Personal and Ubiquitous Computing (2018) 22:561–573
https://doi.org/10.1007/s00779-018-1118-5

ORIGINAL ARTICLE

Dynamic cloud resource management for efficient media applications in mobile computing environments

Gangyong Jia[1] · Guangjie Han[2] · Jinfang Jiang[2] · Sammy Chan[3] · Yuxin Liu[4]

Received: 20 November 2017 / Accepted: 10 December 2017 / Published online: 17 May 2018

Abstract

Single-instruction-set architecture (Single-ISA) heterogeneous multi-core processors (HMP) are superior to Symmetric Multi-core processors in performance per watt. They are popular in many aspects of the Internet of Things, including mobile multimedia cloud computing platforms. One Single-ISA HMP integrates both fast out-of-order cores and slow simpler cores, while all cores are sharing the same ISA. The quality of service (QoS) is most important for virtual machine (VM) resource management in multimedia mobile computing, particularly in Single-ISA heterogeneous multi-core cloud computing platforms. Therefore, in this paper, we propose a dynamic cloud resource management (DCRM) policy to improve the QoS in multimedia mobile computing. DCRM dynamically and optimally partitions shared resources according to service or application requirements. Moreover, DCRM combines resource-aware VM allocation to maximize the effectiveness of the heterogeneous multi-core cloud platform. The basic idea for this performance improvement is to balance the shared resource allocations with these resources requirements. The experimental results show that DCRM behaves better in both response time and QoS, thus proving that DCRM is good at shared resource management in mobile media cloud computing.

Keywords Mobile multimedia cloud computing · VM placement · Dynamic cloud resource management · Response time

1 Introduction

Multi-core has been popular due to its advantages in power [1] and more modest computational small cores have replaced the single and powerful processor [2]. All cores of symmetric multi-core have identical performance; therefore, all threads can run on arbitrary cores. However, single-instruction-set architecture (ISA) heterogeneous multi-core processors (HMPs) can have better power consumption, since these are better in catering the needs of diverse workloads [3–5]. Therefore, HMP adoption is promising for all aspects of the Internet of Things (IoT).

Cores in the single-ISA HMP have the same ISA, but different features, size, speed, and power consumption [6, 7]. Therefore, for a normal HMP, there are some fast/big cores as well as small cores. Sequential threads can be assigned to fast cores, while parallel running threads are assigned to small cores, thus improving performance while simultaneously decreasing power consumption. These advantages make HMP a promising platform for cloud computing, especially for multimedia mobile cloud computing.

In multimedia mobile cloud computing, multiple virtual machines (VMs), which provide many different services such as video transcoding, Hi-Definition video, image/video retrieval, streaming, video rendering, and media analytics, will run on a single physical server. These VMs are providing independent services for virtualization technology [8, 9]. All these resources necessary for VMs are managed by the

✉ Guangjie Han
hanguangjie@gmail.com

Gangyong Jia
gangyong@hdu.edu.cn

Jinfang Jiang
jiangjinfang1989@gmail.com

Sammy Chan
eeschan@cityu.edu.hk

Yuxin Liu
13906113387@163.com

1 The Department of Computer Science, Hangzhou Dianzi University, Hangzhou 310018, China

2 The Department of Information and Communication System, Hohai University, Jinling North Road, No. 200, Changzhou 213022, China

3 The Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China

4 The Jiangsu Xin Zhongtian Plastic Industry Co., Ltd., Jiangsu, China

hypervisor or virtual machine monitor (VMM). Therefore, it is central for the hypervisor or VMM to provide an efficient management policy for VMs [10], providing good QoS of the VMs for users.

There are some challenges for resource management policies:

Firstly, more and more VMs are collocated on one physical server to decrease costs; therefore, both the interference and contention are severely increased. Currently running VMs compete with each other for resources to provide services. VMs will slow down to compete for resources. This is a known problem; however, for a HMP based cloud platform, the problem will be more severe. Since fast cores are more sensitive to contention, fast cores decrease more in performance. Consequently, fast cores cannot accelerate sequential threads. The advantages of HMP will disappear, wasting the HMP. In addition, all VMs memory streams are interleaved to interfere with each other in the HMP system; both original spatial locality and bank level parallelism of individual VM will be destroyed. Thus, performance will decrease severely [11, 12], and in bandwidth, interference will also occur.

Secondly, different services have heterogeneous resource demands, which will result in unbalanced resource utilization, thus decreasing efficiency. There are many different services in media cloud computing, rendering services, video transcoding services, and streaming services. Most these media services have different resource demands. Some services are CPU-bound, while others are memory-bound. Therefore, if most the same type VMs are running on one core, and most CPU-bound VMs running on the same core, and consequently, the CPU resource is overloading, while other resources are wasted. Thus, the entire system performance will severely decrease due to resource wasting.

Thirdly, resource demand is much advanced to the physical increase; however, this problem cannot be solved.

Therefore, improving QoS, reducing interference, and balancing resource utilization will be a good policy. To reduce interference, resources partition is a feasible means; to balance resource utilization, VMs placement policies can be used to avoid overloading and wasting. To improve the response time, improve QoS, and maximize the effectiveness of the HMP system in media mobile cloud computing, we propose the Dynamic Cloud Resource Management (DCRM), which combines resource-aware VM allocation with dynamically and optimally partitioned shared resources. The key idea is to profile the resource characteristics of both cores and VMs, estimate their requirements for shared resources, and direct our resource-aware VM placement and resource partitioning based on this estimation. Experimental results demonstrate that DCRM can improve the response time and QoS; moreover, it also enhances power efficiency and throughput for HMP based media mobile computing.

In summary, this paper has the following contributions:

We propose dynamic cloud resource management to improve response time and maximize effectiveness for HMP-based media mobile cloud computing to satisfy QoS. DCRM combines resource-aware VM allocation with dynamically and optimally partitioned shared resources. Our resource-aware VM allocation places VMs at run-time, without prior knowledge of VM demand.

We tackled the challenges of VM resource management in HMP-based media mobile cloud computing. Experimental results show that our DCRM performs well and satisfies QoS.

The remainder of this paper is organized as follows. Section 2 elaborates on essential background and related work. Section 3 analyzes research motivations. Section 4 explains our DCRM platform. Section 5 describes the experimental methodology and Section 6 presents the results of our experiments. Finally, Section 7 concludes the paper.

2 Background & related works

2.1 DRAM system

Memory subsystem organization is shown in Fig. 1. The subsystem consists of multiple levels, memory controller (MC), Dual Inline Memory Module (DIMMs), memory channels, memory banks, DRAM chip, arrays, and other items. When the last level cache has a miss, memory will be accessed. To provide parallel access, multiple channels exist, which can be independently accessed for disjoint physical memory regions [13].

One channel may connect to multiple DIMMs, and every DIMM may connect to multiple DRAM chips. The memory access will finally reach the DRAM chips. Rank is the subset of the DRAM chip, which participates in each access. The number of bits produced once by each chip determines the number of chips in a rank. Up to 16 chips can be in one DIMM, and these chips are organized into 1–4 ranks.

Multiple banks (up to eight) are in every DRAM chip, and each bank has multiple two-dimensional memory arrays. The entire multi-KB row will be transferred to the row buffer after the array has been accessed, which is called activation or row opening. When the row has been transferred to the row buffer, all columns in the row can be read/written. Until another row is activated, the row needs to be written back to the array.

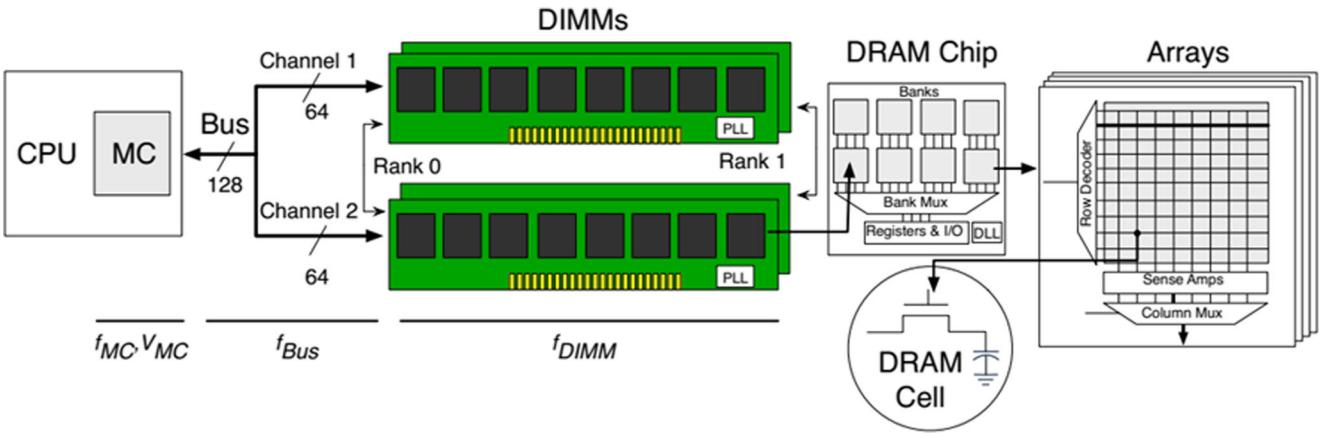

Fig. 1 Organization of a modern memory subsystem

2.2 Related work

A number of related studies have been published in dynamic cloud resource management.

Task Scheduling schedules' tasks among VMs to balance resources (CPU, memory, and bandwidth) to improve system performance [14–18]. The task scheduling problem is NP-complete and some researches have studied several sub-optimal solutions.

Live migration transfers the entire VM from the host machine to a new destination host at runtime, without preempting its execution [19]. Live migration has two steps: firstly, pre-copying the VM status to the destination host while the VM is running on the host machine; secondly, start the VM in the destination machine, and then stop the VM in the host machine. Live migration is a basic tool for current cloud computing that minimizes resource usage when some servers are in low utilization.

Load balance: If the resources in the cloud are used in an unbalanced way, some resources will be wasted due to waiting. To improve resources utilization, load balance will be adopted at times. VM migration is used to move VMs among the running hosts to rebalance resource utilization, migrating some VMs from heavily loaded hosts to lightly loaded hosts [20, 21].

On-demand allocation chooses the best host for the VM according to both VM demands and provided host [22, 23]. Some theoretical models exist to guide allocation. These allocations are theoretical methods, and several problems exist due to lack of information.

Resources partition: In cloud computing, VMs are sharing resources of servers. Therefore, VMs are competing with each other. To reduce competition, resources can be partitioned into parts, thus avoiding competition between different parts [24, 25].

In this paper, based on the special applications of mobile media cloud computing, one of the most popular demands is the aim to decrease the response time, while improving user experience. This is a practical problem for a special actual framework. We combined VM allocation with resource partition to improve the response time for all VMs in the cloud. Moreover, we used some specialty of mobile media cloud computing to address the problem.

3 Motivation

In this section, we discuss existing methods that deal with the challenges of response time, power efficiency, and performance for modern HMP-based media mobile cloud computing.

3.1 Profiling of VM placement

Resources utilization is dependent on the VM placement in cloud computing, especially in HMP based media mobile cloud computing. Most of the same type VMs placed in the same core will waste resources, prolonging the response time. If this happens, the QoS is hard to satisfy. Many services are provided in mobile media cloud computing, such as video transcoding, Hi-Definition video, image/video retrieval, streaming, video rendering, media analytics, and sharing and delivery. Most services have different resource demand. For example, video transcoding service is CPU-bound, and video-streaming services are bandwidth-bound. Collocate some VMs, which are of the same type, in one core will prolong the response time because one or more resources are overloading. Whenever two or more video transcoding services are collocated in one core, the CPU is overloading; therefore, all these services will be low

QoS. However, place different types of VMs, such as one for video transcoding and others for video streaming, will improve the response time for all resources, balancing them without overloading. The average normalized response time is shown in Fig. 2, which runs different services and different numbers of VMs on one core. The x-axis provides different configurations, vt represents video transcoding, vs represents video streaming, vr represents video retrieval, and vd represents video rendering. We used the n-name notation to denote n VMs of the services running on the core. In Fig. 2, 2-vt represents 2 video transcoding services running on the core. The result suggests that the response time will be improved if we placed heterogeneous services on one core. The results will be better when more VMs are running on one core/server.

3.2 Interference profiling

More and more VMs are collocated on one physical server to decrease costs; therefore, both the interference and contention are severely increased. All VMs have good local independence; therefore, exhibit good row buffer hit rate. However, with parallel running VMs, the locality is significantly reduced and the row buffer hit rate is also decreased. Therefore, system performance meets challenges. The row buffer hit rate is shown in the Fig. 3 and the more VMs exist in a system, the worse the hit ratio will be. Row buffer hit rate is shown of the y-axis, and the x-axis illustrates different running configurations. n-T-m-VM represents that there are m VMs with n threads on the server, and every VM has n/m threads.

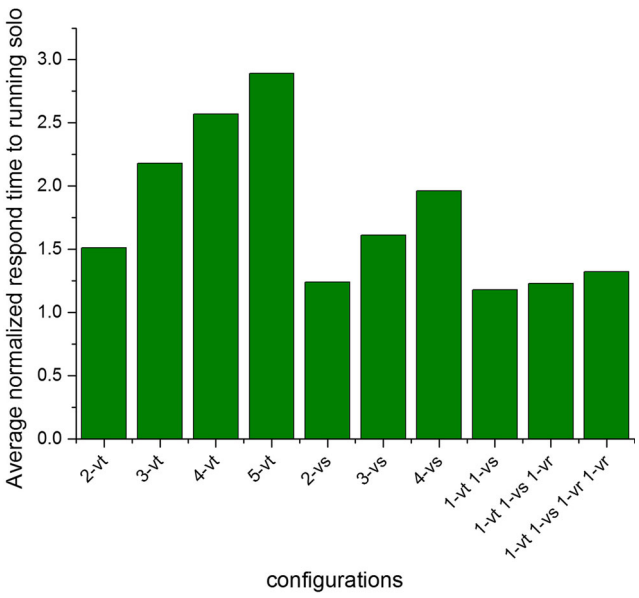

Fig. 2 The average normalized response time with different services and different numbers running on one core

The interference is mainly due to three aspects:

Hypervisor interfere VMs. The hypervisor presents the virtual operating platform for the guest operating systems; furthermore, it manages the execution of the guest operating system. Therefore, multiple instances exist where the operating systems share virtualized hardware resources. The hypervisor is responsible for resources management. Therefore, VMs need to request the hypervisor to allocate physical resources, such as the memory resource, during their lifetime. In the process of memory allocation, the hypervisor may disturb the origin memory access stream of VMs, which is one aspect of memory interference.

The above situation is shown in Fig. 3. 1-T represents that one thread running on the operating system without the 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 generated by the interference-free hypervisor.

Moreover, to analyze the interference effect from hypervisor in detail, we counted the percentage of VM row buffer misses caused by the hypervisor. Figure 4 shows the percentage of VM row buffer misses. The x-axis represents the different media services running on VM. This figure demonstrates that the hypervisor significantly affected row buffer misses.

VM interfere applications running on it. Applications will frequently invoke system calls to get the service of the guest OS before finishing. Therefore, the states will be frequently switched between kernel and user. Switch to the kernel states will happen when requested, and a switch back to the user states will happen after the request finished. For a simple service, the kernel

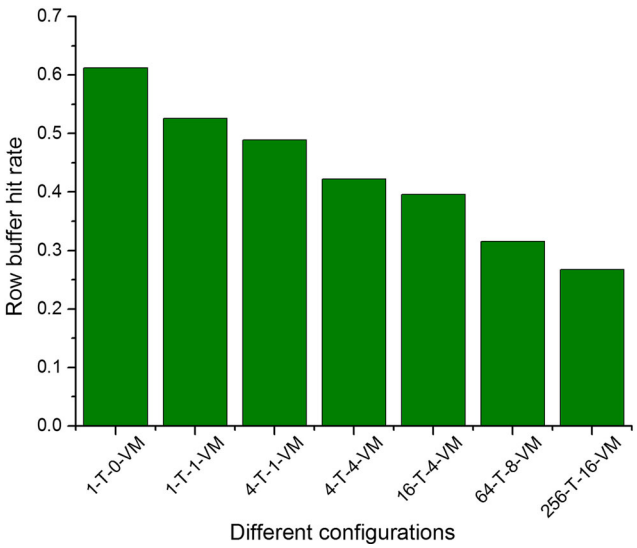

Fig. 3 Row buffer hit rate under different running configurations

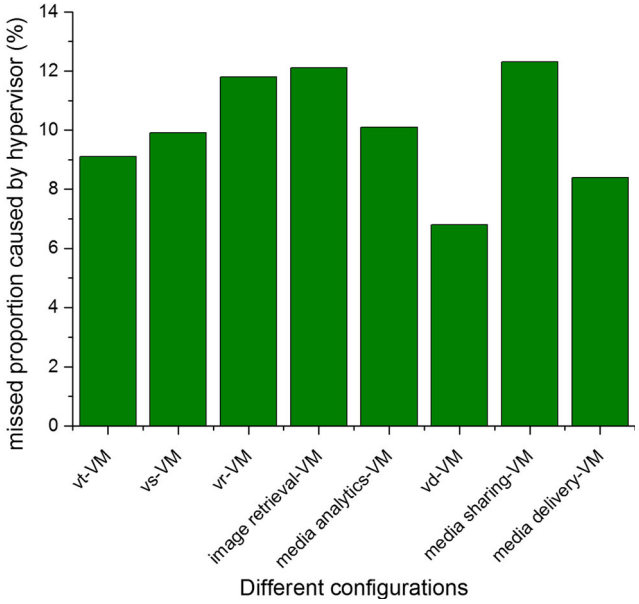

Fig. 4 The percentage of VM row buffer miss

will finish quickly. Although the process is short, most contents in the row buffer will be written back. Moreover, the system call will be frequently invoked [26] and the cost is not negligible. The percentage row buffer miss caused by the guest OS is plotted in Fig. 5; the x-axis shows the different benchmarks. Here, we ran different benchmarks on VM to count the percentage misses caused by invoking system calls and other guest OS interference. The results are clear that the guest OS contributes most row buffer misses to applications.

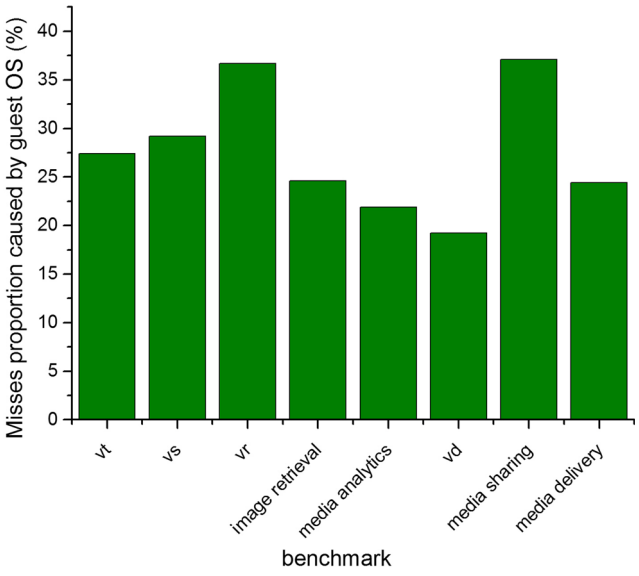

Fig. 5 VM row buffer misses proportion caused by hypervisor in different configurations

Interference among VMs and threads on one VM. VMs and threads on one VM running concurrently contend shared memory in both chip multiprocessors (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 spaces, respectively. The results of Fig. 3 show 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 the row buffer hit rate. Briefly, one VM is better than four VMs that are simultaneously running in row buffer miss rate. Moreover, as the threads number of one VM increases, 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 hit rate deteriorates strongly. Interference among both VMs and threads need to be alleviated for memory performance improvement.

3.3 Profiling of power efficiency

Sequential threads can be assigned into fast cores, while parallel running threads are assigned into small cores, which improve performance, while simultaneously decreasing power consumption. These advantages suggest HMP as a promising platform for cloud computing, especially for mobile multimedia cloud computing. However, the resources contention among simultaneously running VMs will restrict its effectiveness. Figure 6 shows the power efficiency. PSO represents the system with fast cores run CPU-bound VMs and small cores run memory-bond VMs; however, all VMs share resources. OPT represents VM run solo, without contention. The workloads of mix0 to mix9 choose four media services randomly. The results explain that the HMP power efficiency can be severely improved if the resources management had a good policy.

3.4 Analysis of these challenges

All these three challenges arise from two aspects:

Balance. The first aspect is the balance policy in the current system, which ignores the characteristic of the load in one core/server, and only considers the number of VMs or CPU utilization. Therefore, the heterogeneity of the media services can be utilized to place VMs and to improve response time.

Share. Sharing resources is most challenging for the contention and interference. Some resources, such as cache, memory, and bandwidth, are shared among all simultaneously running VMs.

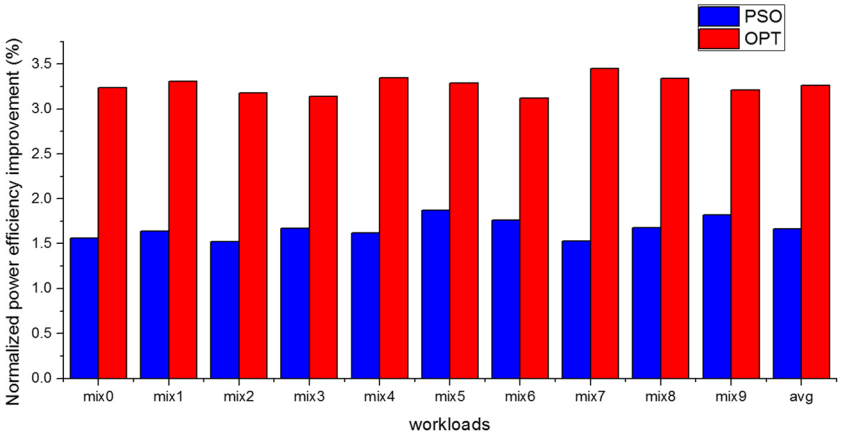

Fig. 6 Normalized power efficiency improvement

3.5 Insights based on the analysis

Limited memory bank requirement Although memory bank can improve parallelism, the necessary amount of memory banks one VM requires is limited. Due to memory dependency, high cache hit rate, and limited number of MSHRs, a VM is unable to generate sufficient concurrent memory requests [27]. However, most modern systems always spread all VMs' memory banks across the entire memory to access all banks to take advantage of bank-level parallelism, which is excess and suffers from memory interference. Therefore, we can prevent sharing among parallel running VMs in the shared environment by partitioning memory banks for each VM, thus eliminating inter-thread bank conflicts. Moreover, as the technique of OS page-coloring is well-known for cache partitioning [28], the technique can also be used to partition memory banks. Therefore, we can simultaneously partition both cache and memory banks, adopting a page-coloring technique to prevent sharing among parallel running VMs in a shared environment.

VM's requirement for shared resources To estimate the required resources of each VM, we need to analyze the characteristics of both VM and core. Firstly, we defined a VM characteristic using three components: memory intensity [29], row-buffer locality [30], and bank level parallelism [30]. Memory intensity describes the frequency of both memory requests and misses in the last-level cache. Row-buffer locality describes the locality where a VM hits the row buffer. Bank-level parallelism describes the number of different banks where a VM independently accesses in parallel. Secondly, we defined a core's characteristic using three components: CPU frequency, cache size, and out-of-order window size. Therefore, we used the six components to estimate the shared resources requirements for every VM. Based on the estimation, we can allocate resources to prevent unfairness.

Core/server's current load To place the newly created VM, we need to know the cores/servers' load, and then choose the most suitable one for the VM. With the count of the hypervisor at real-time, the load information can be used to allocate VMs.

4 Dynamic Cloud Resource Management (DCRM)

In this section, we first provide an overview of DCRM. Then, we introduce the profiling of VM's and core's characteristics, respectively. Subsequently, we introduce VM allocation policy. Finally, we introduce the process of the dynamic cloud resource partitioning.

4.1 Overview of DCRM

DCRM contains four parts. First, resource demand for each VM is defined, mainly characterizing the demands for the CPU, memory, and bandwidth resources. Second, performance characteristics for each core are defined, characterizing the performance of each core in HMP. Different performance cores have different sensitivities to resources. Fast cores are more sensitive, small cores are less sensitive. Core performance is the factor to partition resources. Third, VM placement policy is based on the load information and the resource demand of new created VMs, placing VMs to balance resources utilization. Fourth, partition resources based on both resource demand and performance characteristic of each core to reduce interference and speed up an already fast core. The DCRM framework is shown in Fig. 7.

4.2 Resource demand for each VM

For resource management, one of the most important points is to predict the resource demand for each VM. In this paper,

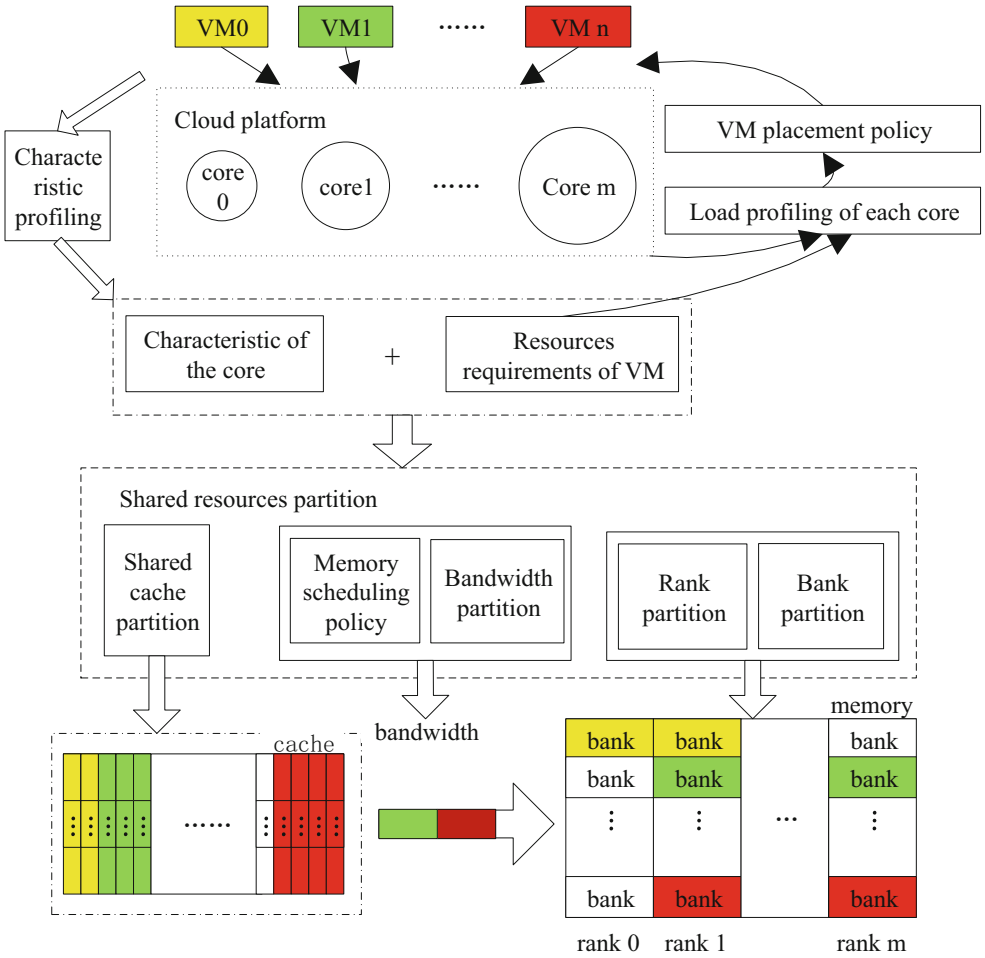

Fig. 7 The framework of DCRM

we considered three resources: CPU, memory, and bandwidth, which are the most considered in most current researches. To characterize the resource demand, we adopted three parameters: memory intensity, which reflects CPU and bandwidth demand; row-buffer locality, which reflects locality; and bank level parallelism, which reflects parallelism.

Memory intensity distinguishes CPU-bound VMs with memory-bound VMs. Last-level cache misses can be used to reflect the memory intensity; therefore, last-level cache misses per thousand instructions (MPKI) was adopted in this paper to represent memory intensity. If the VM has more MPKI, the VM demands more memory and bandwidth resources. Otherwise, the VM has low MPKI, and the VM demands low memory and bandwidth resources; moreover, for these VMs, more memory and bandwidth allocation cannot improve their QoS. However, they are sensitive to responses from memory, which significantly affects their QoS. Therefore, for resource management, low memory intensity VMs require high priority to access memory and bandwidth, and high memory intensity VMs can tolerate a longer waiting time to access memory, but need more memory resources.

Row buffer locality distinguishes locality for high memory intensity VMs. If one VM has high locality, the row buffer hit ratio will be high. In this paper, we adopted the row buffer hit rate (RBH) to reflect the row buffer locality. Although memory banks can be used to hide memory access latency, for the high row buffer locality VMs, allocating more memory banks will not improve performance, which is insensitive to the numbers of memory banks. Sixteen memory banks are sufficient for high row buffer locality VMs. However, they are more sensitive to the interference, which will disturb their locality and reduce the row buffer hit ratio.

Bank level parallelism distinguishes the sensitivity to bank parallelism. Banks can be accessed in parallel to overlap memory access and improve performance. However, the sensitivity for VMs are different.

Moreover, for mobile media cloud computing to provide media services, the services in the cloud are hardly changing. Therefore, we can catch the three characteristic components for each service via simulation. Based on the simulation results, every service VM can predict its three characteristic components online.

4.3 Performance characteristic for each core

In the HMP system, cores have different performance characteristics, because cores have different hardware resources, such as different CPU frequency, different size of cache, different size of out-of-order window, and different branch prediction technique. Therefore, the core performance is directly related to the hardware resources. In this paper, we chose three hardware resources to represent the performance characteristics of each core:

CPU frequency is the most related to the performance of the core. The higher the frequency of the core, the better its performance will be.

Cache size is the storage size in the core. The more private cache size is allocated for the core, the more hit ratio will be in the cache, which will reduce the average time to get the instructions/data. This is important for the system.

Out-of-order window size is the size of the parallel launch instructions in an out-of-order speculative CPU. The window size is finite and new instructions can be launched only after instructions leave the window. If the window size increases, more instructions can be launched in parallel. The instructions can be committed faster. Furthermore, the performance of the core may be improved.

4.4 Resource-aware VM placement

In this paper, the key job of our resource-aware VM placement was to choose one core for the newly created VM. Based on both the VM's characteristics and the core load, our resource-aware VM placement can balance resources utilization. If the load of one core is mainly bandwidth, the newly created VM of CPU-bound can place on this core; otherwise, if the load of one core is mainly CPU computing, the newly created VM of memory-bound or bandwidth can be placed on this core. Algorithm 1 provides our resource-aware VM placement. In the algorithm, we partitioned resource utilization of each core into four levels, level_0 to level_3. If one core the resource utilization of one core falls below 25%, we assigned a resource utilization of level_0; otherwise, if the resource utilization of one core increases above 25%, but remains below 50%, we assigned a resource utilization of level_1; otherwise, if one core's resources utilization is above 50%, but remains below 75%, we assigned a resource utilization of level_2; otherwise, we assigned a resource utilization of level_3. Moreover, in this paper, we mainly considered three resources: CPU, memory banks and bandwidth. Therefore, for each resource, we have four levels: level_0_CPU, level_1_CPU, level_2_CPU, level_3_CPU, level_0_Mem, level_1_Mem, level_2_Mem, level_3_Mem, level_0_Bd, level_1_Bd, level_2_Bd, and level_3_Bd. We also partitioned resources demand into four levels, level_0_mpki, level_1_mpki, level_2_mpki, level_3_mpki, level_0_rbh, level_1_rbh, level_2_rbh, and level_3_rbh. 0 <= level_0_mpki < 10, 10 <= level_1_mpki < 30, 30 <= level_2_mpki < 60, 60 <= level_3_mpki; 0 <= level_0_rbh < 25%, 25% <= level_1_rbh < 50%, 50% <= level_2_rbh < 75%, and 75% <= level_3_rbh <= 100%. Therefore, the algorithm is required to find one core for the new created VM, for which the core can satisfy the VM's resources demand with balance. Firstly, the resources demand (t, k) was determined, which MPKI $\in$ level_t_mpki, RBH $\in$ level_k_rbh. Next, determine resources utilization for each core, (m, n, r), which $C_i \in$ level_m_CPU, $M_i \in$ level_n_Mem, $B_i \in$ level_r_Bd. Finally, that core was chosen that yields a minimal value of t + k + m + n + r. A minimal value of t + k + m + n + r means that the core is one of the most suitable for the new created VM.

Algorithm 1 Our resource-aware VM placement algorithm

Definition:
C_i: CPU utilization of the core i
M_i: the occupied memory banks proportion of the core i
B_i: the occupied bandwidth proportion of the core i
MPKI: the memory intensity of the new created VM
RBH: the row-buffer locality of the new created VM
N: the total cores in the cloud computing
Output:
cn: the core number of new created VM will place
Our resource-aware VM placement:
begin
cn = 0;
i = 0;
temp = ∞;
determine the resources demand (t, k), which MPKI $\in$ level_t_mpki, RBH $\in$ level_k_rbh;
While(i < N)
    determin (m, n, r), which $C_i \in$ level_m_CPU, $M_i \in$ level_n_Mem, $B_i \in$ level_r_Bd;
    If t + k + m + n + r < temp
        temp = t + k + m + n + r;
        cn = i;
return cn;
End

4.5 Process of the dynamic cloud resource partitioning

4.5.1 Memory bank partition

VMs are partitioned into three types according to the resource demands of the VM and its running core performance:

First type, CPU-bound VM. This type of VMs is sensitive to the CPU resources, but insensitive to memory and bandwidth resources. Moreover, this type of VMs is sensitive to the memory access time.

Second type, memory-bound with low row buffer locality VM. This type of VMs is sensitive to memory banks for parallelism access.

Third type, memory-bound with high row buffer locality VM. This type of VMs is less sensitive to memory banks, but sensitive to interference.

Different to a SMP system, the type partition is related to core performance. For example, if one VM demand is represented by ($MPKI_i$, RBH_i), it runs on the core. Furthermore, the core performance is represented by (F_i, C_i, W_i), F_i represents CPU frequency, C_i represents cache size, and W_i represents out-of-order window size. We defined a factor to reflect the effect from the core performance, $T_1 * F_i + T_2 * C_i$, where $0 < T_1, T_2 < 1$, and $T_1 + T_2 = 1$. Therefore, VM demand is changed to ($MPKI_i * (T_1 * F_i + T_2 * C_i)/\Sigma (T_1 * F_k + T_2 * C_k)$, $RBH_i * W_i/\Sigma W_k$) after adding the core performance effect.

In this paper, we partitioned VMs into three types based on the (MPKI, RBH) of the VM, which is added to the core performance. If $MPKI < MPKI_t$, it belongs to CPU-bound VM; else if $RBH < RBH_t$, it belongs to memory-bound with low row buffer locality VM; otherwise, it belongs to memory-bound with high row buffer locality VM.

We partitioned memory banks according to the VM type to reduce interference. For memory-bound VMs with high row buffer locality, we allocated16unique memory banks; for memory-bound VMs with low row buffer locality, we allocated one of 16 bank groups for sharing among same type VMs; for CPU-bound VMs, they can use all memory banks. Our memory bank partition policy is shown in Algorithm 2.

4.5.2 Shared cache partitioning

The address mapping policy for the 32 bits CPU is shown in Fig. 8. The total 32 bits are partitioned into various functions, bank numbers, rank numbers, column numbers, and row numbers. In this paper, we mainly considered bank numbers. As shown in Fig. 8, 5 bank bits were used (bit 13, 14, 15, 21, and 22), which means that 32 different banks were present in the system. Moreover, each page in the memory belonged to a unique bank.

A shared 8MB last level cache is a 16-associate set. Every physical address is referred to a unique cache set, which is determined by the bits 6–18 of the physical address. Moreover, the bits 12–18 of the physical address are cache coloring. Furthermore, the bits 13–15 are used for both cache coloring and bank coloring. Therefore, the shared last level cache can be partitioned into 8 groups, which are independent. Based on the address mapping mechanism, partition shared last level, and memory banks can be simultaneously implemented.

Algorithm 2 Our memory bank partition policy

Definition:
N: core number
M: memory banks number
FT: CPU-bound VM number
ST: memory-bound with low row buffer locality VM number
TT: memory-bound with high row buffer locality VM number
Our memory bank partition policy:
begin
If FT==N
  Allocate M/N banks for every VM;
Else if FT==0
  Allocate M/(ST+TT) banks for every memory-bound with high row buffer locality VM;
  If M/(ST+TT) < 16
    Each two memory-bound with low row buffer locality VMs share M/(ST+TT)*2 banks;
  Else
    Allocate M/(ST+TT) banks for every memory-bound with low row buffer locality VM;
Else
  Allocate M/(ST+TT) banks for every memory-bound with high row buffer locality VM;
  All CPU-bound VMs can access all banks allocated for memory-bound with high row buffer locality VMs;
  If M/(ST+TT) < 16
    Each two memory-bound with low row buffer locality VMs share M/(ST+TT)*2 banks;
  Else
    Allocate M/(ST+TT) banks for every memory-bound with low row buffer locality VM number;
End

4.5.3 Memory bandwidth partitioning

In addition to memory banks and shared last level cache, bandwidth is another shared important resource. Bandwidth affects the QoS, especially for bandwidth-bound VMs. In this paper, we managed bandwidth based on fairness.

If N VMs are running in parallel in a system, which uses VM_1, VM_2, ..., VM_N to represent, and each VM needs Bi bandwidth, we will allocate bandwidth according to each demand. If the total bandwidth resource is B, the key for the memory bandwidth partition is to characterize

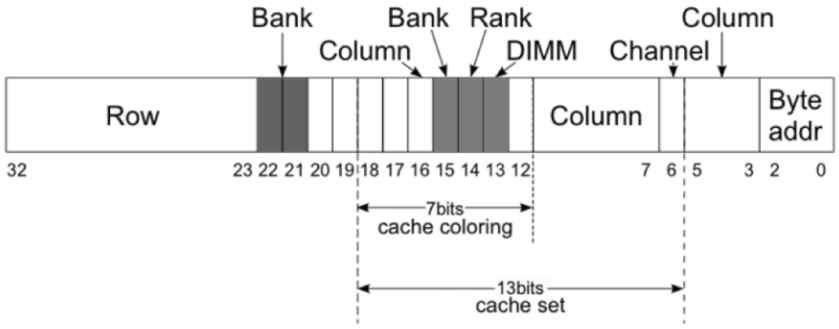

Fig. 8 Address mapping policy

the demand for the bandwidth. However, we can receive committed instructions, CI, and last level cache misses, CM, for each VM online. The performance management unit (PMU) records these events at runtime; therefore, we can read these online.

Equation (1) shows the partition policy, CI_i represents the committed instructions of VM_i, CM_i represents the last level cache misses of VM_i, 1<=i, j<=N. The partition policy partitions bandwidth in proportion to each VM demand. Equation (2) illustrates the restriction, where B_i represents the allocated bandwidth for VM i.

$$(CI_i + B_i)/(CI_i + CM_i) = (CI_j + B_j)/(IN_j + CM_j) \quad (1)$$

$$(B_1 + B_2 + ... + B_N) = B \quad (2)$$

Based on (1) and (2), the total bandwidth can be partitioned for VMs, while sharing fairly.

4.5.4 Memory scheduling

Resource partition will reduce interference to improve QoS; however, it cannot avoid some interference among VMs. For example, the memory access from fast core will be prolonged for many more memory accesses from small cores. To maximize the QoS, the CPU-bound VMs can be sped up. Memory scheduling is integrated into our DCRM.

CPU-bound VMs has higher priority in accessing memory due to an increased sensitivity to the memory access time. Row buffer hit memory accesses have higher priority if memory accesses stem from the same type VMs.

5 Experimental setup

Our experiments were conducted on two 2.0 GHz Intel Xeon E5504 processors with EPT enabled. There were four physical cores in each E5504 processor, and the Hyper-Thread was disabled. Each processor has 3-level caches, both L1 instruction and data cache are 32 KB, L2 cache is 256 KB. Both L1 and L2 are private for each core. However, L3 cache is 16-way 4 MB, shared by all four cores in each processor. All cache block sizes are 64-Byte in the experiment. There is 8 GB physical memory capacity with one dual-ranked of DDR3-800 MHz. The operating system is Ubuntu-12.04 with Linux kernel 3.6.10. DCRM was implemented within the operating system. QEMU [31] with KVM [32] (qemu-kvm-1.2.0) was adopted to support guest VMs. One virtual CPU is configured for each guest VM. Four VMs are boosted in parallel as our default configuration. Moreover, 8 VMs are boosted in parallel for further evaluation. 64-bit Linux-10.10 with Linux kernel 2.6.32 was used for the guest operating system. The following workloads, video transcoding, Hi-Definition video, image/video retrieval, streaming, video rendering, media analytics, sharing and delivery, inside guest VMs are chose to run.

6 Experimental results

In this section, we first evaluated the response time improvement. Then, we analyzed the performance improvement. In this paper, all the results show the average of ten repetitions.

6.1 Response time analysis

Response time is directly related to the QoS of the system, particularly for mobile media cloud computing. Therefore, in this paper, we use response time as the QoS parameter. Shorter response time will improve the QoS. The average normalized response time has been shown in Fig. 9. The x-axis provides the simultaneously running VMs. It is obvious that DCRM needs less response time, which means that DCRM behaves better in QoS.

If 8 VMs are created in the server, which represents one VM that runs on one core for the server had 8 cores. From the VM placement side, DCRM is identical to the default policy; however, DCRM is better in response time. Therefore, the improved response time is from the reduced interference.

To evaluate the resource-aware VM placement part of DCRM, we check the services running on every core when 32 VMs are created. Tables 1 and 2 show the

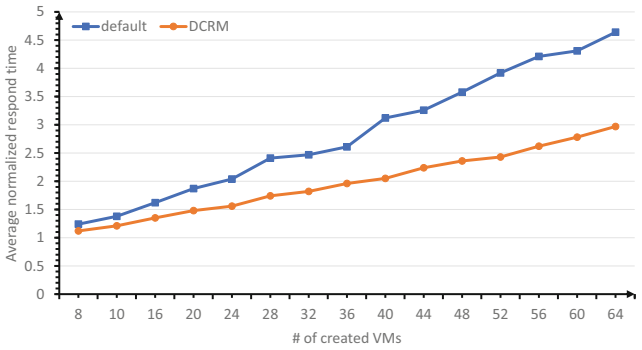

Fig. 9 The average normalized response time that we create from 8 to 64 VMs running media services

services of each core using DCRM and default, respectively. Each core almost runs different services using our DCRM policy, which is good for resource utilization to improve response time. However, with the default policy, each core always runs the same services, which will lead to resource overuse, consequently forming a bottleneck, thus prolonging response time.

Figure 10 shows the hardware utilization on core 0 when the server boots different number VMs. When there are 8 VMs in the server, core 0 has only one VM, the VM is the service of video transcoding both using default and our DCRM. Therefore, the hardware utilization is almost identical for both default and our DCRM. However, after more VMs are booted, core 0 is running more VMs, from 2 to 8. Furthermore, the services running on core 0 are different when using different policies. When using the default policy, the VMs of almost all video-transcoding services were running on core 0. However, using our DCRM, the VMs are running different services on core 0. Therefore, the resource-aware VM placement of DCRM can balance hardware utilization to avoid over usage, which improves the system response time.

Table 1 Services of each core using DCRM

Core	Transcoding	Streaming	Retrieval	Rendering
Core0	√	√	√	√
Core1	√	√	√	√
Core2	√	√	√	√
Core3	√	√√	√	√
Core4	√	√	√	√
Core5		√	√√	√
Core6	√√	√	√	
Core7	√		√	√√

Table 2 Services of each core using default

Core	Transcoding	Streaming	Retrieval	Rendering
Core0	√√√	√		
Core1	√√√	√		
Core2	√	√√	√	
Core3	√	√√	√	
Core4		√	√√	√
Core5		√	√√	√
Core6			√√	√√
Core7				√√√√

6.2 Performance analysis

In this paper, we used Instructions Per Cycle (IPC) to measure each VM speedup. To measure the whole parallel running system, we used the parameter-weighted speedup, which is given by (3). The IPCi represents the IPC of VMi; weighted speedup equalizes the contribution of each VM to the whole system by dividing the instructions executing on each VM by its natural offer rate if run alone; weighted speedup is a fair measure of real work done in multi-core systems.

$$weighted\_speedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} \tag{3}$$

Figure 11 demonstrates the normalized performance improvement in different configurations with our DCRM, from booting 8 virtual machines to 64 virtual machines. The

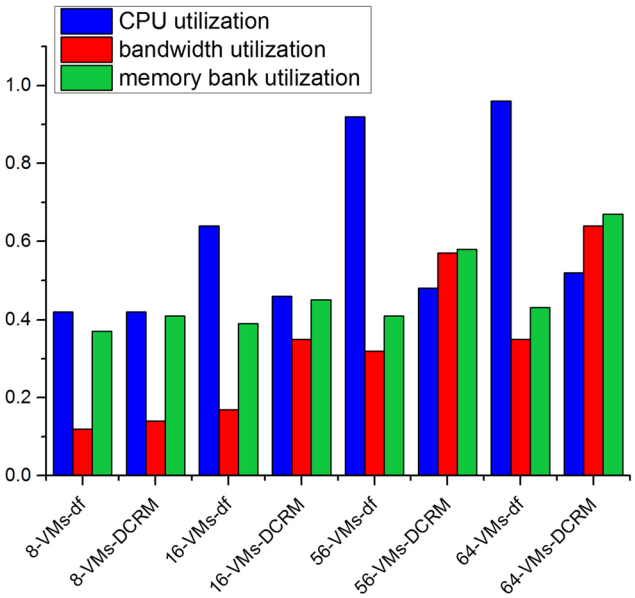

Fig. 10 The situation of some hardware utilization on core 0 when the server boots different number VMs

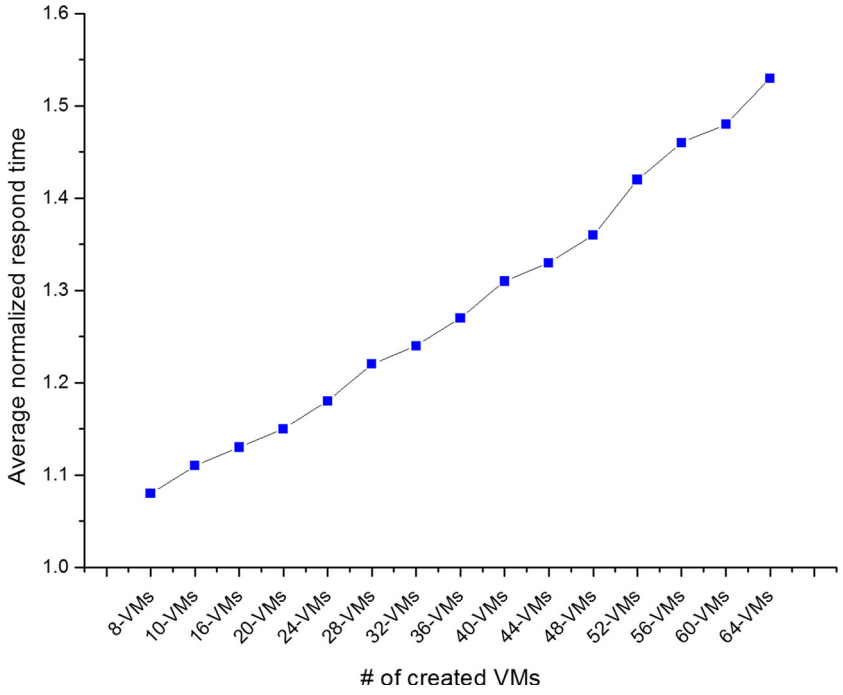

Fig. 11 The normalized performance improvement in different configurations with our DCRM

x-axis presents different configurations, mainly containing the parallel running VMs number.

Figure 11 shows that the more concurrently running VMs on the server the better system performance our DCRM will have. This is due to the more VMs concurrently running on the server causing more interference and more unbalance of the hardware utilization, which results in better chances to bring our DCRM into play.

7 Conclusion

To improve response time, while maximizing the effectiveness of the HMP system to satisfy QoS in the media mobile cloud computing, in this paper, we have proposed the Dynamic Cloud Resource management (DCRM), which combines resource-aware VM allocation with dynamically and optimally partitioned shared resources. The key idea is to profile the resource characteristics of both cores and VMs, estimate their needs for shared resources, mainly CPU, shared caches, memory banks, and memory bandwidth, and then direct both our resource-aware VM placement and resources partitioning based on the estimation. Experimental results have demonstrated that this combination is able to satisfy QoS, and simultaneously enhances power efficiency and throughput for HMP based media mobile computing.

Acknowledgments This work was supported by the National Science Foundation of China under Grant, No. 61602137, 61572172, 61401147 and 61401107, by by the Fundamental Research Funds for the Central Universities, No.2016B10714 and supported by Changzhou Sciences and Technology Program, No. CE20165023 and No. CE20160014 and Six talent peaks project in Jiangsu Province, No. XYDXXJS-00.

References

1. Held J, Bautista J, Koehl S (2006) From a few cores to many: a tera-scale computing research review. Intel Research White Paper, http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf
2. Rodrigues R, Annamalai A, Koren I, Kundu S (2012) Scalable thread scheduling in asymmetric multicores for power efficiency. In: 2012 IEEE 24th international symposium on computer architecture and high performance computing
3. Han G, Que W, Jia G, Shu L (2016) An efficient virtual machine consolidation scheme for multimedia cloud computing. Sensors 16(2):Article 246
4. Hill M, Marty M (2008) Amdahl's Law in the multicore era. Computer 41(7):33–38
5. Winter JA, Albonesi DH, Shoemaker CA (2010) Scalable thread scheduling and global power management for heterogeneous many-core architectures. In: Proceedings of the 19th international conference on parallel architectures and compilation techniques, ser. PACT'10
6. Annavaram M, Grochowski E, Shen J (2005) Mitigating Amdahl's Law through EPI throttling. In: Proceedings of ISCA'05, pp 298–309
7. Kumar R, Farkas KI, Jouppi N et al (2003) Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In: Proceedings of MICRO, vol 36
8. Goldberg RP (1974) Survey of virtual machine research. Computer 7(9):34–45
9. Rosenblum M, Garfinkel T (2005) Virtual machine monitors: current technology and future trends. Computer 38(5):39–47

10. Chen L, Wei Z, Cui Z, Chen M, Pan H, Bao Y (2014) CMD: classification-based memory deduplication through page access characteristics. In: VEE'14
11. Mutlu O, Moscibroda T (2008) Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems. In: ISCA
12. Jeong M, Yoon D, Sunwoo D, Sullivan M, Lee I, Erez M (2012) Balancing DRAM locality and parallelism in shared memory CMP systems. In: HPCA
13. Deng Q, Meisner D, Ramos L, Wenisch TF, Bianchini R (2011) MemScale: active low-power modes for main memory. In: ASPLOS
14. Abdullahi M, Ngadi MA (2016) Symbiotic organism search optimization based task scheduling in cloud computing environment. Futur Gener Comput Syst 56:640–650
15. Abdulhamid SM, Latiff MSA, Abdul-saalam G, Madni SHH (2016) Secure scientific applications scheduling technique for cloud computing environment using global league championship algorithm. Plos One. https://doi.org/10.1371/journal.pone.0158102
16. Abdulhamid SM, Abd Latiff MS, Ismaila I (2014) Tasks scheduling technique using league championship algorithm for makespan minimization in IAAS cloud. ARPN J Eng Appl Sci 9(12):2528–2533
17. Abdulhamid SM, Abd Latiff SM, Bashir MB (2014) Scheduling techniques in on-demand grid as a service cloud: a review. J Theoret Appl Inf Technol 63(1):10–19
18. Madni SHH, Latiff MSA, Coulibaly Y, Abdulhamid SM (2016) Resource scheduling for infrastructure as a service (Iaas) in cloud computing: challenges and opportunities. J Netw Comput Appl 68:173–200
19. Han G, Liu L, Jiang J, Shu L, Hancke G (2017) Analysis of energy-efficient connected target coverage algorithms for industrial wireless sensor networks. IEEE Trans Ind Inf 13(1):342–350
20. Lazri K, Laniepce S, Ben-Othman J (2013) When dynamic VM migration falls under the control of vm users. In: Proceedings of the IEEE 5th international conference on cloud computing technology and science. CloudCom, pp 395–402
21. Han G, Wan L, Shu L, Feng N (2015) Two novel DoA estimation approaches for real time assistant calibration system in future vehicle industrial. IEEE Syst J. https://doi.org/10.1109/JSYST.2015.2434822
22. Ficco M, Esposito C, Palmien F, Castiglione A (2016) A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation. Futur Gener Comput Syst. https://doi.org/10.1016/j.future.2016.05.025
23. Arzuaga E, Kaeli DR (2010) Quantifying load imbalance on virtualized enterprise servers. In: Proceedings of the 1st international conference on performance engineering, (WOSP/SIPEW), pp 235–242
24. Jia G, Han G, Jiang J, Sun N, Wang K Dynamic resource partitioning for heterogeneous multi-core-based cloud computing in smart cities. IEEE Access. https://doi.org/10.1109/ACCESS.2015.2507576
25. Han G, Liu L, Chan S, Yu R, Yang Y (2016) HySense: a hybrid mobile crowdsensing framework for sensing opportunities compensation under dynamic coverage constraint. IEEE Commun Mag. https://doi.org/10.1109/MCOM.2017.1600658CM
26. Jia G, Han G, Jiang J, Liu L (2016) Dynamic adaptive replacement policy in shared last-level cache of DRAM/PCM hybrid memory for big data storage. IEEE Trans Ind Inf. https://doi.org/10.1109/TII.2016.2645941
27. Liu L, Cui Z, Xing M, Bao Y, Chen M, Wu C (2012) A software memory partition approach for eliminating bank-level interference in multicore systems. In: PATC'12
28. Lin J, Lu Q, ing XD, Zhang Z, Zhang X, Sadayappan P (2008) Gaining insights into multicore cache partitioning: bridging the gap between simulation and real systems. In: HPCA-14
29. Xie M, Tong D, Feng Y, Huang K, Cheng X (2013) Page policy control with memory partitioning for DRAM performance and power efficiency. In: ISLPED
30. Muralidhara S, Subramanian L, Mutlu O, Kandemir M, Moscibroda T (2011) Reducing memory interference in multicore systems via application-aware memory channel partitioning. In: MICRO
31. Bellard F (2005) Qemu, a fast and portable dynamic translator. In: Proceedings of the annual conference on USENIX annual technical conference, ATEC'05, pp 41–46
32. Kvm-kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page