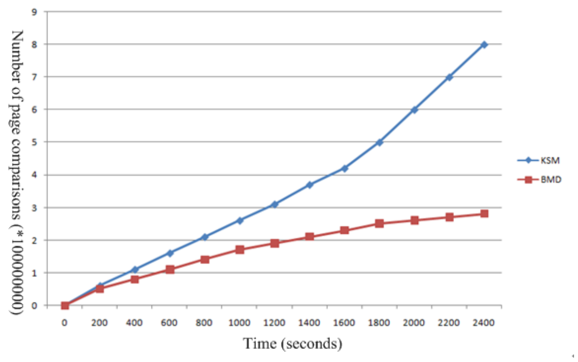
Fig. 13. the page sharing opportunities with 4 VMs



(a) Kernel build



(b) Apache



(c) MySQL

Fig. 14. the number of page comparisons of different workloads with 4VMs

to abandon them.

D. Overhead Analysis

Software Support. There are two parts which requires system software support, the first one is the VMMP, and the other one is the BMD. In the VMMP, based on the three components of each application, memory intensity, row buffer locality, and bank level parallelism, allocate each application corresponding memory banks. Therefore, VMMP needs to collect parameters of these three components. Then, categorize applications into different classifications, low memory intensity classification or memory intensive classification. Finally, based on the partitioned memory bank groups, allocate memory group for each application classification.

In the BMD, in order to reduce memory requirements, we need to detect content of pages to determine whether pages can be shared. But our BMD reduces futile comparison through reducing the comparison memory domain and breaking the comparison trees into multiple small trees simultaneously. So, we add the page access behavior collector which is based on reading the page table and page classification which partitions

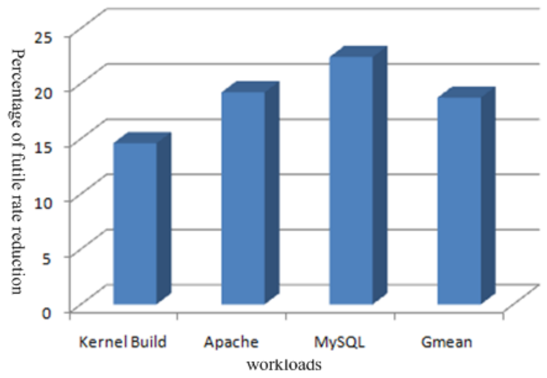


Fig. 15. the percentage of futile rate reduction with 4 VMs, where the baseline is with the KSM approach.

pages into 4 classifications behaving the same status.

Performance Overhead. In order to evaluate the performance overhead after adopting our CMDP, we compare our VMMP with the default method, and the experimental results show the performance of our VMMP is negligible, below 1%; also we compare our BMD with the default method without KSM, although BMD decreases 8% on average, increase the memory sharing by more than 30%. We compare the performance overhead between BMD with KSM in section 5.C. All above results have shown our CMDP behave well.

VI. RELATED WORK

There are a number of related studies to both memory partition and memory deduplication.

Memory partition. Memory partition contains cache partition, channel partition and bank partition.

Cache partition. Either hardware based cache partition [24] or software page coloring based cache partition [25] are employed to partition shared cache to concurrent running threads, which can eliminate the interference between multi-threads and hence reduce conflict at cache level. However, other resources such as MC, memory bus, and DRAM are also shared and confronted with contention and interference [31].

Channel partition. Data of different threads are mapped into different channels according to their memory access behavior in [16, 32], which can eliminate the interference between threads at channel level. However, channel partition cannot be applied to system with cache line interleaving policy between channels [16], which limit its applicable scope. Furthermore, there are usually more threads than channels in a system, so some threads have to be assigned to the same channel, which still interference with each other. Besides, channel partition actually partitions the bandwidth of memory system into several portions. Since the total number of portions is limited by channel amount, which is usually small, it is challenging to seek a balance among channels so as to ensure no bandwidth wasted.

Bank partition. In [26], frequently accessed data of different rows are dynamically migrated into row buffer, which can improve the row buffer usage and performance; power consumption is also lowered by reducing the operations of precharge and active [30]. In [27], the content in row-buffer will be precharged after 4 times access, which target at the reduction of row-buffer conflicts.

Memory deduplication. Waldspurger introduced content-based page sharing in a commodity virtual machine monitor [10]. It finds out identical pages and merges them based on the contents of pages. It based on scanning, which can adjust the scanning speed to reduce overhead. This scheme also uses a hash table to reduce the number of memory comparisons. When two pages have the same hash value, then it compares both pages in byte granularity.

Kernel same page merging (KSM) is memory deduplication technique used in Linux kernel [8]. This scheme uses a red-block tree to reduce the number of memory comparisons. But the comparisons are still too many.

To reduce scanning overhead, Sharma et al [28] proposed only scanning dirtied pages. Once the clean pages are scanned, no additional scanning is required. Chen et al [7, 33] exploits page access characteristics to reduce the number of memory comparisons during memory deduplication. Pages having similar access patterns have high probability to share. This scheme, however, requires hardware modification. Our CMDP neither modifying hardware nor adding additional information, it is really light scheme.

VII. CONCLUSION

In this paper, we propose a coordinate memory deduplication and partition approach named CMDP to reduce memory requirement and interference simultaneously for improving performance in virtualization. CMDP contains two parts of the CMDP. First, to reduce interferences among VMs, VMMP of the CMDP dynamically maps hypervisor, VMs and applications running on VMs onto different memory banks instead of accessing all memory banks. Through isolating their memory requests from others, the spatial locality can be reserved. Second, to reduce memory requests, BMD of the CMDP groups pages into different classifications based on both belonged banks and page accessed behavior. Pages belonged to memory banks of VMs are more possibility with the same content, moreover, pages with similar behavior, especially access behavior, are suggested to have higher possibility with same content. So pages belonged to memory banks of VMs and with similar behavior are grouped into the same classification. Therefore, performing page comparisons is restricted into the same classification, never exceeding to different classifications, which will do many futile comparisons.

REFERENCES

[1] Fei Xu, Fangming Liu, Hai Jin, V. Vasilakos. Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions. Proceedings of the IEEE, pages 11-31, Jan. 2014.

[2] Amazon.com, Customer success. Powered by the AWS Cloud. [Online]. Available: http://aws.amazon.com/solutions/case-studies/

[3] Amazon.com, Amazon elastic compute cloud (Amazon EC2) . [Online]. Available: http://aws.amazon.com/ec2/

[4] J. F. Gantz, S. Minton, and A. Toncheva, Cloud computings role in job creation, Mar. 2012. [Online]. Available: http://www.microsoft.com/en-us/news/features/2012/mar12/03-05cloudcomputingjobs.aspx

[5] R. P. Goldberg. Survey of virtual machine research. Computer, 7(9). pages 34-45. 1974.

[6] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends.Computer, 38(5):39-47, 2005.

[7] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, Y. Bao. CMD: classification-based memory deduplication through page access characteristics. In VEE14, 2014.

[8] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. InProceedings of the Linux Symposium (OLS09), 2009. pages 19-28.

[9] 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 Systems Design and Implementation, OSDI08, pages 309-322, 2008.

[10] C. A. Waldspurger. Memory resource management in vmware esx server. SIGOPS Oper. Syst. Rev., 36(SI):181-194, Dec. 2002.

[11] M. Jeong, D. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez. Balancing DRAM Locality and Parallelism in Shared Memory CMP Systems. HPCA, 2012.

[12] M. Xie, D. Tong, Y. Feng, K. Huang, X. Cheng. Page Policy Control with Memory Partitioning for DRAM Performance and Power Efficiency. ISLPED, 2013.

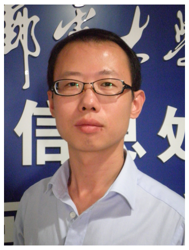
[13] W. Mi, X. Feng, J. Xue, and Y. Jia. Software-Hardware Cooperative DRAM Bank Partitioning for Chip Multiprocessors. NPC, 2010.
[14] M. Xie, D. Tong, K. Huang and X. Cheng. Improving System Throughput and Fairness Simultaneously in Shared Memory CMP Systems Via Dynamic Bank Partitioning. HPCA, 2014.
[15] H. Cheng, C. Lin, J. Li, and C. Yang. Memory Latency Reduction via Thread Throttling. MICRO, 2010.
[16] S. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning. MICRO, 2011.
[17] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. MemScale: Active Low-Power Modes for Main Memory. In ASPLOS, 2011.
[18] Li T, John L K, Sivasubramaniam A, et al. Understanding and improving operating system effects in control flow prediction. ACM Sigplan Notices. ACM, 2002, 37(10): 68-80.
[19] T. Wood, G. Tarasuk-Levin, P. She noy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. SIGOPS Oper. Syst. Rev., vol. 43, no. 3, pp. 27-37, July 2009.
[20] F. Bellard. Qemu, a fast and portable dynamic translator. In Proceedings of the annual conference on USENIX Annual Technical Conference, ATEC 05, pages 41-46, 2005.
[21] Kvm-kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page.
[22] ab - apache http server benchmarking tool. http://httpd.apache.org/docs/2.2/programs/ab.html.
[23] Sysbench: a system performance benchmark. http://sysbench.sourceforge.net/.
[24] M. K. Qureshi, and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In MICRO-39, 2006.
[25] R. Azimi, D. K. Tam, L. Soares, and M. Stumm. Enhancing Operating System Support for Multicore Processors by Using Hardware Performance Monitoring. In ACM SIGOPS Operating Systems Review 43(2): 56-65, 2009.
[26] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-Pages: Increasing DRAM Efficiency with Locality-Aware. In ASPLOS-2010.
[27] D. Kaseridis, J. Stuecheli, and L. K. John. Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the many-core Era. In MICRO-44, 2011.
[28] P. Sharma, and P Kulkarni. Singleton: system-wide page deduplication in virtual environments, in Proc. the 21st Int. Symp. on High-Performance Parallel and Distributed Computing, pp. 15-26, 2012.
[29] G. Jia, G. Han, L. Shi, J. Wan, D. Dai. Combine Thread with Memory Scheduling for Maximizing Performance in Multi-core Systems. ICPADS, 2014.
[30] G. Jia, G. Han, J. Jiang, J.J.P.C. Rodrigues. PARS: A scheduling of periodically active rank to optimize power efficiency for main memory. Journal of Network and Computer Applications, 2015.
[31] G. Jia, G. Han, J. Jiang, A. Li. Dynamic Time-slice Scaling for Addressing OS Problems Incurred by Main Memory DVFS in Intelligent System. Mobile Networks and Applications 20 (2), 157-168, 2015.
[32] G. Jia, G. Han, A. Li, J. Lloret. Coordinate Channel-Aware Page Mapping Policy and Memory Scheduling for Reducing Memory Interference Among Multimedia Applications. IEEE System Journal, 2015.
[33] Z. Shu, J. Wan, D. Zhang and D. Li. Cloud-integrated Cyber-Physical Systems for Complex Industrial Applications, ACM/Springer Mobile Networks and Applications, DOI: 10.1007/s11036-015-0664-6, 2015

Gangyong Jia is currently an Assistant Professor of Department of Computer Science at Hangzhou Dianzi University, China. He received his Ph.D. degree in Department of Computer Science from University of Science and Technology of China, Hefei, China, in 2013. He is currently a Visiting Scholar with Hohai University, Changzhou, China. He has published over 20 papers in related international conferences and journals. He has served as a reviewer of Microprocessors and Microsystems. His current research interests are power management, operating system, cache optimization, memory management. He is a member of IEEE.

Guangjie Han is currently a Professor of Department of Information & Communication System at Hohai University, China. He is also a visiting research scholar of Osaka University from Oct. 2010 to Oct. 2011. He finished the work as a post doctor of Department of Computer Science at Chonnam National University, Korea, in February 2008. He worked in ZTE Company from 2004 to 2006, where he held the position of Product Manager. He received his Ph.D. degree in Department of Computer Science from Northeastern University, Shenyang, China, in 2004. He has published over 180 papers in related international conferences and journals. He has served in the editorial board of up to 14 international journals, including Journal of Internet Technology and KSII Transactions on Internet and Information Systems. He has served as a Co-chair for more than 20 international conferences/workshops; a TPC member of more than 100 conferences. He holds 70 patents. He has served as a reviewer of more than 50 journals. He had been awarded the ComManTel 2014, ComComAp 2014 and Chinacom 2014 Best Paper Awards. His current research interests are Sensor Networks, Computer Communications, Mobile Cloud Computing, Multimedia Communication and Security. He is a member of IEEE and ACM.

Joel J.P.C. Rodrigues received a PhD degree in informatics engineering, an MSc degree from the University of Beira Interior, and a five-year BSc degree (licentiate) in informatics engineering from the University of Coimbra, Portugal. His main research interests include sensor networks, e-health, e-learning, vehicular delay-tolerant networks, and mobile and ubiquitous computing. He is the leader of NetGNA Research Group (http://netgna.it.ubi.pt), the Chair of the IEEE ComSoc Technical Committee on eHealth, the Past-chair of the IEEE ComSoc Technical Committee on Communications Software, Member Representative of the IEEE Communications Society on the IEEE Biometrics Council, and officer of the IEEE 1907.1 standard. He is the editor-in-chief of the International Journal on E-Health and Medical Communications, the editor-in-chief of the Recent Advances on Communications and Networking Technology, and editorial board member of several journals. He has been general chair and TPC Chair of many international conferences. He is a member of many international TPCs and participated in several international conferences organization. He has authored or coauthored over 350 papers in refereed international journals and conferences, a book, and 3 patents. He had been awarded the Outstanding Leadership Award of IEEE GLOBECOM 2010 as CSSMA Symposium Co-Chair and several best papers awards. Prof. Rodrigues is a licensed professional engineer (as senior member), member of the Internet Society, an IARIA fellow, and a senior member of ACM and IEEE.

Jaime Lloret received his M.Sc. in Physics in 1997, his M.Sc. in electronic Engineering in 2003 and his Ph.D. in telecommunication engineering (Dr. Ing.) in 2006. He is a Cisco Certified Network Professional Instructor. He worked as a network designer and administrator in several enterprises. He is currently Associate Professor in the Polytechnic University of Valencia. He is the head of the research group ”communications and remote sensing” of the Integrated Management Coastal Research Institute and he is the head of the ”Active and collaborative techniques and use of technologic resources in the education (EITACURTE)” Innovation Group. He is the director of the University Expert Certificate Redes y Comunicaciones de Ordenadores, the University Expert Certificate Tecnologłas Web y Comercio Electrnico, and the University Master ”Digital Post Production”. He is currently Chair of the Internet Technical Committee (IEEE Communications Society and Internet society). He has authored 12 books and has more than 240 research papers published in national and international conferences, international journals (more than 80 with ISI Thomson Impact Factor). He has been the co-editor of 15 conference proceedings and guest editor of several international books and journals. He is editor-in-chief of the international journal ”Networks Protocols and Algorithms”, IARIA Journals Board Chair (8 Journals) and he is associate editor of several international journals. He has been involved in more than 200 Program committees of international conferences and in many organization and steering committees. He led many national and international projects. He is currently the chair of the Working Group of the Standard IEEE 1907.1. He has been general chair (or co-chair) of 19 International workshops and conferences (chairman of SENSORCOMM 2007, UBICOMM 2008, ICNS 2009, ICWMC 2010, eKNOW 2012, SERVICE COMPUTATION 2013, COGNITIVE 2013, and ADAPTIVE 2013, and co-chairman of ICAS 2009, INTERNET 2010, MARSS 2011, IEEE MASS 2011, SCPA 2011, ICDS 2012, 2nd IEEE SCPA 2012, GreeNets 2012, 3rd IEEE SCPA 2013, SSPA 2013 and local chair of MIC-WCMC 2013). He is co-chairman AdHocNow 2014, MARSS 2014, SSPA 2014 and GreeNets 2014, and local chair IEEE Sensors 2014. He is IEEE Senior and IARIA Fellow.



Wei Li received the B.S. degree in electrical engineering and the M.S. degree in communication engineering from Hohai University, Changzhou, China, in 2003 and 2007, respectively. He is currently working toward the Ph.D. degree with Nanjing University of Posts and Telecommunications. He is currently an Associate Professor with the College of Internet of Things Engineering, Hohai University. His research interests include statistical inference, distributed signal processing, and cooperative localization and tracking in wireless sensor networks.