Performance-Monitoring-Based Traffic-Aware Virtual Machine Deployment on NUMA Systems

Yuxia Cheng, Wenzhi Chen, Zonghui Wang, and Xinjie Yu

***Abstract*—Virtualization technology enables multiple virtual machines (VMs) to share a single physical server. Commercial servers increasingly use the nonuniform memory access (NUMA) architecture due to its scalable memory performance. However, multiple VMs running on a NUMA physical server will cause performance overheads such as remote memory access latency and shared microarchitectural resource contention, which makes the VM performance less efficient and stable. These performance overheads are mainly caused by memory traffic from data-intensive workloads. In this paper, we propose a traffic-aware VM optimization (TAVO) scheme on NUMA systems. Based on the performance monitoring of the data traffic and CPU/memory resource usages in the system, TAVO addresses VM memory access locality and shared resource contention problems via automatic VM initial placement and NUMA-aware VM online scheduling. Our experimental results show that TAVO improves VM performance in terms of benchmark runtime by up to 22.6% compared with the default KVM CFS scheduler. TAVO also achieves a much stable performance with benchmark's average runtime variation under 3%.**

***Index Terms*—Memory traffic, nonuniform memory access (NUMA), performance monitoring, scheduling, virtual machine (VM).**

I. Introduction

VIRTUALIZATION technology enables multiple virtual machines (VMs) to share a single physical server. VM consolidation improves physical resource utilization in cloud data centers. Cloud platforms commonly use commercial servers to host as many VMs as possible to reduce the total cost of ownership. However, cloud providers have to meet the service level agreement with customers. Achieving high VM consolidation faces big challenges [11] due to constraints on performance degradation. Therefore, it becomes increasingly important to exploit performance opportunities and improve the efficiency of cloud data centers.

Performance inefficiency largely stems from a lack of understanding of the VMs' behavior and their interactions with the underlying machine architectures [7]. Traditional VM monitors (VMMs) regard a physical machine as the resource aggregation of cores, main memory, storage space, etc., but without an explicit view of shared microarchitectural resources such as on-chip caches, memory controllers, and interconnects. These microarchitectural resources have a large impact on the overall system performance.

To improve the efficiency of VMs running on modern servers, we have to understand the physical architecture of these servers. Today's physical servers deployed in data centers typically use the nonuniform memory access (NUMA) architecture due to its high aggregated memory bandwidth and system scalability. However, the NUMA multicore multiprocessor architecture has performance overheads [14] such as remote memory access latency and shared microarchitectural resource contention, which brings significant challenges for optimizing VM performance. Traditional VM schedulers scheduling virtual CPUs (VCPUs) onto physical cores depend on CPU load balance, and there is little consideration of NUMA performance overheads [8]. The NUMA unawareness may lead to suboptimal and unstable VM performance [23], [27]. Existing NUMA-aware optimization methods mainly focus on maximizing memory access locality [3], [4] and seldom consider other microarchitectural resource contention problems that are found equally important to the overall system performance.

In this paper, we use hardware performance monitoring counters to measure microarchitectural resource usages. Based on the performance monitoring of the data traffic and other physical resource usages, we propose a traffic-aware VM optimization (TAVO) scheme on NUMA systems. TAVO addresses both memory access locality and shared resource contention problems via automatic VM initial placement and NUMA-aware VM online scheduling. The main contributions of this paper are described as follows.

1) We implement a low overhead performance monitor using hardware performance counters. The performance monitor collects system-level and VM-level performance events to reflect resource usages. In particular, the performance monitor collects data traffic statistics among NUMA nodes to effectively reveal microarchitectural resource contention.
2) Based on the performance monitoring of the NUMA system, we propose a traffic-aware hybrid (TAH) bin packing algorithm to initially place new VMs into NUMA nodes. The TAH algorithm dynamically collects online CPU/memory/bandwidth usages and uses the multidimensional resource vector that takes the CPU load balance, memory access locality, and microarchitectural resource contention into account.
3) Based on the automatic VM initial placement, we further propose a NUMA-aware VM scheduler to dynamically adjust virtual to physical resource mappings online. The scheduler periodically executes NUMA-aware VCPU scheduling and memory page migration in case of workload phase change.

Manuscript received November 10, 2014; revised April 25, 2015 and June 29, 2015; accepted August 12, 2015. This work was supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2013ZX03003010-002.

The authors are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: rainytech@zju.edu.cn; chenwz@zju.edu.cn; zhwang@zju.edu.cn; yuxinjie@zju.edu.cn).

Digital Object Identifier 10.1109/JSYST.2015.2469652

The proposed TAVO scheme is an integral solution to optimize VM performance on NUMA machines. Our experimental results demonstrate that the TAVO scheme achieves better and more stable VM performance on NUMA systems than the traditional VM scheduling method. The TAVO scheme improves VM performance by up to 22.6% compared with the default KVM CFS scheduler, and TAVO also achieves a much stable performance with benchmark average runtime variation under 3%.

The rest of this paper is organized as follows. Section II discusses related work. Section III describes the background and the design motivation. Section IV presents a method of performance monitoring of resource usages. Section V presents the proposed TAVO scheme. Section VI presents the experimental results. Section VII concludes this paper.

II. Related Work

There has been significant research interest on the multicore and NUMA-related optimizations to nonvirtualized and virtualized systems.

To address shared resource contention on multicore systems, researchers have proposed hardware partitioning and page coloring [18] techniques to mitigate the shared cache contention problem on multicore systems. Efficient coscheduling of different threads to more constructively use shared on-chip resources is another promising technique to alleviate contention problems. Zhuravlev *et al.* [8] proposed DI and DIO thread scheduling algorithms to reduce cache contention by spreading cache-intensive threads apart and coscheduling them with less cache-intensive threads. References [23] and [27] analyzed the optimal thread coscheduling on multicore processors. These researches are mainly focused on the single socket multicore systems.

On NUMA multiprocessor systems, more performance factors have to be considered. Besides shared cache contention, the optimization of NUMA systems should also take care of memory controller congestion, interconnection congestion, and remote memory access latency. Awasthi *et al.* [19] analyzed the problems and opportunities posed by multiple on-chip memory controllers. Majo *et al.* [20] proposed the N-MASS thread mapping algorithm that considers both data locality and cache contention problems on NUMA systems. Blagodurov *et al.* [15] extended their shared contention research [8] to NUMA systems and proposed the DINO NUMA-aware management policy. The DINO policy is a combination of cache-contention-aware thread scheduling and NUMA-aware memory migration. Other compiler-based code transformation techniques [21] are proposed to address resource contention and data locality problems.

Tang *et al.* [14] and [16] studied the performance impact of NUMA systems on large-scale data center applications. Mars *et al*. [13] proposed a mechanism named bubble-up to predict the performance degradation due to colocating multiple data center scale applications on a single multicore server. Based on the bubble-up mechanism, they [12] further proposed the bubble-flux mechanism to efficiently colocate latency sensitive applications and batch jobs. These techniques are proposed in the nonvirtualized environments.

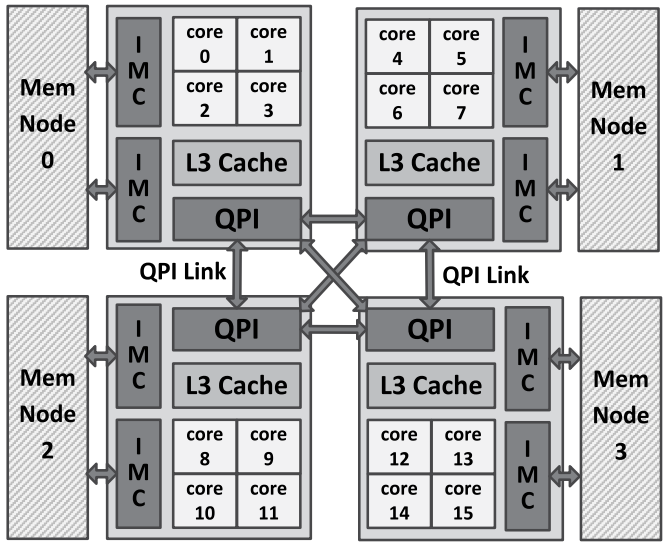


Fig. 1. Simplified architecture of a quad-socket NUMA system.

In virtualized environments, Rao *et al.* [7] proposed a NUMA-aware VCPU migration algorithm. The algorithm uses the uncore penalty metric to predict VCPU performance on NUMA systems and dynamically migrates VCPUs to minimize the system-wide uncore penalty. More recently, Liu *et al.* [11] proposed a NUMA-overhead-aware hypervisor memory management policy. They introduced a method to estimate the memory zone access overhead using hardware performance counters. Based on the estimation, they proposed two optimization techniques: a NUMA-overhead-aware buddy allocator and a P2M swap FIFO. In contrast to the previous studies, our proposed TAVO scheme collects the data traffic performance statistics to help determine VM initial placement and online scheduling on NUMA systems, which both considers VCPU scheduling and memory management.

In cloud data centers, many solutions [31]–[35] were proposed to increase server utilization and reduce resource conflicts. Nathuji *et al.* [31] designed a VM QoS-aware control framework named Q-Clouds. The Q-Clouds framework reserves suitable resources and tunes resource allocations to mitigate performance interference effects. Delimitrou *et al.* [32], [33] proposed a heterogeneity and interference-aware cluster management system that uses collaborative filtering techniques to classify and deploy large-scale different workloads in data centers. Vasic *et al.* [35] proposed the DeepDive system to identify and manage performance interference among VMs colocated on the same physical machine in cloud environments. Their approach to identify interference is based on VM performance classification and exhaustive interference analysis. Once the interference is identified, the VM is migrated to a less loaded machine. In our proposed TAVO scheme, we focused on more fine-grained NUMA node level performance problems within each machine.

III. Background and Motivation

A. NUMA Architecture

The NUMA system has multiple memory nodes and multicore processors. Fig. 1 shows an example of a quad-socket

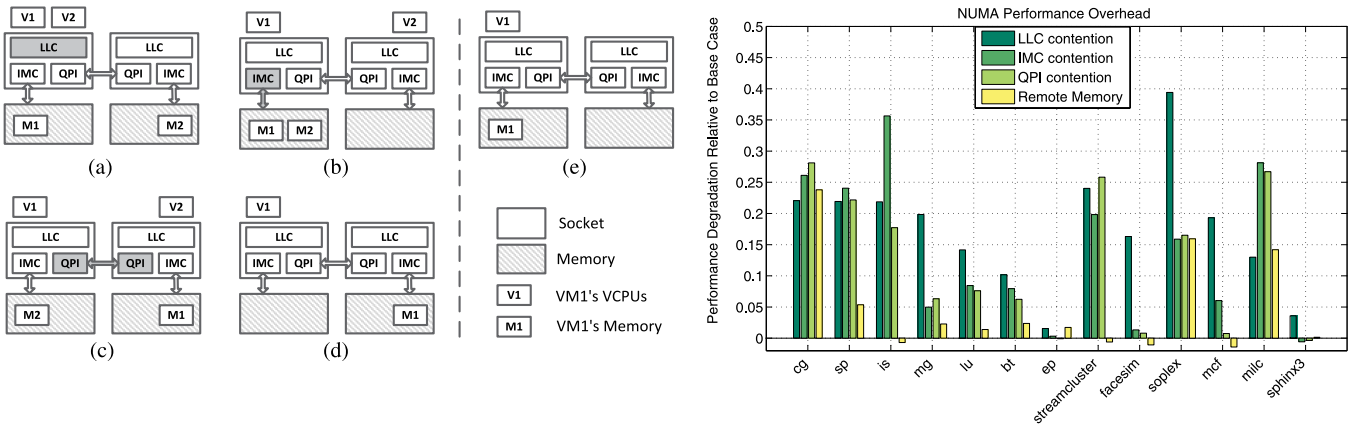


Fig. 2. Comparison of VM performance degradation caused by different performance overheads on the NUMA virtualized systems. (a) LLC contention. (b) IMC contention. (c) QPI contention. (d) Remote memory. (e) Base line local execution.

NUMA system. In each socket, there are four cores sharing the last level cache (LLC or L3 cache) and the integrated memory controller (IMC). Sockets communicate with each other via high-speed interconnect (e.g., Intel QPI links shown in the figure). The IMC in each socket is connected to its local memory node. A NUMA node is composed of a socket and a directly connected memory node. The socket can access remote memory via the remote IMC, and the QPI link is responsible for data transmission between sockets. Due to intersocket communication overhead, remote memory access is slower than local memory access. Although other NUMA multiprocessors (e.g., AMD Opteron) may differ in the number of sockets and cores, the size of shared caches, and the techniques of interconnect (e.g., AMD HyperTransport), they have very similar architectural designs. Therefore, most of our discussions are applicable to them. As more cores will be integrated into one socket and more sockets will be interconnected within one system, the shared resource contention [9] and data traffic problem [10] becomes even more severe in the future hardware.

B. Virtualized System

The system virtualization layer, typically the VMM or the hypervisor, provides a virtual abstraction of the machine hardware for each guest OS. Multiple VMs share hardware resources. In order to access the actual hardware, each guest OS running on the VM needs the virtual to machine translation. Therefore, to better utilize the NUMA physical machines, the mapping from virtual to machine resources should consider the NUMA system's characteristics. However, traditional VMMs provide a Uniform memory access virtual abstraction for the guest OS [25]. Without the knowledge of the underlying NUMA architecture, VMs cannot be aware of the performance overheads running on the NUMA systems. The NUMA unawareness in virtualized systems results in suboptimal and unstable VM performance.

To study the performance impact of virtual to machine resource mappings on NUMA systems, we design four different VM mapping scenarios that represent four sources of performance degradation factors. Fig. 2(a) represents the LLC contention that two VMs' VCPUs are bound to the same socket and their memory is allocated on separate NUMA nodes. Fig. 2(b) represents the IMC contention that two VMs' memory is allocated on the same node and each VM's VCPUs are bound to separate sockets. Fig. 2(c) represents the QPI interconnect contention that two VMs' VCPUs access their memory from remote nodes and contend for the QPI link. Fig. 2(d) represents the remote memory access overhead that one VM accesses its memory from the remote NUMA node. Fig. 2(e) represents the base case that the VM's VCPUs and memory are mapped on the same node. In each scenario, VMs are allocated with the same physical resources of CPU and memory. We compare the VM1's performance of scenarios Fig. 2(a)–(d) with the base case scenario Fig. 2(e). The experimental platform and benchmarks are described in detail in Section VI. We run a set of benchmarks in VMs under different scenarios and record their performance results. We plot the benchmark performance degradation under scenarios Fig. 2(a)–(d) relative to the base case performance under scenario Fig. 2(e).

From the experimental result, we make three observations. First, some applications have higher performance degradation than others under the same VM mapping scenario. We find that data-intensive applications are more sensitive to the NUMA performance overheads. Second, all four sources of NUMA overheads are equally important in terms of impacting on the application's performance. Third, each source of NUMA overhead alone can result in a significant performance degradation. Therefore, in order to optimize the performance of NUMA virtualized systems, it is important to identify the NUMA system performance bottlenecks in terms of microarchitectural resource usages and to balance each shared resource usage across the NUMA system.

C. Problems and Challenges

VMs running on multicore NUMA systems have many shared resources, and different VM corunning combinations can lead to different levels of performance degradation. Previous researchers have found that finding an optimal task to core assignment on the multicore multiprocessor systems to maximize the overall system throughput is an NP-complete problem [23], [27]. As the number of cores, the number of shared resources, the number of NUMA nodes, and the number of simultaneously running tasks in the system increase, the number of possible task assignment grows exponentially. Therefore, it is impractical to enumerate all possible VM

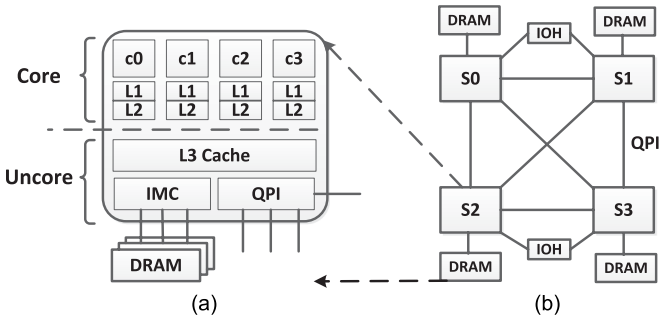


Fig. 3. Performance monitoring of microarchitectural resource usages. (a) Simplified block diagram of the Nehalem processor. (b) Simplified block diagram of the quad-socket NUMA system.

assignments to achieve the optimal system performance. What is more, tasks running in VMs may have changing behaviors, and once the VM workload changes, the system should adjust the VM assignment accordingly.

In practice, one simple method that has been used by current VMMs is to bind a VM to one NUMA node if the NUMA node has enough free CPU and memory resources. That is to allocate the VM's memory in one NUMA node and schedule the VM's VCPUs on the same node. This bind policy guarantees the VMs all memory accesses to be local and achieves a much stable performance. However, the bind policy only addresses the remote memory access problem on NUMA systems. The other performance degradation sources, such as shared cache contention, memory controller contention, etc., remain unsolved.

To more effectively exploit multicore NUMA system performance, we should both consider memory access locality and shared resource contention problems. One practical solution is to capture the online interactions between VMs and underlying physical resources and heuristically balance the system load. Generally, a more balanced use of system resources can achieve better performance [24]. The VMMs that can understand microarchitectural resource utilization on NUMA systems and dynamically adjust virtual to physical resource mappings are expected to have better and more stable VM performance.

IV. Performance Monitoring of Microarchitectural Resource Usage

In this section, we describe the technique of characterizing microarchitectural resource usages using a hardware performance monitoring unit (PMU) in modern multicore processors. It can help us identify the performance bottlenecks in the NUMA system and provide the basis for NUMA-aware performance optimization.

The PMU in modern multicore processors [6] provides the capabilities of monitoring a wide variety of performance events to illuminate the code interactions with the architecture. There are two types of performance events: core events and "uncore" events. As Fig. 3(a) shows, the core events include the events of the processing cores (c0–c4), the private L1 and L2 caches; the "uncore" events include the events related to the shared L3 cache (LLC), the IMC, and the socket interconnection interface (QPI).

In the symmetric multicore processor, each core has the same architecture of the out-of-order pipeline and the private L1 and L2 caches. Data requests that miss in the private L2 cache are sent to the "uncore" memory subsystem. The "uncore" memory subsystem in a multicore socket mainly consists of a shared L3 cache, an IMC, and several QPI links. In a quad-socket NUMA system, as Fig. 3(b) shows, each socket has four QPI links, with one linked to the IOH (I/O hub) and the other three linked to the remote sockets. Therefore, excessive data requests sent by the cores will result in resource contention on the "uncore" memory subsystem.

Data requests that are missed in the shared L3 cache will be sent either to the IMC attached to the local node or to the QPI links connected to the remote node, which depends on the requested data residing either in the local memory node or in the remote memory node. Through monitoring the data traffic that passes through the L3 cache, the IMC, and the QPI links, we can get a holistic view of resource usages on the "uncore" memory subsystem. Traffic congestions on the memory subsystem can severely hurt application performance. Therefore, data traffic management is particularly important for the total throughput of NUMA systems.

To get a holistic view of the data traffic statistics, we calculate three kinds of performance metrics: the L3 cache miss rate, the memory bandwidth usage within each NUMA node, and the interconnect bandwidth usage between every two NUMA nodes. The L3 cache miss rate is calculated by using the number of the L3 cache misses divided by the number of cache references. The L3 cache events are periodically collected. The higher the L3 cache miss rate, the more intensive the L3 cache contention.

The recent Intel processors (microarchitecture code name Nehalem or newer) [6] provide the capability of counting the number of memory controller read requests and the number of full cache line writes to DRAM (the PMU "uncore" events). This allows us to calculate the memory bandwidth usage within each NUMA node by counting the total number of memory controller reads and writes during a certain period of time. In the same way, the interconnect bandwidth usage between two NUMA nodes can be estimated by counting the number of data transmissions coming to and outgoing from the socket through the QPI links.

Therefore, through monitoring data traffic that passes through the "uncore" memory subsystem, we can get the knowledge of the microarchitectural resource usages and identify the imbalanced use of these resources. This kind of performance monitoring capabilities can help us more effectively schedule VMs on NUMA systems.

V. NUMA-Aware VM Placement and Scheduling

Based on the analysis of the NUMA performance overhead and the capability of monitoring performance events described in the previous sections, we propose a TAVO scheme on NUMA systems. The TAVO scheme strives to balance the NUMA resource usages in order to achieve better and more stable VM performance.

A. Overview

The key idea of the TAVO scheme is to optimize performance on consolidated virtualized NUMA systems by balancing physical resource usages among different NUMA nodes

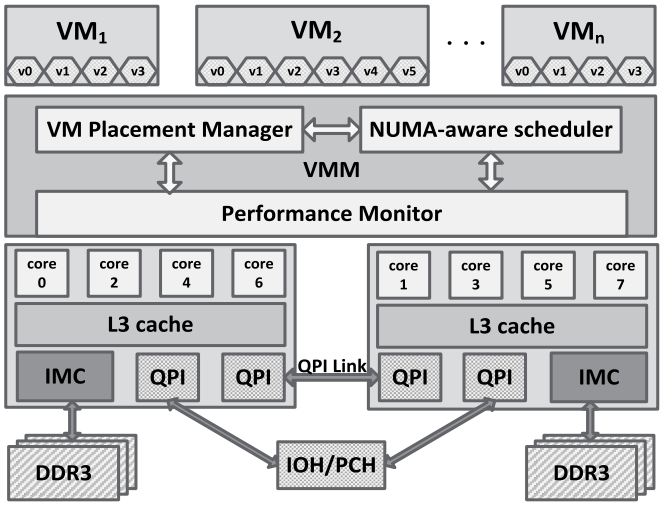


Fig. 4. Architecture overview of the proposed TAVO scheme in the virtualized NUMA system.

and determining efficient VM colocations. The TAVO scheme involves VM initial placement and NUMA-aware VM scheduling. In order to handle multiple NUMA performance overheads, TAVO introduces a traffic-aware multicapacity bin packing algorithm to efficiently determine the VM initial placement. Taking advantage of the VM initial placement, TAVO further adjusts virtual to physical resource mappings online through a flexible NUMA-aware scheduler during the whole VM lifetime. As Fig. 4 shows, TAVO is composed of three major parts: the performance monitor, the VM placement manager, and the NUMA-aware VM scheduler. The performance monitor is responsible for collecting performance events of each VM as well as the whole NUMA system. When new VMs need to be deployed on the NUMA machine, the VM placement manager is activated and uses the performance events collected from the performance monitor as input to execute the VM initial placement algorithm. After the VMs are deployed on the system, the NUMA-aware VM scheduler periodically detects workload phase change and schedules proper VMs onto less loaded NUMA nodes.

B. Performance Monitor

The performance monitor periodically collects performance events online. There are two kinds of performance events collected by the monitor: software events (such as CPU and memory usage statistics that are exported by host OS) and hardware events (collected by reading hardware performance counters). The performance monitor gathers these events and classifies them into system-level and VM-level performance events.

System-Level Performance Events: To get a holistic view of the NUMA system physical resource usage, the monitor periodically updates the following system-level performance statistics.

1) *CPU and memory usage*. The system's CPU usage and memory usage are two basic indicators for measuring overall system load. The monitor collects the CPU and memory usage statistics on a per-node basis.
2) *Memory bandwidth and interconnect bandwidth usage*. Imbalanced use of memory bandwidth and interconnect bandwidth may cause data traffic congestion on the NUMA system [10]. The monitor collects memory bandwidth usage on a per-node basis and collects interconnect bandwidth usage between NUMA nodes (described in Section III).

VM-Level Performance Events: To understand the performance characteristics of different VMs, the monitor periodically collects the following performance statistics on a per-VM basis.

1) *Per-VM CPU and memory usage*. The monitor collects each VM's CPU and memory usage information provided by the host OS.
2) *Per-VM LLC miss rate*. The monitor also collects the LLC miss rate on a per-VM basis. Currently, the PMU does not provide the capability of monitoring per-VM memory bandwidth usage [6]. Therefore, we use the LLC miss rate to infer the VM's memory access intensity.

C. VM Placement Manager

When a new VM needs to be deployed on the NUMA machine, the VM placement manager decides on which NUMA node(s) the new VM should initially be placed (the new VM's memory and CPU should be allocated). The VM initial placement on NUMA systems can be regarded as an incarnation of the bin packing problem [5], which is NP-hard. Therefore, using heuristics is a reasonable way to address the problem. Traditional NUMA optimization technique uses the first fit (FF; or best/worst fit) algorithm to find the proper node(s) that have enough physical CPUs (PCPUs) and enough free memory to accommodate the new VM. Besides considering CPU and memory resources, we add NUMA traffic awareness into the VM placement manager.

The NUMA traffic reflects microarchitectural overheads that are critical to system performance. Therefore, we propose a new TAH bin packing algorithm to more intelligently place VMs into NUMA nodes. The TAH algorithm is a variation of the multidimensional bin packing algorithm. We regard the physical NUMA nodes as bins and regard VMs as items. Each bin has 3-D resource capacities, which represent the CPU, memory, and bandwidth resources. The capacity of a bin is represented by a vector $\vec{C}= (C_{\rm cpu}, C_{\rm mem}, C_{\rm bw})$, where $C_{\rm cpu}$, $C_{\rm mem}$, and $C_{\rm bw}$ represent the bin's available capacities of the CPU, memory, and bandwidth resources, respectively. Each item (VM) is represented by a corresponding 3-D resource requirements vector $\vec{R}= (R_{\rm cpu}, R_{\rm mem}, R_{\rm bw})$, where $R_{\rm cpu}$, $R_{\rm mem}$, and $R_{\rm bw}$ represent the item's resource requirements of the CPU, memory, and bandwidth, respectively. We normalize each resource's capacity and requirement such that their value lies between 0 and 1.

To exploit more performance opportunities, we introduce hybrid bins. A hybrid bin aggregates more than one NUMA node resource capacities. Hybrid bins are dynamically determined by the interconnect bandwidth usage. The NUMA nodes linked by interconnects can be formed as a hybrid bin if their interconnect bandwidth usages are lower than a predefined threshold. If a

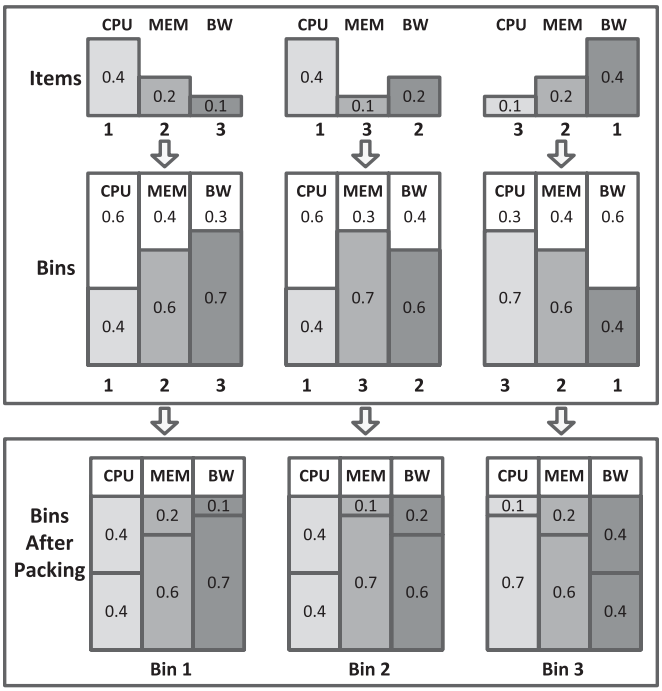


Fig. 5. Examples for balancing 3-D resource capacities in the TAH bin packing algorithm.

VM is placed into a hybrid bin, its memory is interleaved among those NUMA nodes that form the hybrid bin. Thus, the VM can benefit from the aggregated CPU, memory, and bandwidth resources of multiple NUMA nodes.

When a new VM needs to be deployed on the NUMA system, the VM placement manager first collects performance statistics from the performance monitor and creates multiple bins. The created bins include single-node bins and multinode hybrid bins. Each bin is initialized with 3-D resource capacities to reflect the amount of corresponding resources which are currently available on the NUMA system. Then, the VM placement manager looks up a proper bin for the new VM. The look-up procedure is as follows.

1) Sort the bins by the sum of their three resource capacities in descending order. The sorted bins fall into two categories: single-node bins and multinode hybrid bins, with the single-node bins listed ahead of the multinode bins. Therefore, VMs can be first put into single-node bins to prioritize memory access locality.
2) Label the new item (VM) and each bin with the relative order of three resource requirements and capacities. For example, as Fig. 5 shows, the new item $\vec{R}$ has resource requirement that $\vec{R}= (0.4, 0.2, 0.1)$, and we label the relative order of three resource requirements with (1, 2, 3). Similarly, before the new items are packed into the bins, the Bin1's resource capacity $\vec{C}= (0.6, 0.4, 0.3)$, so we label the relative order of three resource capacities with (1, 2, 3).
3) Search the sorted bin list from the beginning until we find a bin that satisfies the following two conditions: a) the resource capacities of the bin meets the resource requirements of the new item. b) The bin has the same relative order of resource capacities with the item's relative order of resource requirements. By satisfying condition b), we try to balance 3-D resource usages on each bin. As Fig. 5 shows, when we pack the items into the bins that have the same relative order of resource requirements and capacities, each bin after packing has more balanced use of 3-D resources. Balanced use of each resource can minimize system performance bottlenecks, and therefore, more VMs can be consolidated into the system.
4) If the search failed in 3), we relax search condition b) so that the bin should satisfy the same maximum resource capacity with the item's maximum resource requirement. If this round search failed again, we further relax the search condition to only satisfy condition a).

Finally, we assign the satisfied bin's corresponding NUMA node(s) as the VM's "home" node(s) in which the VM's VCPUs and memory resources are initially allocated. If the relaxed search still cannot find the satisfied bin, the VM placement manager reports that this new VM cannot be deployed on this NUMA machine due to resource constraint.

D. NUMA-Aware VM Scheduler

The NUMA-aware VM scheduler provides a flexible solution to dynamically adjust virtual to physical resource mappings online. The scheduler has two major functionalities: VCPU scheduling and memory page migration.

VCPU scheduling adaptively maps VM's VCPUs onto PCPUs according to the system load. Each VM is assigned the "home" NUMA node(s) in the VM initial placement phase. To more flexibly use physical resources, we do not statically pin the VM's VCPUs onto PCPUs of its "home" node(s). Instead, the NUMA-aware scheduler prefers to schedule VCPUs to their "home" node(s) but provides the opportunity of letting VCPUs run on other nodes.

Initially, the NUMA-aware scheduler sets the CPU affinity of each VM to its "home" node(s). VCPU scheduling within each NUMA node is taken over by the default CFS scheduler, which preserves the original CPU load balancing within each node. The NUMA-aware scheduler is responsible for load balancing across different nodes. System-level and VM-level performance statistics are periodically collected from the performance monitor. Then, the NUMA-aware scheduler determines whether there is a need to schedule VCPUs from a heavy loaded node to a light loaded node. The scheduling process is as follows.

1) The scheduler periodically checks whether the system has a load imbalance between NUMA nodes. If there are active VCPUs waiting in the CPU run queue on one NUMA node while there are idle PCPUs on other NUMA nodes, then the scheduler determines if there exists a load imbalance between NUMA nodes. If the load imbalance is not detected, the NUMA-aware scheduler sleeps for a period of time (we set 1 s in our implementation).
2) After the load imbalance is detected, the scheduler finds the VCPU scheduling source node and destination node. The NUMA node that has the most number of active VCPUs waiting in the run queue is selected as the scheduling source node. The NUMA node that has the most number of idle physical cores is selected as

the scheduling destination node. If there exist multiple source nodes and destination nodes, the scheduler selects the source and destination node pair that has the lowest interconnect bandwidth usage between these two nodes to reduce the interconnect contention.

3) The scheduler then determines how many VCPUs should be scheduled from the source node to the destination node. The number of VCPUs to be scheduled is equal to the minimum value of the number of idle physical cores in the destination node and the number of waiting VCPUs in the source node. It is better to let the VCPUs run on idle cores in other nodes rather than waiting in their "home" node, even if these VCPUs running on other nodes may have remote memory access latency.
4) The scheduler further decides which VCPUs should be scheduled out from the source node. To minimize remote memory access latency and interconnect bandwidth contention, the scheduler selects the VCPUs whose "home" node is equal to the destination node or the VCPUs that have the least LLC miss rate in the source node. Finally, the scheduler updates the CPU affinity of the selected VCPUs to the destination node. The migrated VCPUs will be scheduled back to its "home" node, once its "home" node has idle physical cores or its "home" node becomes the least loaded node in the system.

Memory page migration provides the capability of moving memory pages between two NUMA nodes online. Note that memory page migration causes performance overheads; we prohibit frequent page moves during VM's lifetime. The memory page migration is triggered when a VM's "home" node needs to be updated because a long-term load imbalance between NUMA nodes is detected in the system. During the VCPU scheduling phase, the NUMA-aware scheduler records the number of times each VM's VCPUs are scheduled out from their "home" node(s) in the last ten scheduling epochs. If one VM's VCPUs have more than eight times out of ten being scheduled out from their "home" node(s), then the scheduler determines that the VM's "home" node(s) should be updated because, most of the time, the VM's VCPUs are running on other nodes. Updating VM's "home" node(s) involves recalculating the system resource requirements and capacities. Moreover, the scheduler invokes the VM initial placement algorithm to recalculate the VM's new "home" node(s). After updating the VM's new "home" node(s), the NUMA-aware scheduler migrates the VM's corresponding memory pages to its new "home" node(s).

E. Implementation

We implement a prototype of the proposed TAVO scheme in the KVM virtualized platform. The performance monitor, the VM initial placement manager, and the NUMA-aware VM scheduler are implemented as individual daemons in the KVM host operating system. The performance monitor accesses the hardware performance counters via the perf_event module provided by the Linux kernel, and the monitor also collects other software performance statistics via parsing the pseudo-*proc* file system. The VM initial placement manager obtains the NUMA topology via parsing the pseudo-*sysfs* file system and automatically sets the VM's configure file according to the placement decisions. The NUMA-aware scheduler uses the sched_setaffinity() and move_pages() system calls provided by the Linux kernel to dynamically adjust VCPU to PCPU mappings online and migrates VM's memory pages when the system load changes.

VI. Performance Evaluation

We run the experiment on the quad-socket Dell R910 server. The server is configured with four 1.87-GHz Intel Xeon E7520 processors based on the Nehalem-EX architecture. Each processor has four cores sharing a 18-MB L3 cache. The processors are interconnected via the Intel QPI links. The R910 server has a total of 16 physical cores and 64-GB memory, with each NUMA node having four physical cores and 16-GB memory. With the Intel HyperThreading enabled, there are a total of 32 hardware threads in the system.

VMs run on the qemu-kvm (version 1.0) virtualized platform. Both the host and guest operating systems used in the experiments are Ubuntu 12.04 with the Linux kernel version 3.8.0-35. Each VM is configured with four VCPUs and 8-GB memory. We select the following benchmarks to run in VMs and record their execution times for each run.

1) **NPB**. The NAS parallel benchmark (NPB) suite [1] is a set of benchmarks developed for evaluating the performance of parallel systems. The NPB benchmark suite consists of five parallel kernels and three simulated application benchmarks. We used the OpenMP version with each benchmark compiled with four threads and set the scale to class B.
2) **SPEC CPU 2006**. SPEC CPU 2006 [2] is an industry-standardized CPU- and memory-intensive benchmark suite. The benchmarks stress a system's processor and memory subsystem resources.

A. Improvement on VM Performance

We evaluate the proposed TAVO scheme with the following two different scheduling strategies.

1) **Default**. The QEMU-KVM default strategy uses the kernel's default completely fair scheduler (CFS) and default memory allocation policy. The CFS schedules VCPU threads to different physical cores depending on the CPU load balance and seldom considers NUMA overheads. The default memory allocation policy allocates a VM's memory on the NUMA nodes where the VM's VCPUs are running. In the default strategy, a VM's VCPUs and memory will be scattered around different NUMA nodes.
2) **FF-Pin**. The FF-Pin strategy uses the VCPU pinning and memory binding methods to statically place a VM onto a NUMA node. The FF algorithm is used to find a proper NUMA node for each VM. The FF algorithm searches the NUMA node that first satisfies the CPU and memory resource requirements for each VM. The FF-Pin strategy benefits from local memory access but lacks the flexibility of using other NUMA node resources.

Fig. 6 shows the performance comparison of benchmark workloads under three different scheduling strategies: default,

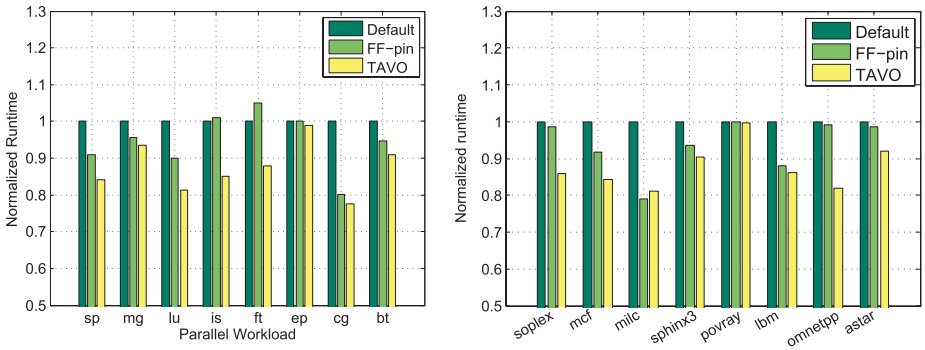


Fig. 6. Performance comparison of VM performance under three different scheduling schemes: default, FF-Pin, and TAVO. The performance results are normalized to the workload runtime of each individual benchmark under the default scheduling strategy.

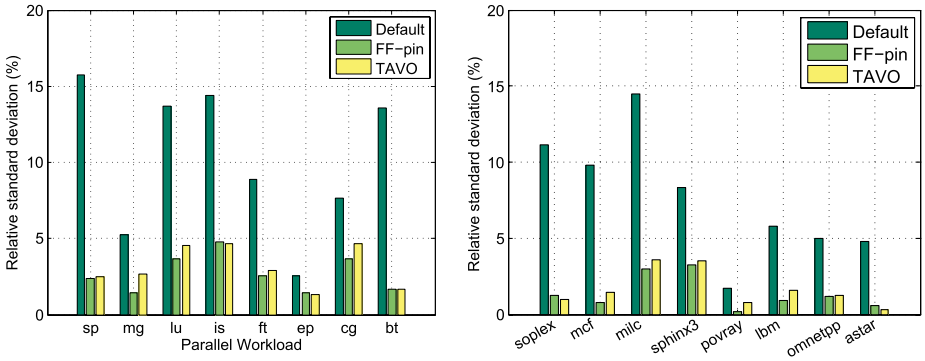


Fig. 7. Performance stability comparison of benchmarks running in VMs under three scheduling strategies: default, FF-Pin, and TAVO. The RSD value is plotted for each benchmark with five individual runs under each strategy.

FF-Pin, and TAVO. The performance of each benchmark is normalized to the benchmark runtime of the default strategy. Each benchmark runtime is the average of five runs under the same strategy. For the NPB benchmark [Fig. 6(a)], the TAVO scheme outperformed both the default QEMU-KVM and the FF-Pin strategies. Compared with the default strategy, the performance improvement of TAVO ranged from 1.2% (ep) to 22.6% (cg). We examined the benchmarks of ep and cg and found that cg consumes a much larger memory bandwidth than ep. Thus, cg is more memory intensive than ep. The TAVO scheme can more effectively improve the performance of the memory-intensive workload. For the SPEC CPU 2006 benchmark [Fig. 6(b)], the TAVO scheme outperformed both the default QEMU-KVM and the FF-Pin strategies. For example, the performance of soplex benchmark under TAVO scheme improved by 14.2% and 13.8% compared with the other two strategies, respectively. As for the povray benchmark, the performance improvement was not obvious. This is because povray is not a memory-intensive workload and is not sensitive to NUMA overheads.

There are two reasons that the TAVO scheme has performance advantages against the default and FF-Pin strategies. First, the VM initial placement in the TAVO scheme assigns each VM the "home" node(s), which guarantees VMs' most memory accesses to be local compared with the default strategy. The FF-Pin strategy also guarantees VMs' local memory accesses, but it only considers CPU and memory resources without taking into account microarchitectural resources. Second, the NUMA-aware scheduler in the TAVO scheme provides flexible solutions to schedule VCPUs and migrate memory pages when the system workload phase changes. Therefore, the TAVO scheme can exploit more performance opportunities than the default and FF-pin strategies.

B. Reduction on VM Performance Variation

In Fig. 7, we compare the performance variations of workloads running in VMs under different scheduling strategies. We use the relative standard deviations (RSDs) of benchmark runtime to represent the degree of performance variation. The RSD value is calculated for each benchmark with five individual runs under each strategy. The smaller the RSD value, the more stable the workload performance.

Fig. 7(a) shows the RSD comparison of NPB benchmarks among three different scheduling strategies. It is obvious that the default strategy has a much higher performance variation than the FF-Pin and TAVO strategies. The default strategy only considers the load balance of physical cores when scheduling VCPU threads and seldom takes into account the NUMA performance overheads. Thus, the default strategy causes considerable performance variations. The FF-Pin strategy has fixed VCPU-to-core mappings, so it has very small RSD values across all benchmark workloads. On average, the NPB benchmarks under the FF-Pin strategy have no more than 4% runtime variations. Except for the is benchmark, which has a very short runtime. The TAVO scheme achieves a similar performance variation with the FF-Pin strategy. The average RSD value of the NPB benchmark under the TAVO scheme does not exceed 5%.

Fig. 7(b) shows the performance variation of the SPEC CPU2006 benchmark under three scheduling strategies. The

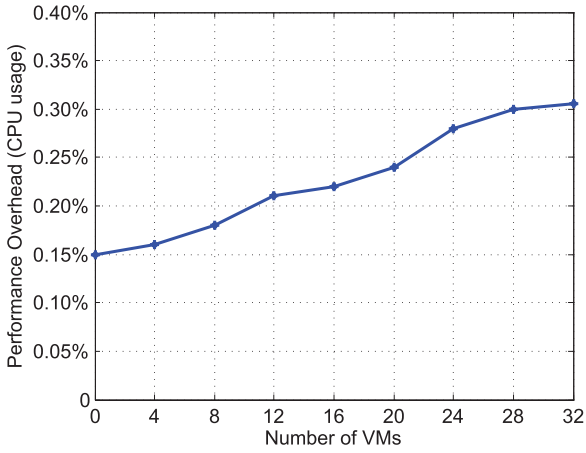


Fig. 8. Average runtime CPU overhead of the TAVO scheme.

results are similar with the NPB benchmark. From the experimental results, we observe that the memory-intensive workloads (soplex, mcf, milc, and sphinx3) have larger RSD values than the relatively less memory-intensive workloads (povray, lbm, omnetpp, and astar). This is because the performances of the memory-intensive workloads are more likely affected by the NUMA overheads when their VCPU-to-core mappings change due to scheduling.

C. Overhead Analysis

Fig. 8 shows the CPU usage of TAVO. The CPU usage includes the performance monitor thread, the VM initial placement manager thread, and the NUMA-aware scheduler thread. The main runtime overhead is due to periodically updating system-wide and per-VM performance statistics. The VM initial placement manager is only activated when new VMs are needed to be deployed on the NUMA system. Therefore, the runtime overhead of the VM initial placement manager is trivial compared with the performance monitor and the NUMA-aware scheduler. Updating the system-wide performance statistics has a constant runtime overhead, while updating the per-VM performance statistics has a runtime overhead correlated with the number of VMs in the system. We set the updating cycle to 1 s, which is a good balance between obtaining accurate online performance statistics and keeping the runtime overhead relatively low. As Fig. 8 shows, the performance overhead increases slightly with the increasing of the number of VMs. When the system has a total of 32 VMs, the CPU usage of TAVO is around 0.3%.

VII. Discussion

The proposed TAVO scheme is implemented as low overhead daemons in the user-level space of the KVM host OS. Unlike the previous NUMA optimization techniques that mostly need to modify critical kernel codes [7], [11], the TAVO scheme can be more easily deployed in the production systems without updating and recompiling kernels. The major weakness of TAVO is its relatively slow response to VM's workload phase change due to its user-level implementation [17], compared with more radical optimization methods implemented in the kernel/hypervisor space. However, as typical services deployed in VMs are long running applications, it is acceptable to make optimization adjustment due to workload change within seconds.

In the experimental section, the TAVO scheme is tested in the KVM virtualized environment. We have tested VM performance overheads in the Xen system and found similar performance degradations; therefore, we infer that the TAVO scheme is also applicable in other VMMs as long as the underlying physical servers have the same NUMA overheads. However, to demonstrate the effectiveness of the TAVO scheme under different VMMs, we need to implement the corresponding performance monitor, VM initial placement manager, and VM scheduler daemons in different systems. We plan to design the daemons with an architecture-specific layer, a VMM-specific layer, and an algorithm layer, so that we can deploy the TAVO scheme on as many servers and VMMs as possible in the future.

VIII. Conclusion and Future Work

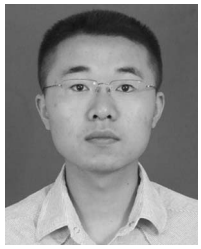
Modern NUMA multiprocessor systems impose significant challenges to the achievement of optimal and stable program performance. Especially in the virtualized environment, the virtualization layer limits the visibility of the NUMA topology to applications running inside VMs. Server consolidation further complicates the problem. Multiple VMs with various memory behaviors consolidated on a single server will contend for shared resources on NUMA multiprocessor systems. To address these problems, we have used the hardware performance monitoring technique to characterize VM memory behaviors and monitor data traffic on the NUMA system. Then, we have proposed the TAVO scheme on NUMA systems. The TAVO scheme consists of three major parts: the performance monitor, the VM initial placement manager, and the NUMA-aware VM scheduler. The three parts work together to minimize NUMA performance overheads. Experimental results showed that our proposed scheme achieves better and more stable VM performance on NUMA systems than the traditional VM scheduling policy.

Future computer architecture will integrate more cores into the system. The microarchitecture and memory subsystem design will become even more complex. Understanding the interactions between software and hardware is of great importance to improve system efficiency. The hardware performance monitoring technique is a promising way to help understand the software behaviors on the hardware. For future many-core and heterogeneous systems, scheduling resources based on hardware performance monitoring can more effectively exploit performance opportunities on these future hardware platforms.

References

[1] NAS Parallel Benchmarks. [Online]. Available: http://www.nas.nasa.gov/publication-s/npb.html
[2] SPEC CPU, 2006. [Online]. Available: http://www.spec.org/cpu2006/
[3] AutoNUMA: The other approach to NUMA scheduling. LWN.net, Mar. 2012. [Online]. Available: http://lwn.net/Articles/488709/
[4] Xen4.3 NUMA Aware Scheduling. [Online]. Available: http://wiki.xensource.com/wiki/Xen_4.3_NUMA_Aware_Scheduling
[5] Bin Packing Problem. [Online]. Available: http://en.wikipedia.org/wiki/Bin_packing_problem
[6] Intel 64 and IA-32 Architectures Software Developer's Manual, Intel, Santa Clara, CA, USA, Volume 3: System Programming Guide, 2011.

[7] J. Rao, K. Wang, X. Zhou, and C. Xu, "Optimizing virtual machine scheduling in NUMA multicore systems," in *Proc. HPCA*, 2013, pp. 306–317.
[8] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. ASPLOS*, 2010, pp. 129–142.
[9] Y. Cheng and W. Chen, "Evaluation of virtual machine performance on NUMA multicore systems," in *Proc. 3PGCIC*, 2013, pp. 136–143.
[10] M. Dashti *et al.*, "Traffic management: A holistic approach to memory placement on NUMA systems," in *Proc. ASPLOS*, 2013, pp. 381–394.
[11] M. Liu and T. Li, "Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads," in *Proc. ISCA*, 2014, pp. 325–336.
[12] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers," in *Proc. ISCA*, 2013, pp. 607–618.
[13] J. Mars, L. Tang, K. Skadron, M. L. Soffa, and R. Hundt, "Increasing utilization in modern warehouse scale computers using bubble-up," *IEEE Micro*, vol. 32, no. 3, pp. 88–99, May/Jun. 2012.
[14] L. Tang *et al.*, "Optimizing Google's warehouse scale computers: The NUMA experience," in *Proc. HPCA*, 2013, pp. 188–197.
[15] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for NUMA-aware contention management on multicore systems," in *Proc. USENIX ATC*, 2011, p. 1.
[16] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *Proc. ISCA*, 2011, pp. 283–294.
[17] Y. Cheng, W. Chen, X. Chen, B. Xu, and S. Zhang, "A user-level NUMA-aware scheduler for optimizing virtual machine performance," in *Proc. APPT*, 2013, pp. 32–46.
[18] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multi-core cache management," in *Proc. EuroSys*, 2009, pp. 89–102.
[19] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers," in *Proc. PACT*, 2010, pp. 319–330.
[20] Z. Majo and T. R. Gross, "Memory management in NUMA multicore systems: Trapped between cache contention and interconnect overhead," in *Proc. ISMM*, 2010, pp. 11–20.
[21] Z. Majo and T. R. Gross, "Matching memory access patterns and data placement for NUMA systems," in *Proc. CGO*, 2012, pp. 230–241.
[22] D. Kaseridis, J. Stuecheli, J. Chen, and L. K. John, "A bandwidth-aware memory-subsystem resource management using non-invasive resource profilers for large CMP systems," in *Proc. IEEE HPCA*, 2010, pp. 1–11.
[23] Y. Jiang, X. Shen, C. Jie, and R. Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors," in *Proc. PACT*, 2008, pp. 220–229.
[24] W. Wang *et al.*, "Performance analysis of thread mappings with a holistic view of the hardware resources," in *Proc. ISPASS*, 2012, pp. 156–167.
[25] D. S. Rao and K. Schwan, "vNUMA-mgr: Managing VM memory on NUMA platforms," in *Proc. HiPC*, 2010, pp. 1–10.
[26] M. Lee and K. Schwan, "Region scheduling: Efficiently using the cache architectures via page-level affinity," in *Proc. ASPLOS*, 2012, pp. 451–462.
[27] P. Radojkovic *et al.*, "Optimal task assignment in multithreaded processors: A statistical approach," in *Proc. ASPLOS*, 2012, pp. 235–248.
[28] J. Rao, K. Wang, X. Zhou, and C. Z. Xu, "Towards fair and efficient SMP virtual machine scheduling," in *Proc. HPCA*, 2014, pp. 273–286.
[29] W. Wang, T. Dey, J. W. Davidson, and M. L. Soffa, "DraMon: Predicting memory bandwidth usage of multi-threaded programs with high accuracy and low overhead," in *Proc. ASPLOS*, 2014, pp. 1–13.
[30] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, "Using OS observations to improve performance in multicore systems," in *Proc. MICRO*, 2008, pp. 54–66.
[31] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-Clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. Eurosys*, 2010, pp. 237–250.
[32] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. ASPLOS*, 2013, pp. 77–88.
[33] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proc. ASPLOS*, 2014, pp. 1–17.
[34] R. C. Chiang, J. Hwang, H. Huang, and T. Wood, "Matrix: Achieving predictable virtual machine performance in the clouds," in *Proc. USENIX ATC*, 2014, pp. 1–12.
[35] D. Novakovic *et al.*, "DeepDive: Transparently identifying and managing performance interference in virtualized environments," in *Proc. USENIX ATC*, 2013, pp. 219–230.

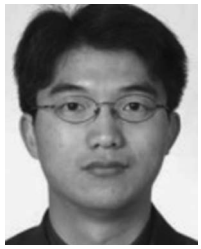


Yuxia Cheng received the B.S. degree in computer science and technology from Hangzhou Dianzi University, Hangzhou, China, in 2010. He is currently working toward the Ph.D. degree in computer science and technology at Zhejiang University, Hangzhou.

His current research interests include operating systems, virtualization technology, and multicore systems.

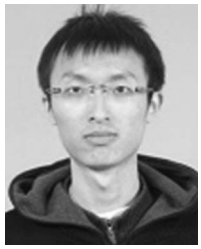
Wenzhi Chen was born in 1969. He received the Ph.D. degree from Zhejiang University, Hangzhou, China.

He is currently a Professor and a Ph.D. Supervisor with the College of Computer Science and Technology, Zhejiang University. His areas of research include computer graphics, computer architecture, system software, embedded systems, and security.



Zonghui Wang was born in March 1979. He received the Ph.D. degree from the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, in 2007.

He is a Lecturer with the College of Computer Science and Engineering, Zhejiang University. His research interests focus on cloud computing, distributed systems, computer architecture, and computer graphics.



Xinjie Yu received the B.S. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2013, where he is currently working toward the M.S. degree in computer science and technology.

His current research interests include operating systems, virtualization technology, and distributed systems.