IEEE Access
The journal for rapid open access publishing

Received October 7, 2015, accepted October 25, 2015, date of publication November 2, 2015,
date of current version November 13, 2015.

Digital Object Identifier 10.1109/ACCESS.2015.2496959

An Adaptive Framework for Improving Quality of Service in Industrial Systems

GANGYONG JIA[1,2], (Member, IEEE), GUANGJIE HAN[3], (Member, IEEE), DAQIANG ZHANG[4], (Senior Member, IEEE), LI LIU[3], and LEI SHU[5], (Member, IEEE)

[1]Department of Computer Science, Hangzhou Dianzi University, Hangzhou 310018, China
[2]Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China
[3]Department of Information and Communication Systems, Hohai University, Changzhou 213022, China
[4]School of Software Engineering, Tongji University, Shanghai 200092, China
[5]Guangdong Petrochemical Equipment Fault Diagnosis Key Laboratory, Guangdong University of Petrochemical Technology, Guangdong 525000, China

Corresponding author: G. Han (hanguangjie@gmail.com)

This work was supported in part by the Qing Lan Project, in part by the National Science Foundation of China under Grant 61572172, Grant 61472283, Grant 61401147, and Grant 61401107, in part by the Educational Commission of Guangdong Province, China, under Project 2013KJCX0131, in part by the Fok Ying-Tong Education Foundation, China, under Grant 142006, in part by the Fundamental Research Funds for the Central Universities under Grant 2013KJ034, Grant 2100219043, and Grant 1600219246/001, and in part by the Zhejiang Provincial Natural Science Foundation under Grant LQ14F020011.

ABSTRACT Limited memory bandwidth is considered as the major bottleneck in multimedia cloud computing for more and more virtual machines (VMs) of multimedia processing requiring high memory bandwidth simultaneously. Moreover, contending memory bandwidth among parallel running VMs leads to poor quality of service (QoS) of the multimedia applications, missing the deadlines of these soft real-time multimedia applications. In this paper, we present an adaptive framework, Service Maximization Optimization (SMO), which is designed to improve the QoS of the soft real-time multimedia applications in multimedia cloud computing. The framework consists of an automatic detection mechanism and an adaptive memory bandwidth control mechanism. With the automatic detection mechanism, the critical section to the multimedia application performance in the VMs is detected. Then, our adaptive memory bandwidth control mechanism adjusts the memory access rates of all the parallel running VMs to protect the QoS of the soft real-time multimedia applications. From the case studies with real-world multimedia applications, our SMO significantly improves the QoS of the soft real-time multimedia applications with a negligible penalty on system throughput.

INDEX TERMS Memory bandwidth, multimedia application, memory access, quality of service, soft real-time.

I. INTRODUCTION

With the better and better in scaling computing resources and providing a simple pay-as-you-go business model for customers, cloud computing is a promising economical computing paradigm, and has gained much attention 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, services based on cloud computing is the dominate model in the future for both flexibility and budget, the International Data Corporation (IDC) has reported that the business revenue brought by cloud computing will reach $1.1 trillion in 2015 [4].

In cloud computing, a single physical server can collocate multiple virtual machines (VMs), and through the virtualization technology VMs can operate independently [5], [6]. Virtualization technology provides flexible allocation, migration of services, and better security isolation. In the virtualization environment, hypervisor (or Virtual Machine Monitor, VMM) manages physical resources (such as memory bandwidth). 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 VMs of multimedia applications require high memory bandwidth, therefore, the soft real-time multimedia applications have poor QoS for serious contending memory bandwidth with parallel running VMs. The demand for memory bandwidth is much advance to the increasing speed, so the limited memory bandwidth is the major bottleneck to the QoS of the multimedia applications.

In order to address the limited memory bandwidth induced soft real-time multimedia applications poor QoS problem, there are some strategies:

1) Disable all VMs of non-real-time applications to eliminate the shared memory bandwidth contention problem [8], which decreases the whole system performance seriously, it is unacceptable in normal cloud computing.
2) Reduce memory access from shared last level cache. This strategy can be achieved through improving utilization of the last level cache, such as optimizing the cache replacement policy, and so on.
3) Place VMs of different multimedia applications running parallel for both different memory bandwidth requirements and different QoS requirements. Due to the heterogeneity of the media applications, such as streaming service, video transcoding services, rendering services and so on, media cloud has brought up the need for an efficient VM allocation in the cloud platform to utilize the memory bandwidth efficiently and satisfy the QoS requirements of the media applications, especially when different atomic media applications are composed to meet the customer demands [10]. These VMs of different media applications have different memory bandwidth and QoS requirements, and need VM placement at the run-time. In addition, the dynamic of the media services demands makes it difficult to efficiently allocate VM in the cloud platform, while fulfilling QoS demands.
4) Optimize the memory scheduling, maximizing the efficiency of the memory bandwidth. Utilize the significant amount of parallelism, such as out-of-order cores, multi-bank DRAM, and so on, which can process a considerable degree of concurrent accesses without any noticeable performance impacts [9]. Therefore, it is highly desirable to develop a solution that can provide better QoS of multimedia applications while protecting the whole system performance through leveraging both the parallelism multi-bank and memory scheduling.

To improve QoS of the soft real-time multimedia applications in the media cloud computing, we propose an adaptive framework, SMO, which consists three parts: the first is the memory deduplication to reduce memory access, because the same contents have already been in the cache; the second is the VM placement, place VMs of different memory bandwidth requirements running parallel to satisfy QoS better; the third is the memory scheduling, priority schedule memory access from real-time applications, and protect the whole system simultaneously. Our SMO has two mainly mechanisms, the automatic detection mechanism and the adaptive memory bandwidth control mechanism. With the automatic detection mechanism, the critical section to the multimedia applications performance in the VMs is detected. Then our adaptive memory bandwidth control mechanism adjusts the memory access rates of all the parallel running VMs to protect the QoS of the soft real-time multimedia applications.

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

1) We propose an adaptive framework to improve QoS in multimedia cloud computing while protecting the whole system performance simultaneously. SMO consist memory deduplication, bandwidth-aware VM placement and memory scheduling.
2) Our SMO can detect the critical section to the multimedia applications performance in the VMs automatically.
3) Experimental results show our SMO is well in both improving QoS and maximizing performance of the whole system.

The rest of this paper is organized as the follows. Section II elaborates on essential background and the related work. Section III discusses research motivations. Section IV explains our adaptive framework, SMO. Section V describes experimental methodology and section VI presents the results of our experiments. Finally, section VII concludes this paper.

II. BACKGROUND & RELATED WORKS

In this section, we provide a review of DRAM system and discuss how past research dealt with the challenges of providing QoS to the soft real-time multimedia applications in multimedia cloud computing.

A. DRAM SYSTEM

DRAM Organization: Figure 1 demonstrates the organizations of multiple levels memory subsystem. After receiving the memory accesses from CPU, the memory controller (MC) sends corresponding commands to the off-chip memory subsystem through the bus. Normally, the MC is integrated into the same package of the CPU recently [26]. In order to improve parallelism, a memory subsystem consists multiple channels, which can act independently, and each channel connects multiple DIMMs. Moreover, each DIMM comprises multiple DRAM chips. The DRAM chips are the ultimate destination of the MC commands. According to the participation of each access, the DRAM chips are partitioned into ranks. However, how many chips a rank contains are depending on CPU. Recently, a DIMM can have up to 16 chips, organized into 1-4 ranks [11].

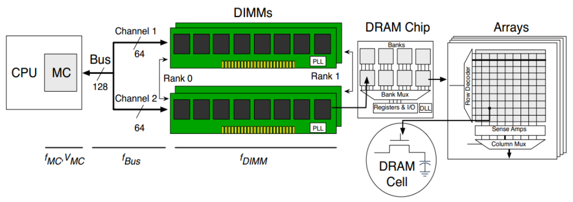


FIGURE 1. Organization of a modern memory subsystem.

In order to access parallel, each DRAM chip has many banks (8 or more banks nowadays). Every bank contains multiple two-dimensional memory arrays. The basic unit is the DRAM cell which is one bit of the storage and a simple capacitor. Thus, in a DRAM chip of x8, which means every bank has 8 arrays. So, each array in the bank accesses one bit at a time. However, when an array is accessed, its corresponding entire multi-KB row is transferred to the row buffer. This operation is called a row opening or activation. Then, any column of the row can be read/written in one burst. Otherwise, the row needs to be written back. After so, new row and column can be accessed.

B. RELATED WORKS

There are a number of related studies to our proposed framework.

Memory Deduplication: Waldspurger introduced content-based page sharing in a commodity virtual machine monitor [18]. 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 [13]. 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 and Kulkarni [19] proposed only scanning dirtied pages. Once the clean pages are scanned, no additional scanning is required. Chen *et al.* [14] exploit 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.

VM Scheduling: Scheduling algorithms aimed to distribute VMs to get an even distribution of miss rate among multiple caches are proposed in [20] and [27], which avoid severe contention on shared resource of cache, memory controller, memory bus and prefetching hardware. Similar mechanisms are also proposed in [21] and [22]. Although these methods can alleviate contention, they hardly eliminate the bank interference among VMs [28].

Thread-Based Memory Scheduling: Memory controllers are designed to distinguish the memory access behavior at VM-level in [23], [24], and [29], so that scheduling modules can adjust their scheduling policy at the running time. TCM [25], which dynamically groups VMs into two clusters (memory intensive and CPU intensive), and assign different scheduling policy to different group, is the best scheduling policy, which aim to address fairness and throughput at the same time. Yet, this method needs modification to memory controller, and the overhead at running time cannot be neglected.

III. MOTIVATIONS

A. PROFILING OF MEMORY SHARING

Basically, memory deduplication of reducing memory bandwidth, which many contents have been in the last level cache from other VMs access, 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 VMs of different applications. The software as well as the data used in VMs can be similar [12]. Therefore, through merging those identical contents pages, the physical system can share those pages to reduce requirements of the memory bandwidth.

Moreover, the guest OS running on each VM also have many identical contents. Figure 2 demonstrates the identical pages proportion between two VMs without applications running on them. In the figure, the x-axis presents

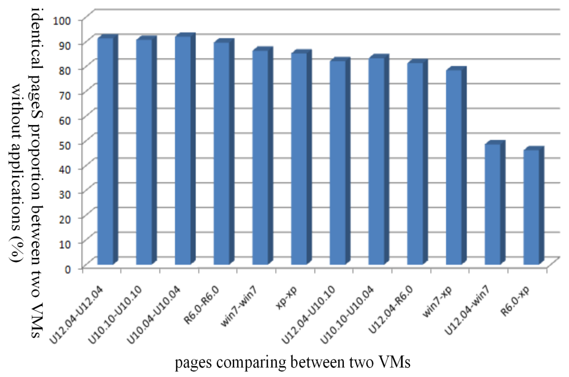


FIGURE 2. Identical pages proportion between two VMs without applications.

the two guest 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%. Normally, every server has only one kind of guest OS for services. Therefore, it is a good opportunity to alleviate memory bandwidth requestments using memory deduplication.

B. COMPARISON OVERHEAD ANALYSIS OF KSM

Kernel Samepage Merging (KSM) [13], 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 is the implementation of Content Based Page Sharing (CBPS), 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.

C. PROFILING OF VM PLACEMENT

VM placement in the multimedia cloud computing, especially with some VMs need satisfy soft real-time requirement, is the key to meet the shared memory bandwidth requirement, which is also the QoS. The multimedia cloud provides services like image/video retrieval, video transcoding, streaming, video rendering, media analytics, sharing and delivery and so on, and these services are heterogeneous in shared memory bandwidth demands. For example, the application of video transcoding demands more CPU, but the application of video streaming demands more bandwidth. If we place two VMs of the same video streaming running parallel, the memory bandwidth for each one will decrease seriously, but place two VMs, one for video transcoding and the other for video streaming, will satisfy both applications, the video transcoding can use the whole CPU and the video streaming can occupy almost the whole memory bandwidth. Figure 3 provides the average normalized performance with different services and different numbers running parallel. In the figure, x-axis provides the different configurations, vt represents video transcoding, vs represents video streaming, vr represents video retrieval and vd represents video rendering. n-name demonstrates n VMs of the name services running parallel, for example, 2-vt demonstrates 2 VMs of video transcoding services running parallel. Through the figure, we can know if we place heterogeneous services parallel, the

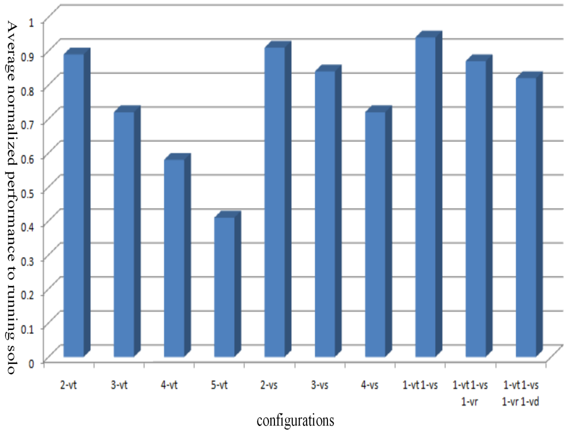


FIGURE 3. The average normalized performance with different services and different numbers running parallel.

performance will be improved, especially when more VMs running parallel.

D. PROFILING OF MEMORY SCHEDULING

Memory scheduling is important to the performance of the soft real-time multimedia applications after some applications running parallel. Figure 4 demonstrates two applications running parallel while sharing the off-chip memory. If application 0 is soft real-time, and application 1 is not. In current memory scheduling policy, first in first out (FIFO), application 0 will miss the deadline seriously, while application 1 can contribute more memory bandwidth. Therefore, we can priority schedule memory access from soft real-time application to meet deadline while protecting other parallel running applications.

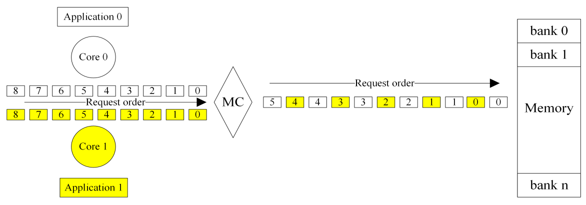


FIGURE 4. Two applications run parallel while sharing the off-chip memory.

Moreover, memory scheduling can improve efficiency of the memory bandwidth. Firstly, we use the memory access trace: (0, 0, 0), (1, 0, 0), (0, 0, 1), (1, 1, 0), (0, 0, 2), (1, 2, 0), (0, 0, 3), (0, 0, 4), (0, 0, 5), (0, 0, 6), (0, 0, 7), (0, 0, 8) as example to compare two different memory scheduling policies, in-order scheduling and burst scheduling. The triplet (x, y, z) represents a memory access, and x stands for the id of the bank, y stands for the row, and z stands for the column. One memory access comes to the system every clock cycle. In this example, bank precharge, row activate and column access take $Db = 3$, $Dr = 3$ and $Dc = 1$ clock cycles respectively.

However, there are some rules in the memory scheduling: 1) every clock cycle there is only one micro operation can be issued; 2) for the bank activate needs 3 clock cycles, even two accesses are in the same bank, row activate must be issued two clock cycles after bank precharge; 3) for the row activate needs 3 clock cycles, even two accesses are in the same row, a column access must be issued two cycles after row activate; 4) for the column access only needs 1 clock cycles, if two accesses are in the same column, column access can be issued Immediately.

Figure 5 (left) and (right) show the result sequence of the in-order memory access scheduling and bursts respectively. The x-axis shows the scheduling order from left to right and y-axis shows the execution clock cycles from top to bottom. The in-order memory scheduling takes 36 clock cycles to finish the example memory access and the burst memory scheduling takes 30 cycles to finish the same memory access. Therefore, the burst memory scheduling reduces 16.67% clock cycles, which also means reduces 16.67% memory bandwidth. So, we can improve efficiency of the memory bandwidth through memory scheduling to improve QoS.

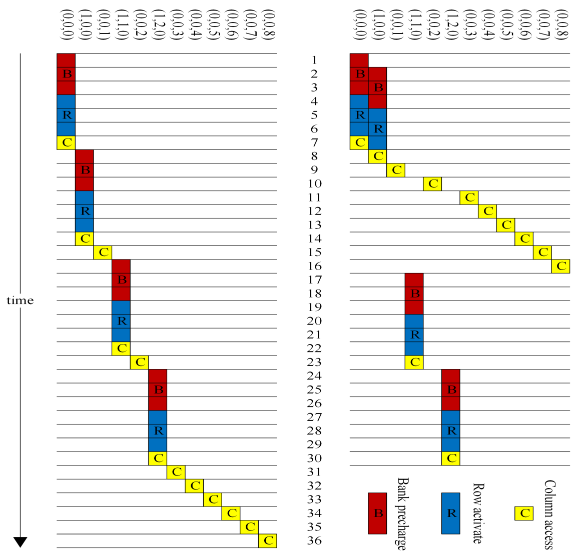


FIGURE 5. In-order memory accesses scheduling (left) and burst memory accesses scheduling (right).

IV. THE ADAPTIVE FRAMEWORK (SMO)

In this section, we firstly introduce the overview of the adaptive framework (SMO) in subsection 3.1. Then we introduce the automatic detection mechanism to determine the memory bandwidth requirement of every VM in subsection 3.2, introducing a lightweight page behavior-based memory deduplication approach in subsection 3.3. In subsection 3.4, we introduce VM placement policy. Finally, we introduce the adaptive memory bandwidth control mechanism through memory scheduling in subsection 3.5.

A. OVERVIEW OF SMO

To improve QoS of the soft real-time multimedia applications in multimedia cloud computing while protecting the whole systems performance, we propose an adaptive framework (SMO). It contains four parts of the SMO.

First, to estimate VMs needs for shared memory bandwidth, the automatic detection mechanism is to mainly profile VMs memory intensity.

Second, to reduce memory access from shared last level cache, improve efficiency of the last level cache, propose a lightweight page behavior-based memory deduplication approach. Through sharing pages, more contents will be in the cache.

Third, propose a memory bandwidth-aware VM placement policy to reduce competition among parallel running VMs on memory bandwidth.

Finally, we propose an adaptive memory bandwidth control mechanism through memory scheduling, which priority schedules memory accesses from soft real-time multimedia applications while improving efficiency of the memory bandwidth. Figure 6 demonstrates our SMO adaptive framework.

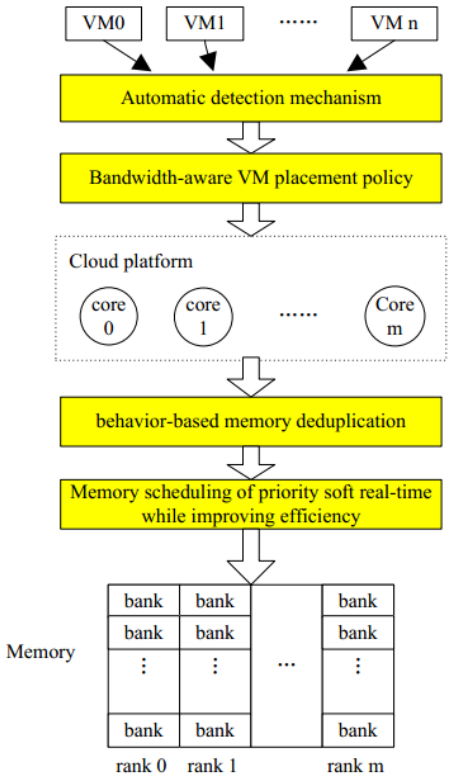

VM0
VM1
......
VM n
Automatic detection mechanism
Bandwidth-aware VM placement policy
Cloud platform
core 0
core 1
......
Core m
behavior-based memory deduplication
Memory scheduling of priority soft real-time while improving efficiency
Memory
bank
rank 0
rank 1
rank m

FIGURE 6. The adaptive framework of SMO.

B. AUTOMATIC DETECTION MECHANISM

We define a VMs requirements of the memory bandwidth using memory intensity. Memory intensity, the frequency of a VM misses in the last-level cache or generates memory requests, is used to distinguish between low memory intensity and high memory intensity. In this paper, we use last-level cache misses per thousand instructions (MPKI) to represent memory intensity. The more MPKI, the VM is higher in memory intensity. The low memory intensity VMs are sensitive to the respond time, which will reduce serious performance if waiting times is longer, however, they have low requirements in memory bandwidth for seldom generating memory requests. Therefore, in order to protect the performance of the whole system, we need to retain as many required memory accesses as possible. The high memory intensity VMs with non-soft real time are unsensitive to the respond time, which will keep performance after decreasing memory bandwidth. Therefore, in order to improve QoS of the soft real-time applications, we need to both retain as many required memory accesses as possible from low memory intensity VMs with non-soft real-time applications and decrease as many accesses as possible from high memory intensity VMs with non-soft real-time applications while meeting the deadline of the soft real-time applications.

C. BANDWIDTH-AWARE VM PLACEMENT

Our bandwidth-aware VM placement is based on both the VMs memory bandwidth requirements and real-time requirements. Place the VMs of different memory bandwidth requirements and different real-time characteristics parallel to improve the utilization of all hardware resources, especially memory bandwidth, therefore, improve the QoS of the soft real-time applications while protecting the whole system performance. If one VMs load is mainly bandwidth with soft real-time, the parallel running VMs can be CPU-bound with non-soft real-time, or VMs of high memory intensive with non-soft real-time.

We define some parameters in our algorithm. C1, if the applications MPKI <C1, it is a low memory intensive VM. C2, if the applications MPKI >C2, it is a high memory intensive VM. In this paper, we dont consider the real-time of the low memory intensive VM, so we distinguish the high memory intensive VM with soft real-time and high memory intensive VM with non-soft real-time. Otherwise, all other applications are normal VMs, which their C1 <MPKI <C2. And all normal VMs are partitioned into soft real-time and non-soft real-time. In this paper, we specify the $C1 = 3$, $C2 = 8$.

So, our bandwidth-aware VM placement policy will consider the five types VMs: low memory intensive (represented by L-VM), high memory intensive with soft real-time (represented by HR-VM), high memory intensive with non-soft real-time (represented by HN-VM), normal memory intensive with soft real-time (represented by NR-VM), normal memory intensive with non-soft real-time (represented by NN-VM). We will place one HR-VM with some VMs like L-VM, NN-VM, or HN-VM parallel, but we would not like to place some HR-VMs or NR-VMs parallel.

D. PAGE BEHAVIOR-BASED MEMORY DEDUPLICATION

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 [14]. The key innovation to reduce futile comparison 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 SMO, 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.

Our SMO 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.

E. MEMORY SCHEDULING

Based on the memory access requirements of all parallel running VMs and the deadline of the soft real-time multimedia applications, we determine all parallel non-soft real-time applications memory access ratio. In order to meet the deadline to improve QoS of the soft real-time VM, our memory scheduling policy need to satisfy the bandwidth from soft real-time VMs firstly. Next, protect the L-VM to improve the whole system performance, so satisfy the memory accesses from L-VMs as much as possible. Finally, the remainder bandwidth is allocated to other VMs.

In order to improve efficiency of the memory bandwidth, we combine the BFS Algorithm, demonstrated in figure 7. The memory accesses to be scheduled are the input of the BFS algorithm. The (x, y, z) triplet represents a memory access. And the sequence of the fulfilling the memory accesses is the output of the algorithm. List all micro operations and empty operations referring to the same bank in a bank list according to their sequence. Define i and the number of empty clock cycles of the list as the bank weight and operation weight of bank list. Similarly, the waited clock cycles is the age weight of bank list, the value is reset to 0 whenever it is scheduled. Schedule a micro operation from bank list based on age weight once after using operation weight Q times.

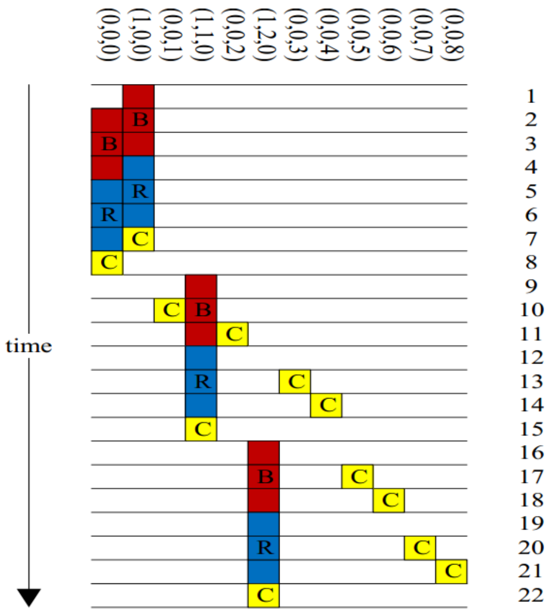


FIGURE 7. BFS memory access scheduling.

The BFS algorithm is described in the Algorithm 1. In order to prevent starvation, each bank list consists of two queues: Queue1 and Queue2, and each queue can take P memory accesses. Initially, both Queue1 and Queue2 are empty, but after P memory accesses has been directed

Algorithm 1 BFS algorithm

```
Input: memory accesses.
Output: scheduled memory micro operations.
Begin
 loop
  if new (x, y, z) comes
   add it into row list y of bank list z;
   move row list y to the right position according
   ascending order in the bank list z;
  if clock cycle % (Q+1) = 0
   use age weight as priority;
  else
   use operation weight as priority;
  for all bank list do
   chose a ready micro operation based on priority;
   add micro operation to to_be_scheduled;
  if to_be_scheduled not null
   schedule to_be_scheduled and delete it from its bank
   list;
   clock cycle ++;
End
```

to a queue, which becomes full, therefore, new memory accesses will go to the other queue. And then both queues are full, new memory access will go to a buffer associated with each bank list. During scheduling, memory accesses are scheduled from one of the queues until empty, after that memory controller will pick memory access from the other queue.

V. EXPERIMENTAL SETUP

We carried out our experiments with one 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 the 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 8MB and shared by all four cores in the 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 SMO based on KSM of Linux 3.6.10. We use QEMU [15] with KVM [16] (qemu-kvm-1.2.0) to support guest VMs. Each guest VM is configured with 1 virtual CPU, we boot 8 VMs simultaneously in the experiment to evaluate our bandwidth-aware VM placement of SMO. We also boot more VMs 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: image/video retrieval, video transcoding, streaming, video rendering, media analytics, sharing and delivery.

VI. EXPERIMENTAL SETUP

In this section, we present case-study results using one soft real-time VM, video player, to evaluate the effectiveness of SMO.

A. QoS OF THE VIDEO PLAYER VM

In this paper, we use the reciprocal of the average frame processing time as the metric of QoS for the video player VM. In this experiment, our video player plays a H264 movie clip with a frame resolution of 1920*816 and a frame rate of 24fps [17].

To investigate the effectiveness of SMO, we conducted a set of experiments. We run video player VM with other VMs parallel. In order to reflect the random placement of the VM, we average as many situations as possible.

Figure 8 shows the results. In the figure, QoS is normalized to the running solo and the x-axis demonstrates some default situations with random VM placement. In default, the video player VMs performance is significantly degraded, dropped by 54% due to memory bandwidth contention. As expect, video player VMs QoS improves significantly after using our SMO.

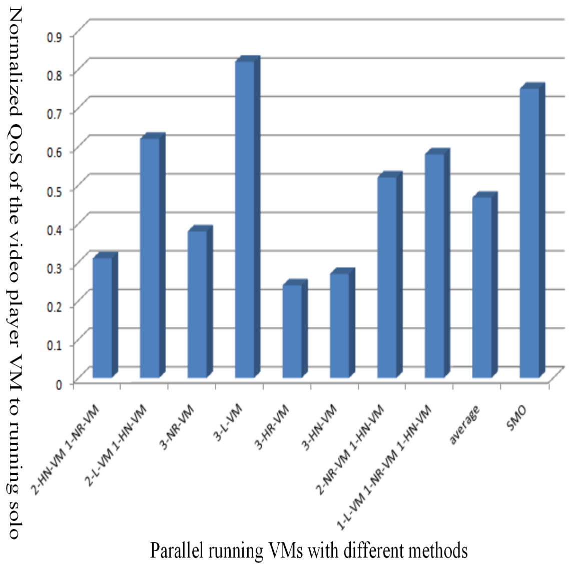


FIGURE 8. Normalized QoS of the video player VM.

Figure 9 demonstrates the results of normalized QoS of the video player VM using SMO without bandwidth-aware VM placement. Comparing to the figure 8, SMO with bandwidth-aware VM placement is better than the SMO without bandwidth-aware VM placement. This result proves our bandwidth-aware VM placement is good for improving QoS of the soft real-time multimedia applications.

Figure 10 shows the normalized bandwidth occupancy rate of the different parallel running VMs in different frameworks. The QoS improvement of video player is mainly from the high occupancy rate.

B. WHOLE SYSTEM PERFORMANCE ANALYSIS

Figure 11 shows the normalized whole system performance. The x-axis demonstrates some random VM placement situations with default management and our SMO. In the figure, the whole system performance of our SMO is better than the default management.

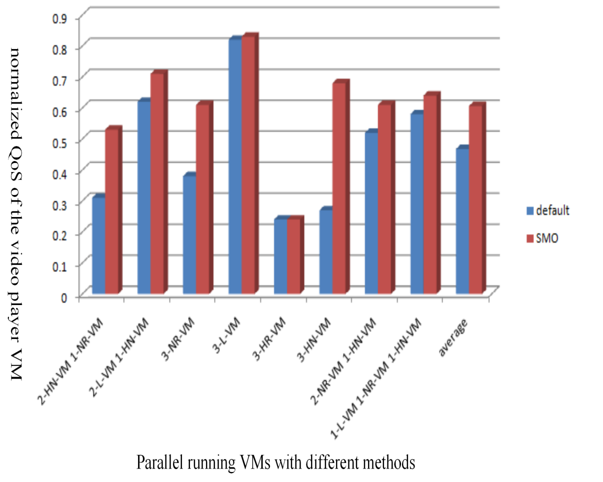


FIGURE 9. Normalized QoS of the video player VM using SMO without bandwidth-aware VM placement.

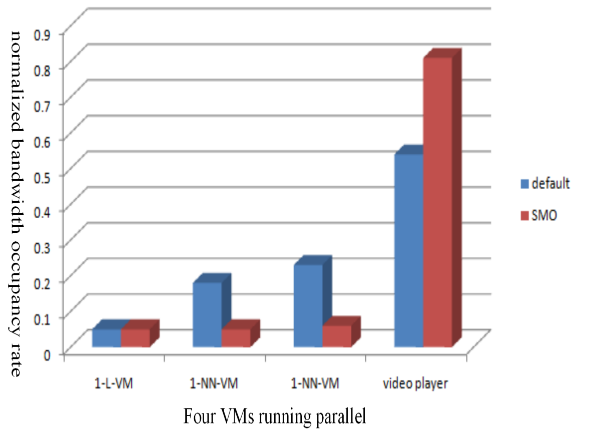


FIGURE 10. Normalized bandwidth occupancy rate of the different parallel running VMs in different frameworks.

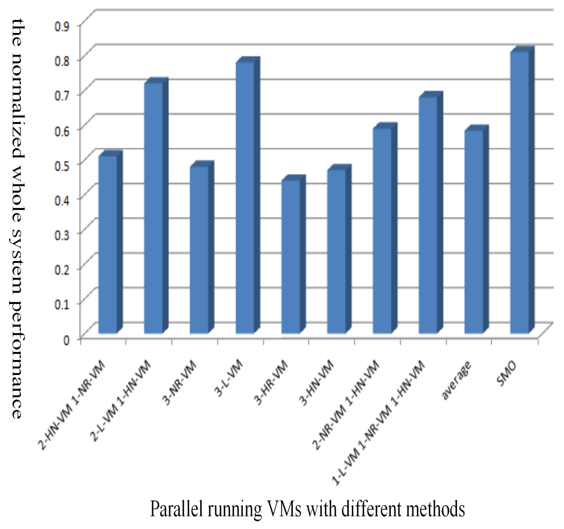


FIGURE 11. Shows the whole system performance normalized.

Figure 12 shows normalized whole system performance using SMO without bandwidth-aware VM placement. The results demonstrate SMO with bandwidth-aware

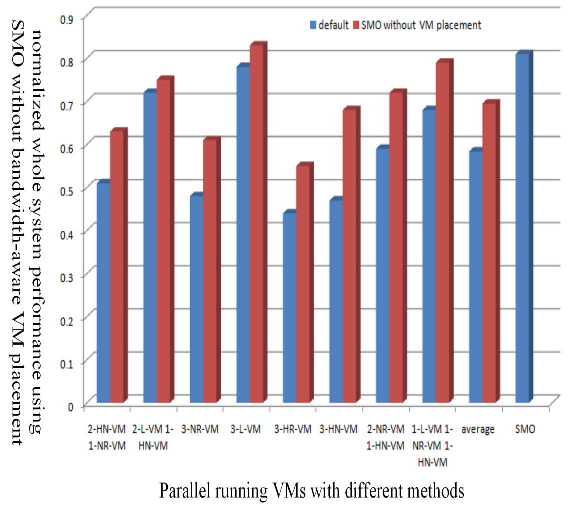


FIGURE 12. **The normalized whole system performance using SMO without bandwidth-aware VM placement.**

VM placement is better than the SMO without bandwidth-aware VM placement, although the SMO without bandwidth-aware VM placement is better than the default management. This result proves our bandwidth-aware VM placement is good for not only improving QoS of the soft real-time multimedia applications but also improving the whole system performance.

Figure 13 demonstrates the reduced last level cache misses normalized to the default. Our SMO can reduce the last level cache misses compared with the default. The result proves our memory deduplication policy can improve the efficiency of the last level cache, which improves the whole system performance.

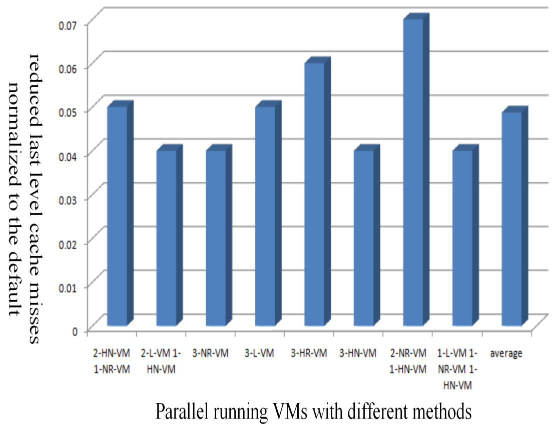


FIGURE 13. **The reduced last level cache misses normalized to the default.**

Figure 14 shows the improved bandwidth efficiency normalized to the default. In this paper, we use the completed memory accesses per second. Our SMO can optimize the bandwidth efficiency compared to the default. The improvement of the bandwidth efficiency is mainly from the memory scheduling, which reduces the average memory access time. Figure 15 demonstrates the reduced average memory access time normalized to the default, which shows our SMO can reduce the average memory access time. Therefore, our BFS memory scheduling can optimize the bandwidth utilization to improve the whole system performance.

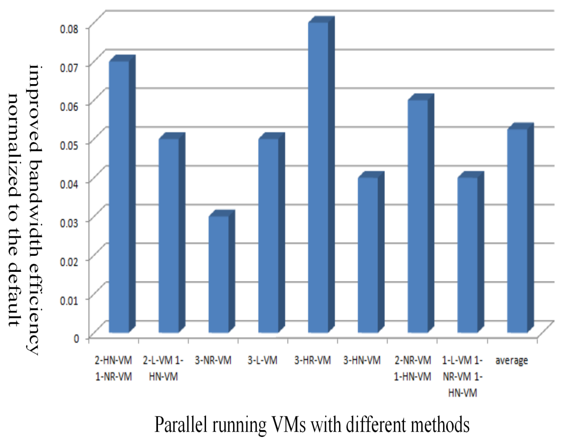


FIGURE 14. **The improved bandwidth efficiency normalized to the default.**

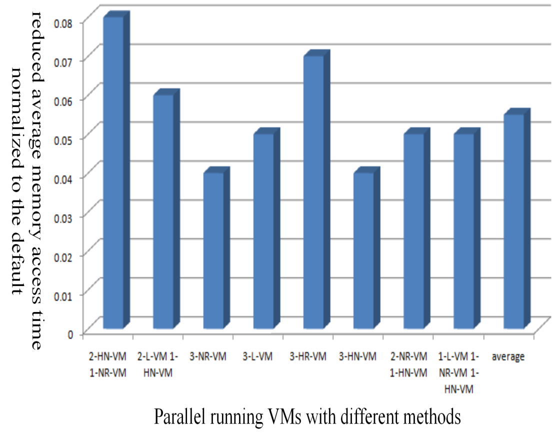


FIGURE 15. **The reduced average memory access time normalized to the default.**

C. OVERHEAD ANALYSIS

Software Support. There are three parts which requires system software support, the first one is the bandwidth-aware VM placement, is based on both the VMs memory bandwidth requirements and real-time requirements. In this paper, we use memory intensity as the VMs requirements of memory bandwidth, which needs to count last-level cache misses per thousand instructions (MPKI). Place the VMs of different memory bandwidth requirements and different real-time characteristics parallel to improve the utilization of all hardware resources, especially memory bandwidth, therefore, improve the QoS of the soft real-time applications while protecting the whole system performance.

The second one is the page behavior-based memory deduplication, in order to reduce memory requirements, we need to detect content of pages to determine whether pages can be shared. But our page behavior-based memory deduplication 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 pages into 4 classifications behaving the same status.

The third one is the memory scheduling, which is based on the memory access requirements of all parallel running VMs and the deadline of the soft real-time multimedia applications, we determine all parallel non-soft real-time applications memory access ratio. In order to improve efficiency of the memory bandwidth, we combine the BFS Algorithm.

Performance Overhead. In order to evaluate the performance overhead after adopting our SMO, we compare our SMO with the default method, and the experimental results show the performance of our SMO is negligible, below 2%; we compare our bandwidth-aware VM placement with the default method, our bandwidth-aware VM placement improves more than 25% performance; also, we compare our page behavior-based memory deduplication with the default method without KSM, although our page behavior-based memory deduplication decreases 8% on average, increase the memory sharing by more than 30%; moreover, our memory scheduling improves more than 10% performance. All above results have shown our SMO behave well.

VII. CONCLUSION

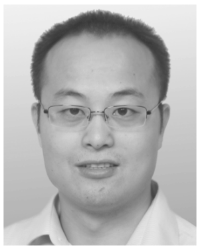
To improve quality of service (QoS) of soft real-time multimedia applications in multimedia cloud computing while improving the whole system performance, we propose an adaptive framework, SMO, which consists three parts: the first is the memory deduplication to reduce memory access, because the same contents have already been in the cache; the second is the VM placement, place VMs of different memory bandwidth requirements running parallel to satisfy QoS better; the third is the memory scheduling, priority schedule memory access from real-time applications, and protect the whole system simultaneously. Our SMO has two mainly mechanisms, the automatic detection mechanism and the adaptive memory bandwidth control mechanism. With the automatic detection mechanism, the critical section to the multimedia applications performance in the VMs is detected. Then our adaptive memory bandwidth control mechanism adjusts the memory access rates of all the parallel running VMs to protect the QoS of the soft real-time multimedia applications. The experiments have proven the effectiveness of the SMO.

REFERENCES

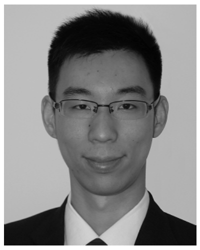
[1] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.
[2] Amazon.com. *Customer Success. Powered by the AWS Cloud.* [Online]. Available: http://aws.amazon.com/solutions/case-studies/, accessed May 2014.
[3] Amazon.com. *Amazon Elastic Compute Cloud (Amazon EC2).* [Online]. Available: http://aws.amazon.com/ec2/, accessed Oct 2014.
[4] J. F. Gantz, S. Minton, and A. Toncheva. (Mar. 2012). *Cloud Computing's Role in Job Creation.* [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*, vol. 7, no. 9, pp. 34–45, Sep. 1974.
[6] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.
[7] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, and Y. Bao, "CMD: Classification-based memory deduplication through page access characteristics," in *Proc. 10th ACM SIGPLAN/SIGOPS Int. Conf. VEE*, 2014, pp. 65–76.
[8] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems—Temporal isolation challenges due to shared resources," in *Proc. Workshop Ind.-Driven Approaches Cost-Effective Certification Safety-Critical, Mixed-Criticality Syst.*, 2013, pp. 1–6.
[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2011.
[10] Q. Lin, D. Tretter, J. Liu, and E. O'Brien-Strain, "Multimedia analysis and composition cloud service," in *Proc. 3rd Int. Conf. Internet Multimedia Comput. Service (ICIMCS)*, 2011, pp. 55–58.
[11] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active low-power modes for main memory," in *Proc. ASPLOS*, 2011, pp. 225–238.
[12] G. Jia, X. Li, J. Wan, L. Shi, and C. Wang, "Coordinate page allocation and thread group for improving main memory power efficiency," in *Proc. HotPower*, 2013, pp. 7:1–7:5.
[13] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," in *Proc. Linux Symp. (OLS)*, 2009, pp. 19–28.
[14] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, and Y. Bao, "CMD: Classification-based memory deduplication through page access characteristics," in *Proc. VEE*, 2014, pp. 65–76.
[15] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf. (ATEC)*, Sep. 2013, pp. 41–46.
[16] *KVM-Kernel Based Virtual Machine*. [Online]. Available: http://www.linux-kvm.org/page/Main_Page, accessed Sep. 2013.
[17] H. Yun, S. Gondi, and S. Biswas, "BWLOCK: A dynamic memory access control framework for soft real-time applications on multicore platforms," Univ. Kansas, Lawrence, KS, USA, Tech. Rep.
[18] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, Dec. 2002.
[19] P. Sharma and P. Kulkarni, "Singleton: System-wide page deduplication in virtual environments," in *Proc. 21st Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2012, pp. 15–26.
[20] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. ASPLOS XV*, 2010, pp. 129–142.
[21] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A system for energy efficient computing in virtualized environments," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2009, pp. 243–248.
[22] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, "Using OS observations to improve performance in multicore systems," *IEEE Micro*, vol. 28, no. 3, pp. 54–66, May/Jun. 2008.
[23] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," in *Proc. 15th Ed. ASPLOS*, 2010, pp. 335–346.
[24] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era," in *Proc. MICRO-44*, 2011, pp. 24–35.
[25] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. MICRO*, Dec. 2010, pp. 65–76.
[26] G. Jia, G. Han, J. Jiang, and J. J. P. C. Rodrigues, "PARS: A scheduling of periodically active rank to optimize power efficiency for main memory," *J. Netw. Comput. Appl.* [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2015.08.001
[27] G. Jia, G. Han, J. Jiang, and A. Li, "Dynamic time-slice scaling for addressing OS problems incurred by main memory DVFS in intelligent system," *Mobile Netw. Appl.*, vol. 20, no. 2, pp. 157–168, 2015.
[28] G. Jia, G. Han, L. Shi, J. Wan, and D. Dai, "Combine thread with memory scheduling for maximizing performance in multi-core systems," in *Proc. IEEE 20th ICPADS*, Dec. 2014, pp. 298–305.
[29] G. Jia, X. Li, C. Wang, X. Zhou, and Z. Zhu, "Memory affinity: Balancing performance, power, thermal and fairness for multi-core systems," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2012, pp. 605–609.

GANGYONG JIA (M'13) received the Ph.D. degree from the Department of Computer Science, University of Science and Technology of China, Hefei, China, in 2013. He is currently an Assistant Professor with the Department of Computer Science, Hangzhou Dianzi University, China. He has authored over 20 papers in related international conferences and journals. His current research interests are power management, operating system, cache optimization, and memory management. He has served as a Reviewer of *Microprocessors and Microsystems*.

GUANGJIE HAN (S'03–M'05) received the Ph.D. degree from the Department of Computer Science, Northeastern University, Shenyang, China, in 2004. He was with ZTE Company from 2004 to 2006, where he was a Product Manager. He was a Visiting Research Scholar with Osaka University from 2010 to 2011. He finished the work as a Post-Doctoral Fellow with the Department of Computer Science, Chonnam National University, Korea, in 2008. He is currently a Professor with the Department of Information and Communication System, Hohai University, China. He has authored over 130 papers in related international conferences and journals. He holds 55 patents. His current research interests are sensor networks, computer communications, mobile cloud computing, multimedia communication, and security. He is a member of Association for Computing Machinery (ACM). He has served as a Co-Chair for more than 20 international conferences/workshops and a TPC Member of more than 70 conferences. He received the ComManTel 2014 and Chinacom 2014 Best Paper Awards. He has served on the Editorial Board of up to 14 international journals, including the *Journal of Internet Technology* and the *KSII Transactions on Internet and Information Systems*. He has served as a Reviewer of more than 50 journals.



DAQIANG ZHANG (M'09–SM'14) received the joint Ph.D. degree in computer science from Shanghai Jiao Tong University and Hong Kong Polytechnic University. From 2011 to 2012, he held a post-doctoral position at Telecom SudParis, Institut Mines-Telecom, France. He is currently an Associate Professor with the School of Software Engineering, Tongji University. His research includes mobile computing, distributed computing, and wireless sensor networks. He has authored over 80 papers in major journals and international conferences in those areas. He has got papers published at the *ACM Transactions on Embedded Computing*, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Emerging Topics in Computing, the IEEE Transactions on Industrial Informatics, the *IEEE Wireless Communication*, the IEEE Network, the IEEE Systems Journal, the IEEE Wireless Communications, the *IEEE Communications*, the IEEE Communications Letters, ACM International Conference on Ubiquitous Computing, IEEE International Conference on Mobile Ad hoc and Sensor Systems, Global Communications Conference, Exhibition and Industry Forum, Ieee International Conference on High Performance Computing and Communications, International Conference on Parallel Processing, IEEE International Conference on Communications, and IEEE Wireless Communications and Networking Conference. He is a Senior Member of China Computer Federation. He got one most downloaded paper up to 2014 at*Mobile Networks and Applications* (ACM/Springer), one most viewed paper up to 2013 at *Pervasive and Mobile Computing* (Elsevier), and the best paper award from ACCV'09 and the IEEE UIC'12. He is an Editor of *Telecommunication Systems* (Springer), *European Transactions on Telecommunications* (Wiley), *KSII Transactions on Internet and Information Systems* (Korea Society of Internet Information) , and *New Review of Hypermedia and Multimedia* (Taylor & Francis). He was the Guest Editor of the IEEE Systems Journal, the IEEE Access, *Computer Networks* (Elsevier), the *Journal of Universal Computer Science*, and *Mobile Networks and Applications* (ACM/Springer).



LI LIU received the B.S. degree from Hohai University, Changzhou, China, in 2014, where he is currently pursuing the M.S. degree with the College of Internet of Things Engineering. His research interests include coverage and connectivity for wireless sensor networks.

LEI SHU (M'07) received the Ph.D. degree from the National University of Ireland, Galway, Ireland, in 2010. Until 2012, he was a Specially Assigned Researcher with the Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Japan. Since 2012, he has been with the Guangdong University of Petrochemical Technology, Maoming, China, as a Full Professor. Since 2013, he has been a Ph.D. Supervisor with the Dalian University of Technology, Dalian, China, and a Master Supervisor with the Beijing University of Posts and Telecommunications, Beijing, China. He has also been the Vice Director of the Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis with the Guangdong University of Petrochemical Technology. He is the Founder of the Industrial Security and Wireless Sensor Networks Laboratory. He has authored over 200 papers in related conference proceedings, journals, and books. He has an h-index of 25. His research interests include wireless sensor networks, multimedia communication, middleware, security, and fault diagnosis. He is a member of the IEEE Communication Society, the European Alliance for Innovation, and the Association for Computing Machinery. He received the 2010 IEEE Global Communications Conference and the 2013 IEEE International Conference on Communications Best Paper Awards. He served as a Co-Chair of more than 50 various international conferences/workshops, such as the IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), the IEEE International Conference on Communications (ICC), the IEEE Symposium on Computers and Communications (ISCC), the IEEE International Conference on Computing, Networking and Communication, and the International Conference on Communications and Networking in China. He also served/will serve as a Symposium Co-Chair of IWCMC 2012 and ICC 2012, a General Chair of Chinacom 2014 and the 2015 International Conference on Heterogeneous Networking for Quality, Reliability, Security, and Robustness, a Steering Chair of the 2015 International Conference on Industrial Networks and Intelligent Systems, and a Technical Program Committee Member of more than 150 conferences, including the IEEE International Conference on Distributed Computing in Sensor Systems, the IEEE International Conference on Mobile Ad Hoc and Sensor Systems, ICC, Globecom, the IEEE International Conference on Computer Communications and Networks, the IEEE Wireless Communications and Networking Conference, and ISCC. He currently serves as the Editor-in-Chief of the *European Alliance for Innovation Endorsed Transactions on Industrial Networks and Intelligent Systems*, and the Associate Editor of a number of renowned international journals.