Workload-aware VM Scheduling on Multi-core Systems

Insoon Jo, Im Y. Jung*, Heon Y. Yeom
School of Computer Science and Engineering
Seoul National University
Seoul, Korea, 151-742
{ischo, iyjung}@dcslab.snu.ac.kr, yeom@snu.ac.kr
*Corresponding Author

***Abstract*—In virtualized environments, performance interference between virtual machines (VMs) is a key challenge. In order to mitigate resource contention, an efficient VM scheduling is positively necessary. In this paper, we propose a workload-aware VM scheduler on multi-core systems, which finds a system-wide mapping of VMs to physical cores. Our work aims not only at minimizing the number of used hosts, but at maximizing the system throughput. To achieve the first goal, our scheduler dynamically adjusts a set of used hosts. To achieve the second goal, it maps each VM on a physical core where the physical core and its host most sufficiently meet the resource requirements of the VM. Evaluation demonstrates that our scheduling ensures efficient use of data center resources.**

Keywords- server consolidation; virtualization; virtual machine scheduling; multi-core systems.

I. INTRODUCTION

The soaring energy consumption by data centers [15, 8, 22] and their low utilization [18, 3, 8] have become main drivers for server consolidation initiatives. Server consolidation using virtualization is a well-known approach to reduce the total number of servers that an organization requires. However, consolidating servers while ensuring performance is very challenging, because current virtualization techniques do not provide effective performance isolation between VMs. Contention for shared resources can cause significant variance in observed system throughput [2, 6, 23]. An efficient VM scheduling is positively needed to mitigate resource contention. Prior works on VM scheduling can be classified into two types according to the scheduling scope. VMs have been scheduled with respect to either other VMs in the current host (within-host) [11, 4, 14, 21, 17] or all VMs in the system (system-wide) [5, 25, 9, 10, 26]. Within-host schedulers focused on how to exploit CPUs in order to maximize throughput of a given host. VM placement considering workload characteristics was out of their concern. System-wide schedulers mainly considered system-wide placement to minimize resource contention between VMs. Contrary to within-host schedulers, they disregarded mapping of VMs to physical cores.

We argue that prior works have missed a good opportunity of cost and performance optimization by disregarding workload-aware VM placement or multi-core systems. Firstly, contention for physical resources impacts performance differently in different workload configurations, causing significant variance in observed system throughput [1, 16, 20, 19, 24]. To this end, characterizing workload that generates resource contention and reflecting such contention in VM placement are important to maximize the system throughput. Secondly, though more and more multi-core CPUs are put into servers in data centers, multi-core systems cannot automatically obtain scalable performance according to the number of cores. Physical cores compete for the interconnect bandwidth, memory controllers, and caches. Therefore, VM schedulers should consider core-level mapping for scalable performance with increasing core counts [13, 12, 21, 17]. Our finding is that two types of scheduling are so complementary that their integration can improve the system throughput.

In this paper, we propose a workload-aware VM scheduler on multi-core systems, which finds a system-wide mapping of VMs to physical cores. To map VMs on cores, we assign a signature to each of VMs and physical cores. A VM signature is a vector representing its resource demands, while a physical core signature is a vector representing remaining resource levels of itself and its host. Our work aims not only at minimizing the number of used hosts, but at maximizing the system throughput. To achieve the first goal, our scheduler dynamically adjusts a set of used hosts. Whenever new jobs arrive or inadequately utilized (i.e., either overloaded or under-loaded) hosts are detected, VM scheduling is triggered. Our scheduler estimates the minimum number of hosts to be able to meet the resource requirements of VMs. Considering current load of each host, it chooses the estimated number of hosts, and turns off hosts not chosen. To achieve the second goal, our scheduler attempts to mitigate

resource contention. Given VMs and chosen hosts, it maps each VM on a physical core where the physical core and its host most sufficiently meet the resource requirements of the VM.

We implemented our scheduler as simulation, and also simulated two types of system-wide schedulers for performance comparison. We ran well-known benchmark applications with various characteristics to demonstrate that our simulation was reasonable. Experimental results illustrate that our scheduling brought a maximum of 10.5% performance improvement compared to prior works.

We discuss related work in Section 2. In Section 3, we describe how our scheduling works. Section 4 presents simulation and experimental results. We conclude with a summary of our work in Section 5.

II. RELATED WORK

A. *Importance of Workload Characterization*

VM schedulers repeat three steps: workload characterization, performance estimation, and scheduling decision. For example, PAC [10] periodically characterizes each VM as repetitive time series of resource consumption, and each host as repetitive time series of remaining resources. It estimates VM's performance on each host by measuring difference between time series of VM and host, and then schedules VM to a host with minimum difference. For VM schedulers, how to characterize workload determines how to estimate VM's performance on each target (either physical core or host). Scheduling decision is just a process of selecting the most suitable target based on estimation.

B. *Prior Approaches*

Most of prior works have characterized VMs as their CPU usages. However, characterization solely based on CPU usages is inappropriate for modeling resource contention, because CPU sharing between VMs is least problematic in virtualized environments. TABLE I represents the benchmark suits we used in this paper, and Fig. 1 shows the performance interference between co-located two Xen VMs.

TABLE I. BENCHMARK SUITS USED IN OUR EXPERIMENTS

Benchmark	***Programs***	***Remarks***
SEPCcpu2006	gobmk, bzip2, mcf, gcc, astar, hmmer, lbm, and etc	CPU, MEM-intensive applications
NPB	cg, mg, ep, lu, sp, bt	CPU + Network
TPC-C	tpc	CPU + I/O
Disk	postmark, copy, bonnie++, iozone	CPU + I/O
netperf	netperf	Network

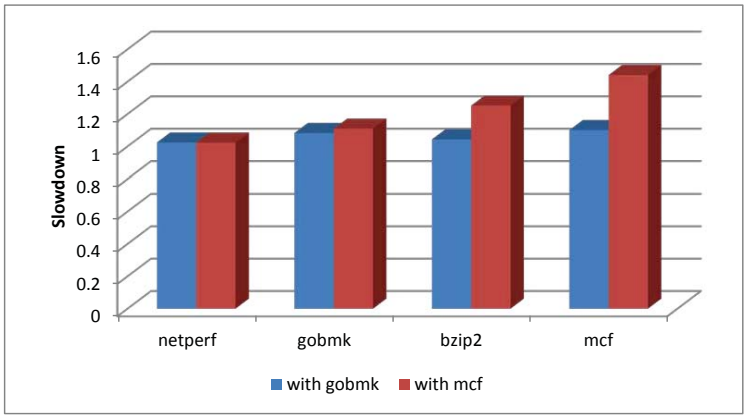

Figure 1. The performance impact of co-locating two VMs in Xen.

In this experiment, each VM ran an application in its entirety, and therefore the behavior of VM represented the behavior of running application. We ran a pair of VMs (VM_1, VM_2) by pinning them to different physical cores. VM_1 ran either *gobmk* or *mcf*, and VM2 ran one of *netperf*, *gobmk*, *bzip2*, and *mcf*. Y-axis values in Fig. 1 represent slowdown of VM_2 when running with VM_1. Note that performance was normalized. Thus, any application running as a stand-alone VM has normalized performance of one. *gobmk* from SEPCcpu2006 is CPU-intensive workload, while *mcf* from the same suite is memory-intensive one. Fig. 1 shows how much CPU-intensive and memory-intensive applications interfere with other applications. *gobmk* had little interference with

all other applications, while running with *mcf* together brought non-trivial variance in application's throughput. In Fig. 1, applications are ordered by their memory utilization. The more applications consumed memory, the more they slowed down. Slowdown reached a maximum of 44%.

To more correctly characterize VMs, some recent works considered contention for multiple shared resources. They characterize each VM as repetitive time series (signatures) of multiple resource consumption [10, 26], and hosts as signatures of available resources. Then, they choose a host whose signature is most similar to that of the VM. These works must incur severe overhead caused from complex signatures. Their signatures are based on time series of resource consumption. Finding repetitive patterns among time series and matching them require intensive computation. Moreover, they focused on choosing the most suitable hosts, but disregarded mapping of VMs to physical cores. Physical cores are another source of performance interference as shown in Fig. 2. In the second experiment, we had a pair of VMs run the same application, and ran them twice with different core mappings. We averaged out runtime of each pair of VMs, and normalized it by runtime as a stand-alone VM. Fig. 2 shows performance impact of core mappings. Four CPU-intensive applications showed performance variance from 6% to 22% according to the core mapping. Other studies have shown similar results. How efficiently to schedule VMs to physical cores brought 16% [21] and 40% [17] performance improvements.

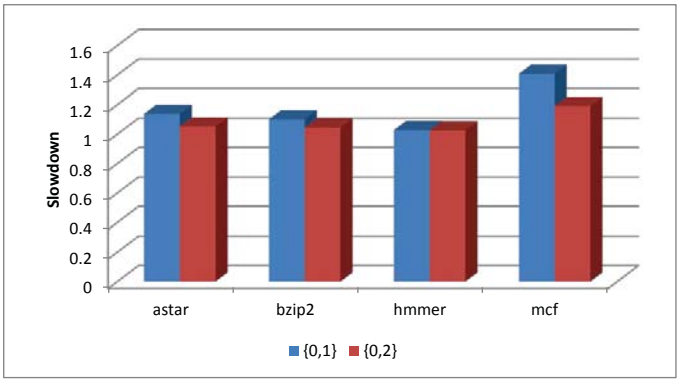

Figure 2. The performance impact of core mappings.

Lim et al. [20] proposed a mathematical model to characterize workload using multiple resource usages. They characterize a host as a collection of *m* resource queues. They also characterize each application as a vector of size *m*, where *i*-th element is calculated as the amount of time using *i*-th resource divided by its runtime when running in stand-alone mode. However, this model is never practical. According to the model, running multiple applications together should not take longer than their sequential execution. This is not true in virtualized environments. Severe contention between two VMs may lead to slowdown of more than twice. The fundamental problem is how to obtain resource vectors. Generally, resource usage is represented as utilization or throughput. Measuring the amount of time using I/O and network is unusual and not easy.

III. DESIGN

In this section we present our scheduling algorithm, and describe how it works.

A. Workload Characterization

We characterize each VM as its signature. Given an application, we start a stand-alone VM, which runs the application in its entirety. We define its signature as a vector whose elements are average values of CPU, memory, I/O, and network consumption during running in stand-alone mode. VM signature represents its resource demands. To accurately capture them, we consider multiple resource consumption. If using monitoring tools like *xentop* and *libxenstat*, we can continuously measure CPU, memory, I/O, and network consumption of a VM without noticeable overhead. By collecting data till the application ends, we can easily obtain time series of each resource consumption during runtime. Time series are put into four groups according to the resource type. To generate VM signature, we just average out time series in each group. Contrary to using time series themselves, using their average values must incur no overhead. Therefore, the key challenge of characterizing workload as average resource consumption is the nature of general workload. If most applications have high variance in time series of resource consumption, performance estimation based on average consumption must cause significant variance in observed system throughput. We argue that most applications have either low variance in time series of resource consumption or no regularity in consuming resources. For the first case, characterizing workload using average consumption is an efficient approach. For the second case, estimating resource consumption based on historical time series must incur severe overhead without providing reasonable throughput. Contrary to this, performance estimation based on average consumption must incur no overhead while providing as much throughput as using time series. Therefore, our estimation can be a reasonable solution

for the second case, too. To justify our argument, we traced 32 applications, which belong to benchmark suits shown in TABLE I. TABLE II shows eleven representative applications among 32. As shown in Fig. 3, applications with intensive workload tended to constantly and fully utilize specific resources.

TABLE II. REPRESENTATIVE WORKLOAD BENCHMARK

Application	*Characteristics (CpuMemDiskNet)*	*Remarks*	*Application*	*Characteristics (CpuMemDiskNet)*	*Remarks*
gobmk	Cd	Intensive workload	*bzip2*	Cm	Mixed workload
mcf	CM	Intensive workload	*gcc*	Cmd	Mixed workload
postmark	D	Intensive workload	*cg*	cmn	Mixed workload
netperf	cN	Intensive workload	*mg*	cmn	Mixed workload
copy	d	Lightly weighted I/O	*tpc*	cmn	Mixed workload
			bonnie++	cd	Mixed workload

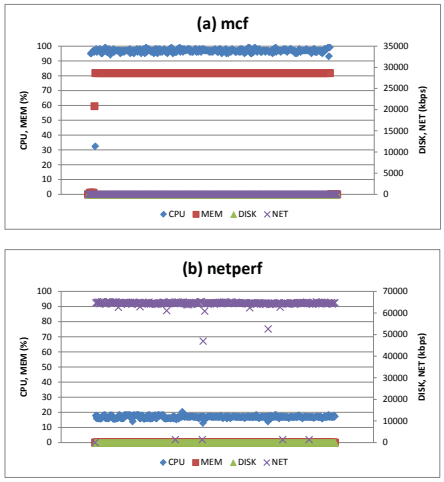

Figure 3. Resource utilization of intensive workload.

Fig. 4 shows resource usages of mixed workload. Some applications ((a) and (b)) showed almost constant consumption activities, while the others ((c), (d), and (e)) showed no regular patterns.

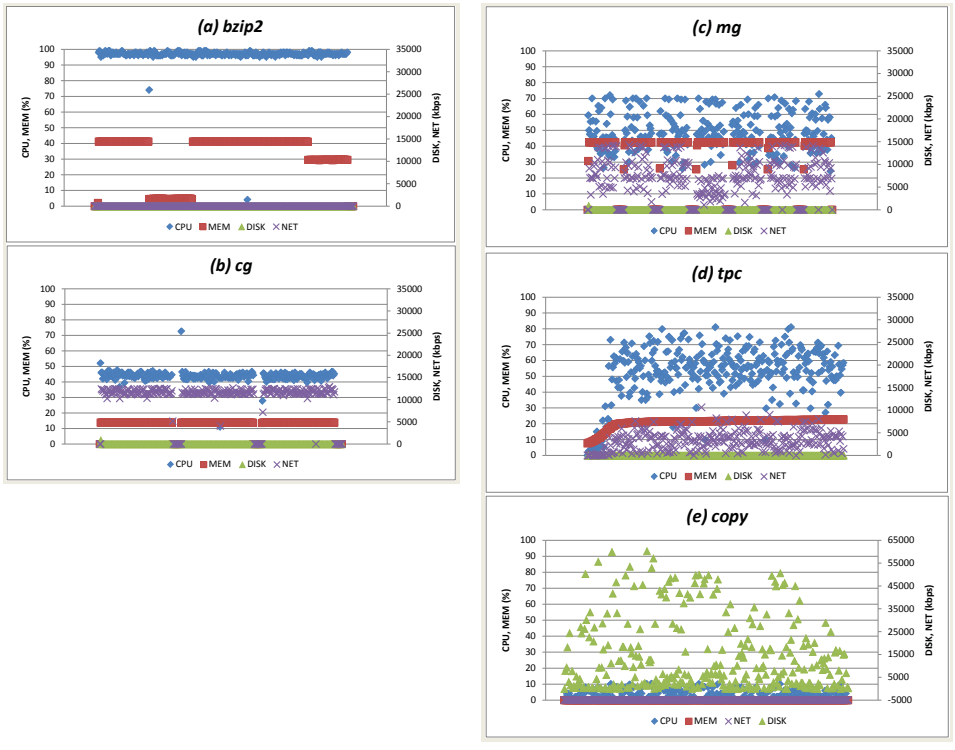

Figure 4. Resource utilization of mixed workload.

To map each VM on a physical core, we also characterize physical cores. For each physical core, we measure its utilization. We also measure memory utilization, I/O throughput, and network throughput of its host. We generate a physical core signature by subtracting current utilization from maximum utilization of each system component. A VM signature is static, since rarely updated once made. Contrary to this, a physical core signature is dynamic. Because it is a vector representing remaining resource levels of itself and its host when scheduling decision is made. Using distance measure between two vectors, our scheduler assigns each VM into a host with a physical core whose signature has the longest distance to the signature of the VM. This is worst-fit assignment, i.e., we map a VM to a physical core where the physical core and its host most sufficiently meet the resource requirements of the VM.

B. Notation and Assumptions

We use the following notations in this paper:

1) *System S is a set of n hosts* $\{h_1, h_2, \cdots, h_n\}$.
2) h_i $(1 \le i \le n)$ *runs zero or more VMs.* h_i *is considered overloaded when its consumption of any resource exceeds an upper threshold* δ_{upper}. *Similarly,* h_i *is considered under-loaded when its consumption of every resource is under a lower threshold* δ_{lower}.
3) $\{uh_i\}$ $(1 \le i \le n)$ *is a set of used hosts. It is a subset of* $\{h_i\}$, *where* uh_i *runs at least one VM. We assume* h_i *in* $\{h_i\} \backslash \{uh_i\}$ *turn off.*
4) h_i *has* nc_i *cores* ($nc_i \ge 1$). c_{ij} $(j \ge 1)$ *means j-th core of* h_i.
5) *The physical core signature* $sigc_{ij}$ *of* c_{ij} *is a vector of size four. First two elements represent available bandwidth of I/O* ($sigc_{ij1}$) *and network* ($sigc_{ij2}$) *of* h_i. $sigc_{ij3}$ *is available memory utilization of* h_i, *and* $sigc_{ij4}$ *is available utilization of* c_{ij}.
6) c_{ij} *is characterized by* $sigc_{ij}$.
7) *Workload W is a set of l applications* $\{a_1, a_2, \cdots, a_l\}$. *We assume dynamic workload, and therefore l changes over time. W has one or more instances of some applications, i.e.,* a_i *and* a_j $(1 \le i, j \le l)$ *may be instances of the same application even though* $i \ne j$.
8) *To obtain signature,* VM_i *runs* a_i *in stand-alone mode. The VM signature* $sigv_i$ *of* VM_i *is a vector of size four. First two elements are average throughput of I/O* ($sigv_{i1}$) *and network* ($sigv_{i2}$) *during runtime. The last two are average utilization of memory* ($sigv_{i3}$) *and CPU* ($sigv_{i4}$) *during runtime.*
9) VM_i *is characterized by* $sigv_i$.
10) τ_i *denotes a runtime of* a_i *when running with other VMs together.*
11) $\tau_W = max\{\tau_1, \tau_2, ..., \tau_l\}$. τ_W *means a completion time of W.*

C. Scheduling of VMs

We define VM scheduling as a problem to find a mapping $\{(VM_i, c_{jk})\}$ $(1 \le i \le l, 1 \le j \le n, 1 \le k \le nc_j)$, which reasonably lengthens τ_W by consolidation. We operate VM scheduling in the following two cases: (i) when new jobs arrive (ii) when inadequately utilized hosts are detected. Algorithm 1 shows how our scheduler works.

First, it estimates the minimum number of hosts to be able to meet the resource requirement of VMs. VM placement has been considered an instance of bin packing problem, the problem of assigning a set of items (VMs) into a set of bins (hosts) while minimizing the number of bins in use. Bin packing problem is known as a combinatorial NP-hard problem, and best-fit, first-fit, and worst-fit heuristics are among the simplest heuristic algorithms for solving it. Best-fit heuristic is simplest to use for finding out the exact number of bins to accommodate a set of items, and therefore our scheduler operates *Best Fit Decreasing (BFD)* heuristic to estimate the minimum number of used hosts. *BFD* works by first sorting the list of elements into decreasing order, and then placing each item into the bin with the least amount of space left. To operate *BFD*, our scheduler sorts VM signatures into decreasing order. As we noted in the previous Subsection, VM signature $sigv_i$ of VM_i is a vector of four elements. Our scheduler sorts VM signatures by one element and then the other according to the order of vector indices. Then, it starts best-fit assignment with an empty set of used hosts. Using distance measure between two vectors, it maps each VM in decreasing order on a physical core where the physical core and its hosts most closely meet the resource requirements of the VM. Given VM_i, *calcDistance* calculates the distance between two vectors by Euclidean distance in 4-dimensional space. If a physical core and its host do not meet the resource requirements of VM_i, the distance is set infinite. The shorter the distance is, the more closely a physical core and its host meet the resource requirements of VM_i. If the distance between VM_i and c_{xy} is shortest, VM_i is pinned to c_{xy}. If all distances are infinite, our scheduler adds a new host to the set, and maps VM_i on the first physical core of new host. To simplify estimation, we assume that each host has no load before entering the set of

used hosts. *initializeSignature* sets every physical core signature of new host maximally available state. Because a new assignment increases the loads of c_{xy} and its host h_x, the signature of c_{xy} should be updated. *updateSignature* updates the physical core signature $sigc_{jk}$ of c_{jk}. We measure the utilization of c_{jk}. We also measure memory utilization, I/O throughput, and network throughput of h_i. By subtracting current utilization from maximum utilization of each system component, we update $sigc_{jk}$. After mapping the last VM, the size of the set of used hosts (i.e., $|\{sh_j\}|$) is what we want, the minimum number of used hosts to be able to accommodate $\{VM_i\}$.

ALGORITHM 1. VM SCHEDULING

```
input: {VM_i} (1 ≤ i ≤ l)          /* VMs of changed workload W */
{sigv_i}                           /* signatures of {VM_i} */
{h_j}(1 ≤ j ≤ n)                   /* all hosts in S */
{c_jk}(1 ≤ k ≤ nc_j)               /* physical cores of {h_j} */
{(VM'_i, c'_jk)}                   /* current mapping */
{uh'_j} (1 ≤ j ≤ n)                /* used hosts */
output: Re-arrangement of {VM_i} by an adjusted mapping {VM_i, c_jk} which minimizes τ_W

/* estimate the minimum number of used hosts for {VM_i} using BFD */
sort {VM_i} into decreasing order of sigv_i;
{sh_j} =ϕ;
foreach VM_i in decreasing order do
    foreach sh_j do
        foreach c_jk do
            if remaining physical memory of sh_j is sufficient for domain memory of VM_i then
                d_ijk = calcDistance(sigv_i, sigc_jk);

    if |{sh_j}| == 0 || every d_ijk is infinite then
            add a new host h_x to {sh_j};
        foreach c_xk do  /* initialize physical core signatures of h_x */
                initializeSignature(sigc_xk);
            d_ix0 = calcDistance(sigv_i, sigc_x0);

        map VM_i to c_xy whose d_ixy is shortest;
    sigc_xy = updateSignature(c_xy, h_x);

/* obtain the estimated number, and adjust it for worst-fit assignment */
n_hosts = |{sh_j}|;
foreach non-negative integer z do
    assign {VM_i} to n_hosts+z hosts using WFD;
  if {VM_i} fits in n_hosts+ z hosts then
      n_hosts += z;
        update {(VM_i, c_jk)} by this worst-fit mapping;
        break;

/* choose the estimated number of hosts considering current load */
{uh_j} = chooseUsedHosts({(VM'_i, c'_jk)}, n_hosts);
turn on hosts in {uh_j}\{uh'_j};

/* apply the worst-fit mapping to chosen hosts */
foreach (VM_i, c_jk) do
  if VM_i arrives now then
      start VM_i by pinning on c_jk;
  else if (VM_i, c_jk) is not in {(VM'_i, c'_jk)} then
      if c_jk and c'_jk belong to different hosts then
            migrate VM_i to h_j and pin it on c_jk;
      else
            pin VM_i on c_jk;

/* turn off unused hosts and update status */
turn off hosts in {uh'_j}\{uh_j};
set {(VM_i, c_jk)} current mapping, and {uh_j} used hosts;
```

Secondly, our scheduler adjusts the estimated number so as to accommodate VMs by worst-fit heuristic. To mitigate resource contention between VMs, we prefer worst-fit assignment to other-fit assignments. Worst-fit assignment more evenly distributes VMs to hosts than others, and therefore the former can deliver better performance than the latter. Therefore, our scheduler obtains the initial size by best-fit heuristic, and then slightly adjusts it for worst-fit assignment. At this time, it operates *Worst Fit Decreasing (WFD)* heuristic. *WFD* works by first sorting the list of elements into decreasing order, and then placing each item into the most amount of space left. After sorting VM signatures into decreasing order, our scheduler starts worst-fit assignment with a set of used hosts with the initial size. Using the distance measure between two vectors, it maps each VM in decreasing

order on a physical core where the physical core and its hosts most sufficiently meet the resource requirements of the VM. If no physical core and its host do not meet the resource requirements of VM_i, it aborts assignment. After increasing the number of used hosts by one, it restarts *WFD* from the first VM. If the assignment is safely done, our scheduler obtains the number of hosts to be adjusted (i.e., n_{hosts}) and the worst-fit mapping of VMs to n_{hosts} hosts (i.e., $\{(VM_i, c_{jk})\}$).

Lastly, our scheduler implements the worst-fit mapping obtained in the second step. For dynamically relocating VMs according to the mapping, we utilize live migration [7]. We assume that all VM images are held in NFS server, and the overhead caused from live migration is low. It took less than 20 seconds to migrate a VM with 2GB memory from one host to another using 1Gbps Ethernet [19]. *chooseUsedHosts* chooses a new set of used hosts $\{uh_j\}$ with size n_{hosts} as mapping target. In order to minimize live migration, our scheduler considers current load of each host (i.e., $\{(VM'_i, c'_{jk})\}$), and mainly selects highly-utilized hosts. Before applying mapping to chosen hosts, it turns on hosts in $\{uh_j\}\backslash\{uh'_j\}$, because they are unused and turn off. Then, it rearranges $\{VM_i\}$ in $\{uh_j\}$ according to $\{(VM_i, c_{jk})\}$. If $\{(VM_i, c_{jk})$ is not changed, it does nothing. If VM_i is a new job, it pins VM_i on c_{jk}. If VM_i is active and should be placed on different host, it migrates VM_i to h_j and pins VM_i on c_{jk}. Otherwise, it re-pins VM_i on c_{jk}. To save energy, it turns off hosts in $\{uh'_j\}\backslash\{uh_j\}$, because they will be used no more.

IV. EVALUATION

A. Experimental Setup

We used Intel Quad-core 2.83Ghz machines with 8GB memory, which were connected to 1Gbps Ethernet switch. Our machines ran Ubuntu 10.04.3 LTS with Xen 4.0.1 hypervisor. We configured each VM with 1 VCPU, 2GB of memory, and 20GB of VM image. All VM images are held in NFS server for live migration, and VMs use NFS storage rather than local disk. The benchmark suits used in our experiments are already shown in Table I. We used SEPCcpu2006, NPB (NAS Parallel Benchmarks), TPC-C, various disk benchmarks, and netperf. SEPCcpu2006 is an industry-standardized CPU intensive benchmark suits, comprised of real-world applications intensively computing integer or floating point operations. NPB is widely used to evaluate parallel machines. TPC-C is comprised of transaction processing and database applications, which are popularly used to evaluate the performance of the OLTP systems. We traced all 32 applications from benchmark suits, and defined their signatures using average resource consumption during running in stand-alone mode.

B. Monitoring Framework

There are many monitoring tools for VM performance analysis like *libxenstat*, *xenopof*, and *xentrace*, which provide a huge number of different metrics on fine-grained events. However, we were only interested in per-VM resource utilization. Therefore, we chose *xentop* as base monitoring tool. Per domain, we monitored the following performance data every 2 second: CPU utilization, memory utilization, I/O throughput (reads and writes), and network throughput (sent and received). We put time series of gathered data into four groups, and averaged out time series in each group.

C. Simulation Results

We implemented our scheduler as simulation according to Algorithm 1. For performance comparison, we also simulated system-wide schedulers using *BFD* and *WFD* heuristics. We simulated system-side schedulers based on average resource consumption of VMs and available resources of hosts. In our simulation, each host can create a maximum of eight VMs. First, the number of VMs a host can create and use is limited by its physical memory size. In other words, it cannot create a set of VMs whose total memory size exceeds its physical memory size. Secondly, assigning multiple VMs to one physical core is common. For example, standard instances of Amazon EC2 assign one, two, or four virtual cores to each physical core. Therefore, assigning a maximum of eight VMs to a host is reasonable assumption, as our machines have quad-cores. We randomly ordered 32 applications from benchmark suits. We simulated that they arrived one at a time, and none of them finished till the last application arrived.

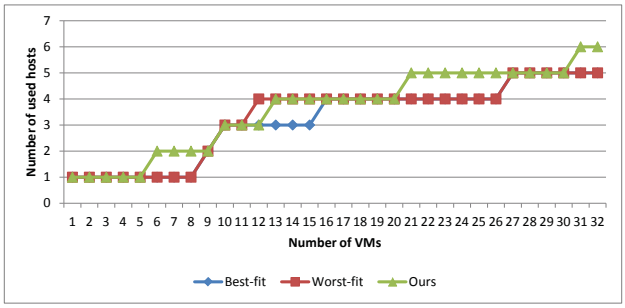

Figure 5. The number of used hosts as a function of the number of VMs.

Fig. 5 shows how the number of used hosts increased as more and more jobs arrived in system. In all cases, best-fit scheduling needed the minimum number of hosts. Though our scheduler is based on worst-fit heuristic, the required number of hosts did not significantly differ from best-fit scheduler to ours. As shown in the figure, the difference was at most one.

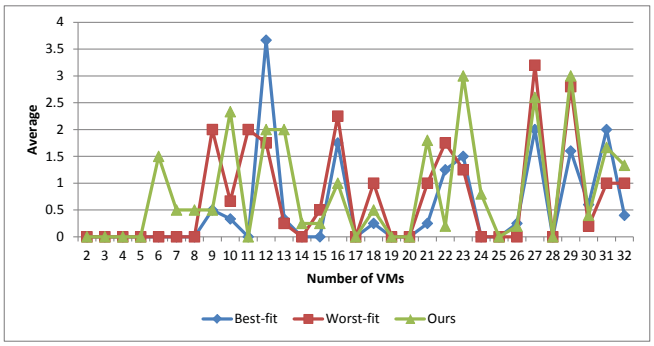

Figure 6. Average number of migrations per host as a function of the number of VMs.

In our simulation, whenever new jobs arrive, VM scheduling is triggered. Each VM scheduler estimates the number of used hosts, and rearranges VMs in newly-chosen set of hosts. This rearrangement may include movement of VMs between hosts, which is carried out by live migration. Performance degradation caused from resource contention can be severe. Therefore, live migration is considered unavoidable in virtualized environments, besides, the overhead caused from a single live migration is low. Fig. 6 shows average number of migrations each host performed when a new job arrived. Fortunately, these numbers did not linearly increase, though the number of VMs in the system increased.

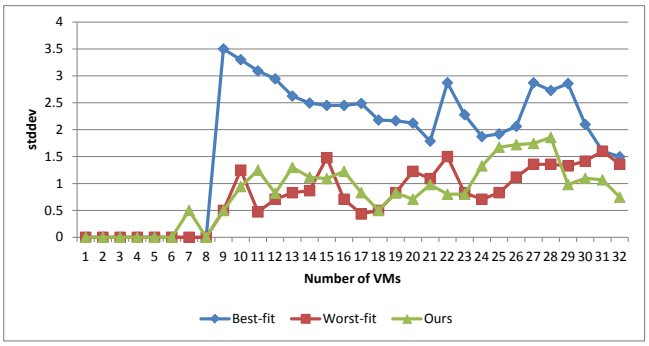

Figure 7. Standard deviation of the number of VMs per hosts.

To maximize overall performance in virtualized environments, contention for shared resources should be minimized. To mitigate resource contention, our scheduler not only adopted worst-fit heuristic, but considered physical cores. Fig. 7 represents the standard deviation of the number of VMs assigned to each host. As the number of VMs increases, best-fit scheduler shows the highest deviation among three. This means that the number of VMs each host ran was skewed, because it tends to more unevenly distribute VMs to hosts than worst-fit heuristic.

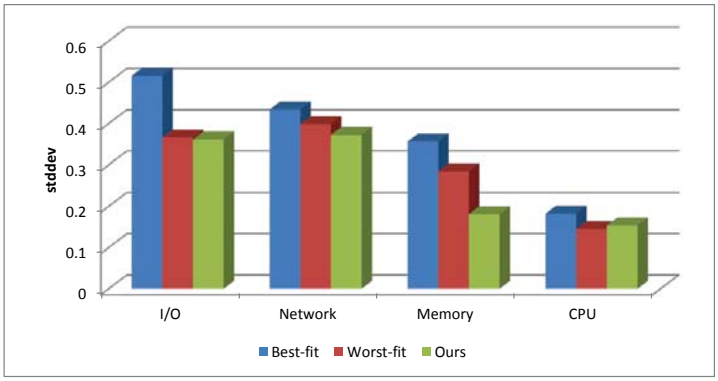

Figure 8. Standard deviation of resource utilization per host.

Fig. 8 illustrates the resource contention caused from this uneven distribution. This figure represents standard deviation of resource utilization of each host when all 32 VMs were running. As for best-fit scheduler, its uneven distribution caused more contention for I/O and memory resources than the other schedulers, which can lead to severe performance degradation in virtualized environments. Ours shows the lowest deviation for most types of resources. This means that our scheduler tends to most appropriately distribute VMs to each host, resulting in mitigating overall resource contention.

D. Result from Running Benchmark Applications

To demonstrate that our simulation was reasonable, we scheduled a set of seven benchmark applications using real machines. Our machines have a total of 8GB physical memory. Each VM image was configured with 2GB memory, and Xen hypervisor also consumes physical memory. Therefore, our machines can create a maximum of three VMs. All schedulers assigned seven VMs to three machines. Fig. 9 illustrates that different scheduling schemes lead to different slowdown results. Note that performance was normalized. Thus, any application running as a stand-alone VM has normalized performance of one. As shown in the figure, applications experienced the largest slowdown under best-fit scheduling, while the smallest slowdown under ours. Average slowdowns of best-fit, worst-fit, and ours were 1.305, 1.232, and 1.168, respectively. Therefore, our scheduler improved performance by 10.5% compared to best-fit scheduler, and 5.2% compared to worst-fit scheduler.

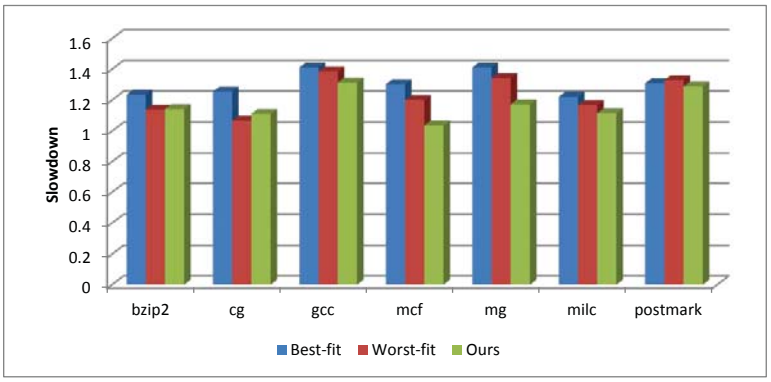

Figure 9. Slowdown of each application as a function of scheduling scheme.

V. CONCLUSION AND FUTURE WORK

By disregarding workload-aware VM placement or multi-core systems, prior works on VM scheduling have missed a chance for better optimizing cost and performance. To address this problem, this paper proposes a workload-aware VM scheduler on multi-core systems, which finds a system-wide mapping of VMs to physical cores. Our scheduler has strengths in cost and performance compared to prior works. First, it minimizes the number of used hosts by adjusting a set of used hosts according to time-varying workload. Secondly, it mitigates resource contention by conducting worst-fit mapping of VMs to physical cores. Experimental results illustrate that our scheduling brought a maximum of 10.5% performance improvement compared to prior works.

In the future, we would like to conduct an in-depth analysis of performance interference in virtualized environments, and describe it using a quantitative model. By being integrated with this model, our scheduler can more accurately estimate VM's performance on each host, and provide a more efficient VM scheduling.

ACKNOWLEDGMENT

This research was supported by the KCC(Korea Communications Commission), Korea, under the CPRC(Communications Policy Research Center) support program supervised by the KCA(Korea Communications Agency) (KCA-2011-1194100004-110010100)

REFERENCES

[1] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, "Characterization & analysis of a server consolidation benchmark," In 4th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE'08), pp. 21–30, March 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Communications of the ACM, 53(4):50–58, April 2010.

[3] L. A. Barroso, and U. Holzle, “The case for energy-proportional computing,” Computer, 40(12):33–37, December 2007.

[4] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato, “A simple power-aware scheduling for multicore systems when running real-time applications,” In 2008 IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), April 2008.

[5] N. Bobroff, A. Kochut, and K. Beaty, “Dynamic placement of virtual machines for managing sla violations,” In 10th IFIP/IEEE International Symposium on Integrated Network Management (IM2007), pp. 119–128, May 2007.

[6] R. Buyya, A. Beloglazov, and J. Abawajy, “Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges,” In 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010), July 2010.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hanseny, E. July, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” In 2nd conference on Symposium on Networked Systems Design and Implementation (NSDI'05), pp. 273–286, May 2005.

[8] W. Forrest, and K. Brill, “The problem of power consumption in servers,” 2008, http://www.mckinsey.com/clientservice/bto/pointofview/Revolutionizing.asp.

[9] D. Gmach, J. Rolia, and L. Cherkasova, “Satisfying service level objectices in a self-managing resource pool,” In 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009), pp. 243–253, September 2009.

[10] Z. Gong, and X. Gu, “Pac: Pattern-driven application consolidation for efficient cloud computing,” In 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10), pp. 24–33, August 2010.

[11] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing performance isolation across virtual machines in xen,” In 7th ACM/IFIP/USENIX Middleware Conference (MIDDLEWARE 2006), pp. 342–362, November 2006.

[12] M. H. Jamal, A. Qadeer, W. Mahmood, A. Waheed, and J. J. Ding, “Virtual machine scalability on multi-core processors based servers for cloud computing workloads,” In 2009 IEEE International Conference on Networking, Architecture, and Storage (NAS 2009), pp. 90–97, July 2009.

[13] N. E. Jerger, D. Vantrease, and M. Lipasti, “An evaluation of server consolidation workloads for multi-core designs,” In IEEE 10th International Symposium on Workload Characterization (IISWC 2007), pp. 47–56, September 2007.

[14] E. Kalyvianaki, T. Charalambous, and S. Hand, “Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters,” In ICAC'09, pp. 117–126, June 2009.

[15] J. G. Koomey, “Estimating total power consumption by servers in the u.s. and the world,” Technical report, Oakland, CA, 2007.

[16] A. Kvalnes, D. Johansen, P. Halvorsen, and C. Griwodz, “Support for enterprise consolidation of i/o bound services,” SOFTWARE - PRACTICE AND EXPERIENCE, 40(12):1035–1051, December 2010.

[17] Y. Kwon, C. Kim, S. Maeng, and J. Huh, “Virtualizing performance asymmetric multi-core systems,” In 38th International Symposium on Computer Architecture (ISCA 2011), June 2011.

[18] C. Lefurgy, X. Wang, and M. Ware, “Server-level power control,” In 4th International Conference on Autonomic Computing (ICAC'07), June 2007.

[19] S.-H. Lim, J.-S. Huh, Y. Kim, and C. Das, “Migration, assignment, and scheduling of jobs in virtualized environment,” Technical report, August 2011.

[20] S.-H. Lim, J.-S. Huh, Y.-J. Kim, G. M. Shipman, and C. R. Das, “A quantitative analysis of performance of shared service systems with multiple resource contention,” Technical report, 2010, http://www.cse.psu.edu/research/publications/tech-reports/2010/cse-10-010.pdf.

[21] H. Lv, X. Zheng, Z. Huang, and J. Duan, “Tackling the challenges of server consolidation on multi-core systems,” In 2010 IEEE International Symposium on Workload Characterization (IISWC-2010), December 2010.

[22] L. Minas, and B. Ellison, “Revolutionizing data center efficiency,” 2009, http://drdobbs.com/215800830.

[23] I. S. Moreno, and J. Xu, “Energy-efficiency in cloud computing environments: Towards energy savings without performance degradation,” International Journal of Cloud Applications and Computing (IJCAC), 1(1), 2011.

[24] F. Y.-K. Oh, H. S. Kim, H. Eom, and H. Y. Yeom, “Enabling consolidation and scaling down to provide power management for cloud computing,” Technical report, August 2011.

[25] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, “Server workload analysis for power minimization using consolidation,” In 2009 Conference on USENIX Annual technical conference (USENIX'09), June 2009.

[26] Q. Zhu, J. Zhu, and G. Agrawal, “Power-aware consolidation of scientific workflows in virtualized environments,” In 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10), pp. 24–33, November 2010.

AUTHORS PROFILE

Insoon Jo is a PhD candidate of the School of Computer Science and Engineering of Seoul National University in Korea. She received her BS and MS degrees in computer science from the same university in 2004 and 2006, respectively. Her major research interests are distributed systems, usable security, and cloud computing.

Dr. Im Y. Jung received the first B.S. degree in chemistry from Pohang University of Science and Technology in 1993 and the second B.S degree in computer science from Seoul National University in 1999. And she received her M.S. degree in computer science and engineering from Seoul National University in 2001. Since February 2001, she had been a researcher for 3 years in the Electronics and Telecommunications Research Institute(ETRI), South Korea. She was granted Ph.D in Computer Engineering from Seoul National University in 2010. Her current research interests include distributed computing system, IT convergence, data and system security, and storage system.

Heon Y. Yeom is a professor with the School of Computer Science and Engineering of Seoul National University in Korea. He received his BS degree in computer science from Seoul National University in 1984 and received the MS and PhD degrees in computer science from Texas A&M University in 1986 and 1992, respectively. From 1992 to 1993, he was with Samsung Data Systems as a researcher. He joined

the Department of Computer Science of Seoul National University in 1993, where he currently teaches and researches on distributed systems, cloud computing and power-saving systems, etc.