iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud

Fei Xu, Fangming Liu, *Member, IEEE,* Linghui Liu, Hai Jin, *Senior Member, IEEE,* Bo Li, *Fellow, IEEE,* Baochun Li, *Senior Member, IEEE*

Abstract—Large-scale datacenters have been widely used to host cloud services, which are typically allocated to different virtual machines (VMs) through resource multiplexing across shared physical servers. While recent studies have primarily focused on harnessing live migration of VMs to achieve load balancing and power saving among different servers, there has been little attention on the incurred performance interference and cost on both source and destination servers during and after such VM migration. To avoid potential violations of service-level-agreement (SLA) demanded by cloud applications, this paper proposes *iAware*, a lightweight *interference-aware* VM live migration strategy. It empirically captures the essential relationships between VM performance interference and key factors that are practically accessible, through realistic experiments of benchmark workloads on a *Xen* virtualized cluster platform. *iAware* jointly estimates and minimizes both migration and co-location interference among VMs, by designing a simple multi-resource demand-supply model. Extensive experiments and complementary large-scale simulations are conducted to validate the performance gain and runtime overhead of *iAware* in terms of I/O and network throughput, CPU consumption and scalability, compared to the traditional interference-unaware VM migration approaches. Moreover, we demonstrate that *iAware* is flexible enough to cooperate with existing VM scheduling or consolidation policies in a complementary manner, such that the load balancing or power saving can still be achieved without sacrificing performance.

Index Terms—Cloud computing, virtualization, live migration, performance interference.

1 Introduction

Virtualization is widely deployed in large-scale datacenters, due to its ability to isolate co-located application workloads, and its efficiency for resource multiplexing. Most real-world Infrastructure-as-a-Service (IaaS) cloud platforms [1], such as Amazon Elastic Compute Cloud (EC2) [2], take advantage of virtualization to provide flexible and economical computing capacity for tenants, while statistically guaranteeing the service-level-agreement (SLA) in terms of throughput, delays, and the successful requests served. In particular, live migration of running virtual machines (VMs) [3] across distinct physical machines (PMs) serves as the *cornerstone* of realizing load balancing [4] and power saving [5]–[7] functions in such modern datacenters.

There are two types of underlying performance interference that cannot be overlooked, on both the migration source and destination servers, *during* and *after* live migrations. (1) *Migration interference*. It has been observed from realistic applications that during the migration process of VM(s), the migrated VM(s) and the other running VMs hosted on both source and destination PMs could undergo severe performance degradation [8], [9]. (2) *Co-location interference*. Current virtual machine monitor (VMM) like *Xen* [10] only provides resource isolation among co-located VMs by CPU core reservations and static partition of memory and disk capacities. However, it has been shown that resource isolation does not imply performance isolation between VMs [11]. Meanwhile, resources like cache space, memory bandwidth and the interconnection network are very hard to be isolated in practice [12]. Hence, even with built-in performance isolation mechanisms across VMs, it is not uncommon for the resumed VM after migration and other co-located VMs at the destination PM to suffer from additional performance losses [5], due to potential resource contention. As a result, it is essential to make correct VM migration decisions, such that both migration and co-location interference can be alleviated *in a holistic manner*, in order to avoid potential violations of the cloud SLA.

Recently, there have been a number of studies on VM migration, which primarily focused on the migration costs of an *individual* migrated VM, in terms of the duration, downtime, bandwidth and power consumption [13], as well as SLA violations on the migrated VM [7], [14]. There has been little attention on the *mutual* interference among a cluster of VMs across migration source and destination PMs, during and after the migration. There have also been investigations on VM migration and VM co-location interference *separately*, with respect to multi-dimensional resources such as the cache,

- *Fei Xu, Fangming Liu, Linghui Liu and Hai Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology. The corresponding email address is* fmliu@mail.hust.edu.cn.
- *Bo Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His email address is* bli@cse.ust.hk.
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto. His email address is* bli@eecg.toronto.edu.

Manuscript received September XX, 2012; revised August XX, 2013.

memory bandwidth [15] and network bandwidth [16]. However, there has been no systematical study on *jointly* measuring, analyzing and mitigating both the VM migration interference and co-location interference. As a result, though a VM migration operation itself might not incur considerable cost, it is possible that the VMs hosted on a chosen destination PM result in performance interference with the migrated VMs during and after the migration process, which brings additional costs eventually.

To address these challenges, in this paper, we present *iAware*, a lightweight *interference-aware* VM live migration strategy. It empirically captures the essential relationships between VM performance interference and key factors that are practically accessible, through realistic experiments of benchmark workloads on a *Xen* virtualized cluster platform. *iAware* jointly estimates and minimizes both migration and co-location interference among VMs, by designing a simple *multi-resource demand-supply* estimation model that unifies the key factors. Such factors include multi-dimensional resource utilization measured at both PMs and VMs, the number of co-located VMs, network interrupts inside VMs, cache miss ratios in PMs and VMs, as well as the I/O and VCPU scheduling mechanisms adopted in the VMM (*e.g., Xen*).

We conduct extensive experiments and complementary large-scale simulations to evaluate the performance and runtime overhead of *iAware* in terms of I/O and network throughput, CPU consumption and its scalability, compared with traditional interference-unaware VM migration approaches. We demonstrate that under a mix of representative workloads, *iAware* can qualitatively estimate VM performance interference, and improve: (1) I/O and network throughput by 65% for network-intensive workloads, and (2) execution durations and the total amount of successfully served requests by 16%-28% for workloads that are CPU, memory and network-intensive, in comparison to the First-Fit Decreasing (FFD) [7] and Sandpiper [4] algorithms. Meanwhile, the runtime overhead of *iAware* in terms of CPU consumption and network I/O usage is shown to be practically acceptable, even when it scales to thousands of VMs.

To the best of our knowledge, *iAware* represents the first attempt to design and implement an interference-aware VM live migration strategy, with a particular focus on characterizing and minimizing the performance interference during and after the migration of VMs. Moreover, our strategy is sufficiently flexible to cooperate with existing VM scheduling or consolidation policies in a complementary manner, in order to achieve load balancing [17] or power saving [18]–[20] without sacrificing performance.

2 UNDERSTANDING VM INTERFERENCE

In this section, we first seek to understand the following questions: *how severe the performance interference could be during and after the migration of VMs, and what are the key factors that impact such performance interference?*

TABLE 1: Representative benchmark workloads of datacenters.

Workloads	Type	Programs
SPECCPU2006 [21]	CPU, memory-intensive	gobmk, mcf
netperf [22]	Network-intensive	netperf TCP
Hadoop [23]	CPU, network-intensive	TeraSort
NASA parallel benchmark [24]	CPU, memory, and network-intensive	ft
SPECweb2005 [25]	CPU, memory, and network-intensive	banking

We build a *Xen* [10] virtualized cluster platform based on 10 enterprise-class PMs to capture VM performance interference, under a variety of typical datacenter application workloads as characterized in Table 1. Specifically, each PM is equipped with 2 quad-core Intel Xeon E5620 2.40 GHz processors, 12 MB shared last level cache, 24 GB memory, 800 GB network file system (NFS) storage, and two 1 Gbps Ethernet interfaces used for the NFS storage traffic and VM application traffic (including the VM migration traffic), respectively. All PMs are connected to a 1 Gbps network switch, forming a one-tier tree network topology. The PMs are running CentOS 5.5 distribution and the Linux 2.6.18.8-*Xen* kernel patched with *Xen* 4.1.1 [10], which is an open-source VMM responsible for creating multiple guest domains (*i.e.,* VMs) on a PM. The VMs are running CentOS 5.5 with Linux 2.6.18.8 kernel, and they can access the resources of PMs via a privileged domain called *domain-0*. All image files of VMs, which maintain the operating system and user data of VMs, can be accessed through NFS storage.

TABLE 2: VM types and parameter setting of *Xen* Credit Scheduler [26].

VM Type	VCPUs	Memory	*weight*	*cap*
Small VM instance	1	1.7 GB	256	100
Large VM instance	4	7.5 GB	256	400
domain-0	1	2 GB	512	0

In accordance to the production standard of VM instances in Amazon EC2 [2], the VMs in our platform are configured as two representative classes: the *standard small VM instance* with 1 virtual CPU core (VCPU) and 1.7 GB memory versus the *standard large VM instance* with 4 VCPU cores and 7.5 GB memory. The *domain-0* runs on 1 VCPU core and 2 GB memory. To share the physical CPU cores (PCPUs) among VMs including *domain-0*, we adopt the *Xen* Credit Scheduler [26] with the setting of *weight* and *cap* parameters listed in Table 2. In particular, to ensure that *domain-0* gets enough CPU time for serving I/O requests from VMs, we assign a double *weight* (*i.e.,* 512) than other types of VMs and zero upper *cap* of CPU cycles to *domain-0* [27]. Meanwhile, as *Xen* does not support memory over-subscription in production, we allocate an amount of non-overlapped memory to each domain as in [28].

Without loss of generality, we use a four-tuple array $VU\ \langle VU_c, VU_m, VU_i, VU_o\rangle$ to denote the multi-dimensional *virtual resource utilization ratio* (including CPU, memory, network in- and out-throughput) and as

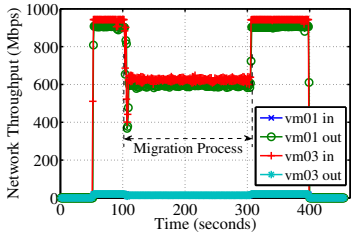


Fig. 1: Network throughput of the netperf TCP application hosted on two large VM instances, during the live migration of another large idle VM instance.

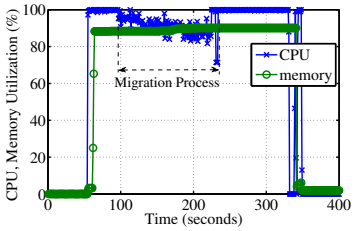


Fig. 2: VU of the large VM instance hosting the mcf application on the migration source PM, during the live migration of a large VM instance running the mcf application.

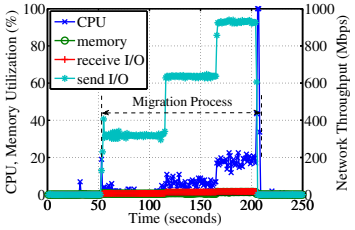


Fig. 3: VU of the *domain-0* on the migration source PM, during the live migration of a large VM instance running the mcf application.

a ratio of the maximal resource capacity of a VM, where $VU_c, VU_m, VU_i, VU_o \in [0,1]$. Correspondingly, we use another four-tuple array PU $\langle PU_c, PU_m, PU_i, PU_o \rangle$ to denote the multi-dimensional *physical resource utilization ratio* and as a ratio of the maximal resource capacity of a PM, where $PU_c, PU_m, PU_i, PU_o \in [0,1]$. For simplicity, we calculate PU as the ratio of the aggregate resource consumption of all hosted VMs (including *domain-0*) to the corresponding physical resource capacity of the PM.

We develop a resource tracing tool to record VU inside each VM, by acquiring the resource utilization information of CPU, memory and network bandwidth from the Linux 'proc' file system (including '/proc/stat' file, '/proc/net/dev' file and '/proc/meminfo' file). Furthermore, we capture the statistics of cache misses and cache requests in PMs and VMs using the system-wide statistical profiler for Linux systems, called *Oprofile* v0.97 [29], patched with the *xenoprof* (Oprofile-0.95-xen-patch) [30] customized for *Xen* VMs. In particular, the cache miss ratio is calculated as the statistics of cache misses divided by that of cache requests in PMs or VMs.

2.1 VM Migration Interference

We measure VM migration interference with representative applications categorized in Table 1. To reserve bandwidth for other VM applications, we enable the rate limiting mechanism [3] in *Xen*, by restricting the network bandwidth for VM migration within 300-600 Mbps.

To emulate a network-intensive scenario, we launch two large VM instances denoted by *vm01* and *vm02* on one PM. Specifically, *vm02* is idle and *vm01* is running the netperf [22] application to transmit a TCP stream to another large VM instance *vm03* hosted on another PM. Fig. 1 plots the network throughput of *vm01* and *vm03*, when *vm02* is migrated to the PM hosting *vm03*. We observe that the VU_o of *vm01* and the VU_i of *vm03* have dropped substantially, from 925 Mbps to around 600 Mbps, during the live migration process of *vm02*. Such performance degradation lasts for up to 203 seconds due to the memory transmission of a large idle VM instance, and it will become worse when migrating a VM with a high memory dirtying rate. Even the migration of one small idle VM instance can last for about 54 seconds in this scenario. This quantitatively shows the severe VM migration interference on both migration source and destination PMs, even with the rate limiting mechanism enabled in *Xen*. Though [11] has achieved performance isolation of network bandwidth between VMs, it actually sacrifices the performance of some victim VMs. In case that the network bandwidth for VM migration is restricted as a victim, the migrated VM could undergo a long period of migration process and downtime, and thus experience severe performance degradation.

We further examine VM migration interference in the CPU and memory-intensive scenarios, by carrying out the mcf application of SPECCPU2006 [21] in *vm01*, *vm02* and *vm03*. To exclude the co-location interference, we pin VCPUs of *vm01* and *vm02* separately to the two CPU processors on the source PM. After the live migration of *vm02*, the VCPUs of *vm02* and *vm03* are correspondingly pinned to the two CPU processors on the destination PM. Our experimental result has shown that the execution durations of the mcf application on *vm01* and *vm03* are extended by 45-50 seconds due to the live migration of *vm02*. Furthermore, Fig. 2 reveals that the VU_c of *vm01* is moderately affected by the migration of *vm02*, while the VU_m of *vm01* remains unchanged during the migration process. As a result, we infer that VM live migration has a moderate impact on the performance of CPU and memory-intensive workloads on the migration source and destination PMs.

While the experiments above illustrate VM migration interference from the viewpoint of guest domains (*i.e.*, a black-box way), Fig. 3 takes a closer look into the VU of the *domain-0* on the migration source PM during the migration of a large VM instance running the mcf application (*i.e.*, a white-box way). We observe that the VU_m of the *domain-0* on the source PM remains nearly zero and the VU_c of the *domain-0* varies from 6%-25%. In sharp contrast, the VU_o varies from 31%-92% even with the rate limiting mechanism enabled in *Xen*. In addition, the VU_c and VU_m of the *domain-0* on the destination PM are almost the same as that on the source PM, and the VU_i of the *domain-0* can also vary from 31%-91%.

In summary, our findings above capture the migration interference in both black- and white-box manners, rather than treating it as a constant value of application performance degradation (*e.g.*, the increase in response time) from the viewpoint of guest domains [7], [31]. In essence, VM live migration can be treated as a network-intensive and CPU-moderate application in *domain-0*, and it interferes with the VMs running network or CPU-intensive workloads on both migration source and destination PMs. The rationale is that, *the aggregated*

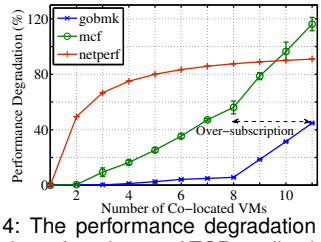


Fig. 4: The performance degradation of the gobmk, mcf, and netperf TCP applications, as the number of co-located VMs increases from 1 to 11, due to VM co-location interference.

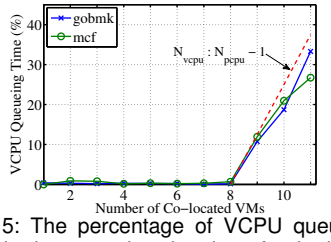


Fig. 5: The percentage of VCPU queueing time in the execution duration of gobmk and mcf applications, with different numbers of co-located VMs on the PM.

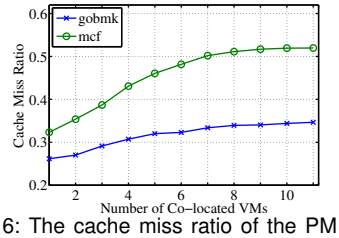


Fig. 6: The cache miss ratio of the PM with different numbers of co-located VMs hosting the gobmk and mcf applications.

network bandwidth and CPU computation "demand" of the domain-0 and other co-located VMs cannot be satisfied by the resource "supply" on the source and destination PMs.

In order to avoid the severe migration interference on the network I/O resource, the best practices for several commercial virtualization platforms (*e.g.*, VMware vSphere [32]) have recently recommended a dedicated network for live VM migration. To emulate such a cloud platform, we configure our experimental setup so that the migration traffic passes through the Ethernet interface for the NFS storage traffic, and re-run our experiments above. As expected, the migration interference on the network I/O resource is completely alleviated. However, the migration interference on the CPU resource and migrated VMs still exists, similar to the shared network scenario. We will consider the case with a dedicated migration network in the *iAware* model in Sec. 3.1.

2.2 VM Co-location Interference

Next, we measure and analyze VM co-location interference to answer two questions: (1) *What is the relationship between VM co-location interference and the number of co-located VMs?* (2) *What are the key factors that can reflect VM co-location interference?* In the next experiment, we use P_h to denote the PM that hosts multiple VMs, and N_{vm} to denote the number of co-located VMs on P_h. Specifically, we take the small VM instance as an example and vary N_{vm} from one to eleven. The VMs are running the gobmk, mcf and netperf workloads as in Sec. 2.1. To quantify VM co-location interference, we define a *performance degradation metric* as the ratio of the degraded performance, in terms of the increase of execution durations or the decrease of network throughput, to the original performance when the respective VMs are running alone on P_h.

Fig. 4 depicts the performance degradation with the gobmk, mcf and netperf TCP applications in co-located VMs, as N_{vm} increases from 1 to 11. We observe that VM co-location interference is highly correlated to N_{vm}. The more VMs P_h hosts, the more severe VM co-location interference becomes. The rationale is that, *the aggregated shared resource "demands" of co-located VMs increase dramatically as P_h hosts more VMs, which far exceeds the fixed amount of shared resource "supply" on P_h.* In more detail, (1) VM co-location interference under CPU and memory-intensive applications (*e.g.*, gobmk, mcf) is roughly *linear* to N_{vm}. In particular, the co-location interference with VM over-subscription ($N_{vm} > 8$) is more severe than that without VM over-subscription ($N_{vm} \leq 8$). (2) Moreover, the co-location interference of mcf applications is generally more severe than that of gobmk applications. (3) Finally, the co-location interference of network-intensive applications (*e.g.*, netperf) can be empirically quantified as $1-\frac{1}{N_{vm}}$. As a result, the simple factor N_{vm} can be used to estimate VM co-location interference among similar types of co-located workloads.

Essentially, observation (1) can be explained by the VCPU scheduling mechanisms of the *Xen* Credit Scheduler [26]. Specifically, to share the physical CPU resources, each PCPU maintains two local queues (*i.e., over* and *under*) of active VCPUs. The VCPUs with negative *credits* join the *over* queue, while the VCPUs with positive *credits* join the *under* queue. Each VM (VCPU) is allocated with an amount of *credits*. At every scheduling time, the VCPU at the head of the *under* queue is scheduled to run on the PCPU, which consumes *credits* of the scheduled VCPU. Accordingly, the more active VMs (VCPUs) the PM over-subscribes, the longer time VCPUs have to wait in the queue, which hence results in severe VM interference. As shown in Fig. 5, the *VCPU queueing time* is nearly zero when there is no over-subscription ($N_{vm} \leq 8$). When over-subscription occurs ($N_{vm} > 8$), the VCPU queueing time shows a roughly *linear* increase. In particular, the percentage of VCPU queueing time in the execution duration of applications is capped by $\frac{N_{vcpu}}{N_{pcpu}} - 1$, where N_{vcpu} and N_{pcpu} denote the number of VCPUs and PCPUs hosted on P_h, respectively.

Next, observation (2) can be explained by: the heavy contention on cache space and memory bandwidth will lead to severe co-location interference [15], [33]. Specifically, we use the cache miss ratio and physical memory utilization PU_m to estimate the cache and memory bandwidth contention on a PM, based on the rationale that the severe cache and memory bandwidth contention will be likely to cause more cache misses and higher memory utilization. As shown in Fig. 6, the cache miss ratios of the PM P_h for both the gobmk and mcf applications basically follow the trend of their respective performance degradation in Fig. 4 without VM over-subscription. In addition, our resource tracing tool reveals that the memory utilization PU_m of P_h is linear to N_{vm}, which can also explain the performance degradation of gobmk and mcf applications in Fig. 4.

Finally, observation (3) is explained by *Xen* I/O mech-

anisms [27]: the *Xen* backend driver in *domain-0* manages the I/O rings from the running VMs in a simple round-robin manner [10]. *Virtual hardware (i.e., network) interrupts* of VMs are sent to the backend driver through the event channel, and such events are processed in strict round-robin order in the backend driver to ensure fairness [27]. Accordingly, the N_{vm} co-located VMs running netperf applications on a PM roughly share the total network bandwidth B_m of the PM. Hence, the network throughput of each VM is around $\frac{B_m}{N_{vm}}$. Since the netperf application tends to make full use of B_m when it runs alone on the PM, the performance degradation of N_{vm} co-located VMs can be quantified as $(B_m - \frac{B_m}{N_{vm}})/B_m = 1 - \frac{1}{N_{vm}}$. In summary, the three explanations above inspire us to utilize not only VU and PU but also the cache miss ratio and VM network interrupts as the key factors, in order to reflect VM co-location interference, as we will elaborate in Sec. 3.

3 A Simple Estimation Model of VM Interference

Based on the empirical understanding above, *how can we effectively estimate both VM migration interference and VM co-location interference in a holistic manner?* As evidenced by Sec. 2, both VM migration and co-location interference can be essentially viewed as the mismatch between aggregated resources "demands" of co-located VMs (including the migrated VM and *domain-0*) and shared resources "supply" provided by the migration source and destination PMs. In response, we design a simple multi-resource demand-supply model [34] in *iAware*, which unifies the key factors that are previously identified in Sec. 2. The notations used in our estimation model are summarized in Table 3.

TABLE 3: Key notations in our estimation model.

Notation	Definition
υ_j	Estimated network I/O interference on a PM P_j
ρ_j	Estimated CPU resource contention on a PM P_j
t_i	Estimated migration time of a VM V_i
ω_j	Estimated migration interference on a PM P_j
γ_{id}	Estimated network I/O interference caused by a migrated VM V_i co-located on a PM P_d
θ_{id}	Estimated CPU resource contention caused by a migrated VM V_i co-located on a PM P_d
ξ_{id}	Estimated cache and memory bandwidth interference caused by a VM V_i co-located on a PM P_d
κ_x	Normalized parameter for a variable x with respect to its maximum values in one round of migration
M, N, T	Estimated VM migration interference, VM co-location interference, and the overall performance interference

3.1 Demand-Supply Model of VM Migration Interference

As demonstrated in Sec. 2.1, VM live migration can be viewed as a network-intensive and CPU-moderate application in *domain-0*, which implies that migrating VMs on the PMs with heavy I/O and CPU contention will cause severe migration interference. This inspires us to model VM migration interference as a function of *network I/O contention and CPU resource contention on both migration source and destination PMs*, which highly depends on *Xen* I/O mechanisms [27] and VCPU scheduling mechanisms [26], as we elaborated in Sec. 2.2.

We first estimate the network I/O interference among VMs based on *Xen* I/O mechanisms [27]. Specifically, we use the number of network interrupts inside a VM to reflect its network I/O "demand." As the virtual interrupts from all VMs on a PM are eventually handled by the backend driver in *domain-0*, the number of network interrupts in *domain-0* can be regarded as how many I/O "demands" from all VMs are "satisfied" by the hosting PM. Accordingly, we characterize the network I/O contention by using a notion of *demand-supply ratio* of the total number of network interrupts observed in VMs to that observed in *domain-0*. Intuitively, the higher such a demand-supply ratio is, the more severe the network I/O contention becomes. Furthermore, we observe that the network throughput of VMs tends to fluctuate wildly under severe network I/O contention. To capture such effects, we also take into account the variation coefficient of network throughput of VMs, which is defined as the ratio of the standard deviation to the mean of network throughput of VMs.

We assume that time t is slotted (*e.g.,* one second per slot). During a period of τ time slots, the network throughput of VM V_i is sampled as $H_i(t,\tau) = \{H_i(t-\tau), H_i(t-\tau+1), \ldots, H_i(t-1)\}$. Likewise, the number of network interrupts of V_i is sampled as $I_i(t,\tau) = \{I_i(t-\tau), I_i(t-\tau+1), \ldots, I_i(t-1)\}$. Then, we calculate the mean μ_{ih} and standard deviation σ_{ih} of $H_i(t,\tau)$, and the mean μ_{it} of $I_i(t,\tau)$ for each VM, including the mean μ_{1t} for the *domain-0* (*i.e., V_1*). Suppose we have α_j VMs hosted on the PM P_j including *domain-0*, we estimate their network I/O interference υ_j on the PM P_j as Eq. (1), which will be qualitatively validated in Sec. 5.1.

$$\upsilon_j = f_\upsilon(\sigma_{ih}, \mu_{ih}, \mu_{it}) \approx \sum_{i=2}^{\alpha_j} \frac{\sigma_{ih}}{\mu_{ih}} + \frac{\sum_{i=2}^{\alpha_j} \mu_{it}}{\mu_{1t}}. \quad (1)$$

We next estimate the CPU resource contention among VMs based on the VCPU scheduling mechanisms [26]. As demonstrated in Sec. 2.2, more active VMs (VCPUs) hosted by a PM will lead to heavier CPU resource contention, which can be characterized by another ratio of the total CPU "demands" of co-located VMs to the CPU "supply" of the hosting PM. Specifically, we calculate the sum of the CPU utilization and percentage of VCPU queueing time observed in a VM as its CPU "demand," and the number of PCPUs on the hosting PM as its CPU "supply." Accordingly, the CPU resource contention ρ_j on a PM P_j is estimated by

$$\rho_j = f_\rho(VU_{ic}, Q_i, N_i, N_j) \approx \frac{\sum_{i=1}^{\alpha_j} (VU_{ic} + Q_i) \cdot N_i}{N_j}, \quad (2)$$

where VU_{ic} and Q_i denote the CPU utilization and percentage of the VCPU queueing time observed in a VM V_i, respectively. N_i is the number of VCPUs hosted by V_i, and N_j is the number of PCPUs hosted by P_j.

By combining the two essential models above, we estimate the migration interference ω_j on co-located VMs hosted by a migration source or destination PM P_j as

$$\omega_j = f_\omega(\upsilon_j, \rho_j) \approx \kappa_\upsilon \cdot \upsilon_j + \kappa_\rho \cdot \rho_j, \quad (3)$$

where κ_υ and κ_ρ are the parameters that normalize the variables υ_j, ρ_j with respect to their respective maximum values obtained across all migration source and destination PMs during one round of VM migration. In particular, κ_υ is set to zero if the datacenter adopts a dedicated migration network, as discussed in Sec. 2.1. κ_ρ is set to zero if the datacenter statically pins the VCPU(s) of *domain-0* to non-shared PCPU(s). We will validate the effectiveness of the *iAware* model in a cloud platform with a dedicated migration network in Sec. 5.2.

In addition, to evaluate the migration interference on the migrated VMs, we also incorporate another key factor, the *VM migration time*, into our estimation model. Specifically, the major factors impacting VM migration time are the memory size m_i, memory dirtying rate D_i of the VM to be migrated V_i, and the available network bandwidth R_i [13] for V_i as well as the number of concurrent migrations c_i [8] with V_i. For simplicity, we approximately calculate the migration time t_i as

$$t_i = f_t(m_i, c_i, R_i, D_i) \approx \frac{m_i \cdot c_i}{R_i - D_i}. \quad (4)$$

In summary, the VM migration interference M on the migration source PM P_s and destination PM P_d as well as the migrated VM V_i can be characterized as

$$M = f_M(\omega_s, \omega_d, t_i) \approx \omega_s + \omega_d + \kappa_t \cdot t_i, \quad (5)$$

where ω_s and ω_d are the migration interference on P_s and P_d, respectively. Similar to κ_υ and κ_ρ in Eq. (3), the parameter κ_t normalizes the VM migration time t_i with respect to its maximum value of all possible choices of VMs to be migrated in one round of VM migration.

3.2 Demand-Supply Model of VM Co-location Interference

According to our key findings in Sec. 2.2, VM co-location interference is mainly caused by the contention of shared and constrained network bandwidth, CPU, as well as the cache and memory bandwidth, with the condition of non-overlapped memory allocated to each active VM. This guides us to model VM co-location interference as a function of *network I/O contention, CPU resource contention, cache and memory bandwidth contention on migration destination PMs*. We assume that the datacenter is provisioned with homogeneous or similar types of PMs [35].

First, we estimate the network I/O interference caused by a migrated VM V_i co-located on the destination PM P_d. Based on *Xen* I/O mechanisms [27], we can infer that if a VM with heavy network I/O "demand" is migrated to a destination PM with constrained bandwidth "supply" and severe network I/O contention (*i.e.,* υ_d), then the I/O throughput of other co-located VMs on the destination PM is more likely to deteriorate. Specifically, we use the mean value of network interrupts μ_{it} inside V_i as its network I/O "demand," the maximum network interrupts μ_{dt} that P_d can sustain per time slot as its network I/O "supply." The network I/O contention υ_d on P_d is estimated by Eq. (1). Accordingly, we estimate the network I/O interference γ_{id} by

$$\gamma_{id} = f_\gamma(\mu_{it}, \mu_{dt}, \upsilon_d) \approx \frac{\mu_{it}}{\mu_{dt}} + \upsilon_d. \quad (6)$$

We next estimate the CPU resource contention caused by a migrated VM V_i co-located on the destination PM P_d. The VCPU scheduling mechanisms [26] elaborated in Sec. 2.2 implies that a VM with a large amount of CPU computation "demand" migrated to a destination PM with scarce CPU resource "supply" will lead to heavy CPU resource contention. Similar to Eq. (6) above, we estimate the CPU resource contention θ_{id} as

$$\theta_{id} = f_\gamma(d_i, N_d, \rho_d) \approx \frac{d_i}{N_d} + \rho_d, \quad (7)$$

where d_i denotes the CPU resource "demand" of V_i. Both the number of PCPUs N_d and CPU resource contention before migration ρ_d denote the CPU resource "supply" of P_d. Specifically, as in Sec. 3.1, d_i is calculated as the CPU utilization plus the percentage of VCPU queueing time observed in V_i, and ρ_d can be calculated by Eq. (2).

We finally estimate the cache and network bandwidth interference caused by a migrated VM V_i co-located on the destination PM P_d. Guided by the analysis in Sec. 2.2, we infer that the contention on cache space and memory bandwidth tends to become severe, if a VM with large cache and memory bandwidth consumption is migrated to a destination PM with scarce remaining cache and memory bandwidth resource. Accordingly, we construct two demand-supply ratios of the cache space and memory bandwidth "demands" of V_i to the remaining cache space and memory bandwidth "supplies" on P_d, respectively, which are combined together to estimate the cache and memory bandwidth interference ξ_{id} as

$$\xi_{id} = f_\xi(PU_m, VU_m, s_{dp}, s_{iv}) \approx \frac{VU_{im}}{1 - PU_{dm}} + \frac{\beta \cdot s_{iv}}{1 - s_{dp}}, \quad (8)$$

where VU_{im} and PU_{dm} denote the memory utilization on V_i and P_d, respectively. s_{iv} and s_{dp} denote the cache miss ratio on V_i and P_d, respectively. In particular, $\beta \in (0, 1]$ is a scale factor of s_{iv} to estimate the "demand" of cache space consumption in V_i (*i.e.,* the cache miss ratio obtained in V_i running alone on a PM). We empirically set the factor β as one minus the cache miss ratio of the migration source PM.

Combining the three essential models above, VM co-location interference N caused by a migrated VM V_i on

the destination PM P_d is estimated as

$$N = f_N(\gamma_{id}, \theta_{id}, \xi_{id}) \approx \kappa_\gamma \cdot \gamma_{id} + \kappa_\theta \cdot \theta_{id} + \kappa_\xi \cdot \xi_{id}, \quad (9)$$

where κ_γ, κ_θ and κ_ξ are the parameters that normalize each variable γ_{id}, θ_{id}, ξ_{id} with respect to their respective maximum values, obtained across all possible choices of VMs to be migrated and their corresponding migration destination PMs in one round of VM migration. In particular, κ_θ is set to zero if the datacenter assigns each active VM with non-shared PCPU(s). We will qualitatively validate the model above in Sec. 5.1.

In summary, the overall VM performance interference T can be considered as a function of the estimated VM migration interference M in Eq. (5) and VM co-location interference N in Eq. (9), which is given by

$$T = f_T(M, N) \approx a \cdot (\kappa_M \cdot M) + b \cdot (\kappa_N \cdot N). \quad (10)$$

For numerical stability, the parameters κ_M and κ_N normalize interference values M, N with respect to their respective maximum values, obtained across all possible choices of VMs to be migrated and their corresponding migration destination PMs in one round of VM migration. In particular, a and b are the relative weights of VM migration interference versus VM co-location interference, respectively, and $a + b = 1$. By tuning the values of a and b, our *iAware* VM live migration strategy is flexible enough to selectively switch our design choices between different aspects of performance interference during and after VM migration, as demonstrated in Sec. 5.2.3.

Algorithm 1: *iAware*: interference-aware VM live migration strategy.

Input: (1) k potential candidate VMs for migration $\mathbb{V} = \{V_1, V_2, \cdots, V_k\}$, (2) l available PMs $\mathbb{P} = \{P_1, P_2, \cdots, P_l\}$.
Output: Selected VM to be migrated and migration destination PM.
Symbol: (1) M, N, T, ω_j, t_i, κ_x are listed in Table 3, (2) the subset of candidate VMs for migration $\mathbb{C}$.
Step *a*). Select candidate VM(s) with the least estimated migration interference on the migration source PM(s).
Initialize $\omega_{\min} \leftarrow \inf$, $\mathbb{C} \leftarrow \varnothing$;
for all VM $V_i \in \mathbb{V}$ **do**
  $\omega_s \leftarrow$ Eq. (3) for computing migration interference on the source PM hosting V_i;
  if $\omega_{\min} > \omega_s$ **then**
    $\omega_{\min} \leftarrow \omega_s$; $\mathbb{C} \leftarrow V_i$;
  else if $\omega_{\min} == \omega_s$ **then**
    $\mathbb{C} \leftarrow \mathbb{C} \cup \{V_i\}$;
  end if
end for
Step *b*). Identify a pair of VM to be migrated and destination PM by jointly minimizing VM migration and co-location interference.
Initialize $T_{\min} \leftarrow \inf$, $V_{\min} \leftarrow -1$, $P_{\min} \leftarrow -1$;
for all VM $V_i \in \mathbb{C}$ **do**
  for all PM $P_j \in \mathbb{P}$ capable of hosting V_i **do**
    $\omega_d \leftarrow$ Eq. (3) with (V_i, P_j) for computing migration interference on the destination PM;
    $M \leftarrow \omega_{\min} + \omega_d + \kappa_t \cdot t_i$ according to Eq. (5);
    $N \leftarrow$ Eq. (9) with (V_i, P_j) for computing co-location interference on the destination PM;
    $T \leftarrow a \cdot (\kappa_M \cdot M) + b \cdot (\kappa_N \cdot N)$ according to Eq. (10);
    if $T_{\min} > T$ **then**
      $T_{\min} \leftarrow T$; $V_{\min} \leftarrow i$; $P_{\min} \leftarrow j$;
    end if
  end for
end for
return $V_{\min}$ and $P_{\min}$ as the selected VM and PM indices in $\mathbb{V}$ and $\mathbb{P}$, respectively.

4 An Interference-Aware VM Migration Strategy

Based on both the empirical understanding and simple estimation model of VM performance interference in previous sections, we present *iAware* in Algorithm 1 for making decisions on *which candidate VMs are to be migrated to which PMs*, in order to jointly mitigate both the VM migration interference and co-location interference in a holistic manner. In particular, *when or whether to migrate VMs* can be periodically evaluated by load balancing, power saving or any other VM (or server) managements in datacenters, rather than by *iAware*.

When one or multiple VMs $\mathbb{V}$ need to be migrated for achieving load balancing or power saving in datacenters, *iAware* first selects a subset $\mathbb{C}$ of candidate VM(s) with the least estimated migration interference on the running VMs hosted by source PM(s). Then, *iAware* decides which PM(s) can serve as the potential migration destination for those candidate VM(s) in $\mathbb{C}$, by further estimating VM co-location interference. Specifically, for each pair of VM in $\mathbb{C}$ and available PM, *iAware* infers whether the PM can host the VM, according to the VM assignment constraint on the memory capacity [28]. Only capable PMs are the candidate destinations worthwhile for jointly calculating both the estimated migration interference and co-location interference, according to our simple multi-resource demand-supply model in Sec. 3. After iteratively electing the appropriate destination PM for each VM in $\mathbb{C}$, *iAware* can identify a pair of VM and PM with the minimum overall performance interference.

Given the detailed notations in Algorithm 1, the complexity of the *iAware* strategy is determined by the number of candidate VMs for migration $|\mathbb{C}|$ and the number of available PMs l. Suppose the total number of PMs in the datacenter is m, the worst-case for strategy computation is in the order of $\mathcal{O}(m \cdot k)$, when the total k VMs in $\mathbb{V}$ are selected as the candidate VMs for migration (*i.e.*, $\mathbb{C} = \mathbb{V}$), and all PMs in the datacenter are available for migration (*i.e.*, $l = m$). In practice, given the average number of VMs hosted on a PM N_{vm} in the datacenter, we have $m \approx \frac{k}{N_{vm}}$. Hence, the complexity of the *iAware* strategy can be further simplified as: (1) $\mathcal{O}(m \cdot k) \approx \mathcal{O}(k^2)$ in the scenario of a PM hosting less than tens of VMs on average, *e.g.*, $N_{vm} = 5$ without VM over-subscription, or (2) $\mathcal{O}(m \cdot k) \approx \mathcal{O}(k)$ in the scenario of a PM hosting hundreds of VMs on average, *e.g.*, $N_{vm} = 100$ by VM over-subscription. We will validate the computational complexity of *iAware* in Sec. 5.3.

The *iAware* strategy can be iterative if multiple rounds of VM migrations are required for reaching a given degree of load balancing or power saving χ. As we will demonstrate in Sec. 5.2, *iAware* is flexible and lightweight enough to be incorporated into existing VM migration

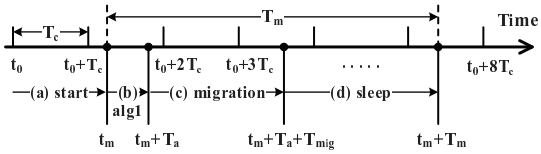


Fig. 7: Timing and operation process of *iAware*, where T_c and T_m are the communication interval and migration decision interval, respectively: (a) first, when or whether to migrate VMs is evaluated by VM (or server) managements in datacenters at t_m, (b) Algorithm 1 execution, which is enabled based on the resource utilization information of PMs and VMs collected in the last communication interval $[t_0, t_0 + T_c]$, takes a processing time of T_a to make the migration decision, (c) following the decision, the live migration is performed with a duration of T_{mig}, (d) then, Algorithm 1 sleeps for T_s until the next migration decision interval.

and consolidation policies (*e.g.,* [4], [7]) in a complementary manner, for mitigating extra performance loss in the cloud. To avoid excessive rounds of VM migrations, the iteration of the *iAware* strategy terminates when any of the following three simple conditions holds: (1) the total rounds of VM migrations exceed a threshold which is equal to the number of potential candidate VMs for migration k, (2) the degree of load balancing or power saving χ_i for i-th iteration is worse than χ_{i-1} for the previous iteration, (3) χ_i surpasses the target χ given by datacenter operators. The iteration-exit conditions of *iAware* will be validated in Sec. 5.2. Please also refer to the online supplementary material for detailed pseudocode and additional experimental results.

More Specifically, Fig. 7 illustrates the timing and operation process of the *iAware* strategy with two types of time intervals: (1) the aforementioned communication interval T_c for periodic information collection, and (2) the migration decision interval T_m for one round of the *iAware* strategy execution, which consists of the processing time T_a of Algorithm 1, migration time T_{mig} of the migrated VM and sleep time T_s of Algorithm 1 after the VM migration. To avoid frequent and noisy migrations, the migration decision interval, which can be flexibly determined by the system administrator, is relatively longer than the communication interval, such as ranging from minutes (*e.g.,* 5 minutes in VMware Distributed Resource Scheduler (DRS) [36]) to one hour.

5 Performance Evaluation

In this section, we carry out a series of experiments using mixed types of representative workloads in Table 1, to evaluate the effectiveness of our estimation model of VM performance interference and VM migration strategy of *iAware*. Specifically, we demonstrate the performance gain and runtime overhead of *iAware*, in comparison to both the First-Fit Decreasing (FFD) [7] and Sandpiper [4] algorithms. Furthermore, complementary large-scale simulations are conducted to validate the scalability of *iAware*.

We implement a prototype of *iAware* with $3,600$ lines of C code, based on our *Xen* virtualized cluster platform illustrated in Fig. 8, which includes: (1) Enterprise-class PMs (described in Sec. 2) that host running VMs. Inside

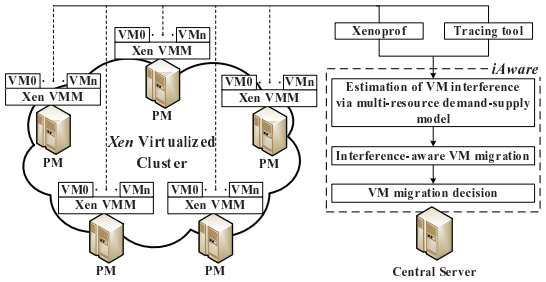


Fig. 8: Implementation of *iAware* upon a *Xen* virtualized cluster.

each VM (including *domain-0*), we use our aforementioned tracing tool together with *xenoprof* [30] to record VU, PU, the cache miss ratio. In addition, the number of network interrupts is measured using the Linux tool "vmstat", and VM memory dirtying rate can be obtained by tracking the shadow page tables in *Xen* as in [13]. (2) Central server(s) with *iAware* that periodically communicates with all active VMs to collect the information of multi-dimensional resource consumption in our *Xen* virtualized cluster. To reduce the communication and computation overhead of servers, we set the communication interval to 10 seconds, and simply input the average of resource utilization of PMs and VMs during the last communication interval in the *iAware* strategy. When one or multiple VMs need to be migrated for load balancing or power saving, *iAware* on the central server(s) will estimate VM performance interference through the simple multi-resource demand-supply model in Sec. 3, and apply the interference-aware strategy in Sec. 4 to make VM migration decisions accordingly.

5.1 Validating Estimation Models of VM Interference

We first validate the *iAware* model of network I/O interference in Sec. 3.1. To this end, we use *IxChariot console* version 5.4 and *endpoint* version 5.1 [37], which are the leading testing tools for real-world network applications, to generate network I/O demands in three co-located VMs on a PM for emulating different network I/O contention scenarios. Specifically, we statically set the network throughput of *vm01* and *vm02* to 200 Mbps and 400 Mbps, respectively, and gradually vary the network throughput of *vm03* from 100 Mbps to $1,000$ Mbps with the step of 100 Mbps. We record their actual I/O throughput and network interrupts for 10 seconds to estimate the network I/O interference υ_j by Eq. (1), and calculate the overall I/O throughput degradation of the three co-located VMs according to the *performance degradation metric* defined in Sec. 2.2. As shown in Fig. 9(a), the estimated network I/O interference follows the trend of actual I/O throughput degradation of the three VMs, which validates the effectiveness of our estimation model of network I/O interference.

Next, we validate the *iAware* model of co-location interference caused by the migrated VM at the destination PM in Sec. 3.2. In particular, we conduct experiments with two PMs (*i.e., pm01*, *pm02*), which act as the migration source and destination PMs, respectively.

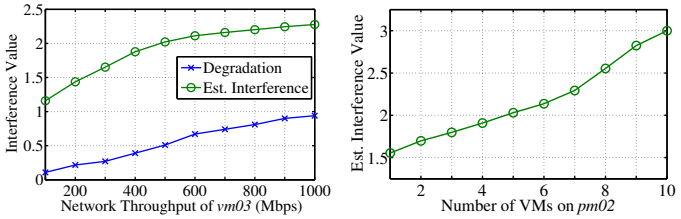


Fig. 9: Validation of the estimation model: (a) The estimated network I/O interference and actual I/O throughput degradation of the three VMs, as the network throughput of *vm03* varies from 100 Mbps to 1,000 Mbps. (b) The estimated co-location interference caused by *vm01* migrated from *pm01* to *pm02*, as the number of VMs on *pm02* varies from 1 to 10.

pm01 hosts the VM *vm01* to be migrated, and *pm02* hosts various numbers of VMs from 1 to 10. Each VM runs the mcf application of SPECCPU2006 [21]. We record PU, VU and cache statistics in PMs and VMs, and estimate VM co-location interference N caused by *vm01* migrated on *pm02* by Eq. (9). As the number of VMs on *pm02* increases in Fig. 9(b), we observe that migrating *vm01* on *pm01* to *pm02* would incur severe VM co-location interference, which captures the actual trend of performance degradation experienced by the mcf application in Fig. 4. Furthermore, the effectiveness of the *iAware* strategy using such qualitative estimations will be further demonstrated by Sec. 5.2.

5.2 Effectiveness and Overhead of *iAware*

We further examine the effectiveness and runtime overhead of *iAware* under mixed types of benchmark workloads in Table 1. Specifically, we use W_1, W_2, W_3, W_4, and W_5 to denote the VMs hosting SPECCPU2006 [21], netperf v2.5.0 [22], Hadoop v0.20.203.0 [23], NASA parallel benchmark (NPB) v3.3.1 [24], and SPECweb2005 [25], respectively. By assigning each type of workloads with a group of 10 VMs, we have 50 VMs in our *Xen* virtualized cluster in total. The VMs are configured as small or large VM instances as elaborated in Sec. 2. We use W_{xL} to denote a large VM instance hosting a workload x, and $W_{x\sim y}$ to denote a sequence of VMs from W_x to W_y, where $x, y \in \{1, 2, 3, 4, 5\}$ and $x < y$. Fig. 10 shows the initial placement of VMs and workloads across PMs, where the CPU resource of P_1, P_2, P_4, P_5 and P_6 is over-subscribed for hosting VMs.

In particular, the workload performance is measured as network throughput, application execution durations or the number of successfully served requests. Specifically, the performance results of SPECCPU and netperf are the average across the instances hosting them on the migration source and destination PMs, whereas Hadoop, NPB, and SPECweb retrieve their performance results from their respective "master" instances, which coordinate other "slave" instances to finish the assigned jobs. To make performance evaluation accurate, we run workloads 5 times when examining each VM migration strategy, and illustrate our performance results with error bars of standard deviations. We initially set the execution duration of workloads to 450 seconds.

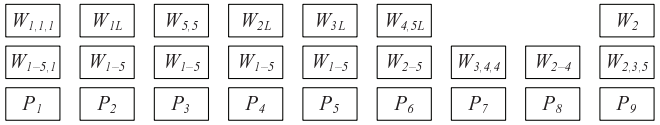


Fig. 10: Initial placement of VMs and workloads on each PM in our *Xen* virtualized cluster with CPU over-subscription.

5.2.1 Performance Comparison of Different Strategies

We first consider a general scenario for VM migration, by setting the VMs hosted on P_2 as the potential candidate VMs for migration and $\{P_1, P_3, P_4, \cdots, P_9\}$ as the available PMs. Fig. 11 illustrates the VM migration decisions made by the FFD [7] algorithm for power saving and the Sandpiper [4] algorithm for load balancing, compared to *iAware* in Algorithm 1. We observe that: (1) both FFD and Sandpiper select the large VM instance W_{1L} as the candidate VM to be migrated from P_2, while *iAware* selects a small VM instance W_3. (2) FFD and Sandpiper choose P_3 and P_7 as the migration destination PMs, respectively, while *iAware* chooses another PM P_8.

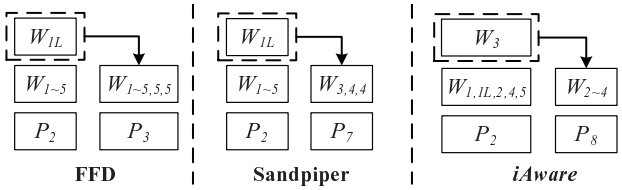


Fig. 11: VM migration decisions made by FFD, Sandpiper and *iAware* strategies.

The rationale is that FFD prefers to migrate the VM with the largest CPU and memory size to the first-fit PM, according to the VM assignment constraint on the memory capacity [28]. Sandpiper prefers to migrate the most loaded VM to the least loaded PM. In particular, the most loaded VM is actually the VM with the maximum volume-to-memory-size ratio [4], where the volume of a VM is calculated with its virtual resource utilization as $\frac{1}{(1-VU_c)\cdot(1-VU_m)\cdot(1-VU_n)}$. Our tracing tool reveals that W_{1L} consumes almost 100% of CPU usage and 90% of memory, and hence it is selected by Sandpiper to be migrated to P_7, which is the least loaded PM according to $\frac{1}{(1-PU_c)\cdot(1-PU_m)\cdot(1-PU_n)}$ [4]. In contrast, by estimating that W_3 has the least estimated VM migration interference on the source PM, *iAware* chooses it to be migrated to P_8 with the least estimated VM co-location interference, so as to jointly minimize VM performance interference both during and after VM migration.

Fig. 12(a) compares the normalized performance of representative benchmark workloads with FFD, Sandpiper and *iAware* strategies in one round of VM migration. We observe that *iAware* is able to improve the performance by around 45%-65% for network-intensive workloads (*i.e.*, netperf), and 16%-28% for workloads that are CPU, memory and network-intensive (*i.e.*, SPECCPU, Hadoop, NPB and SPECweb), as compared to FFD and Sandpiper. To look into the performance gain obtained by *iAware*, we further examine the performance of VMs over time during and after VM migration in Fig. 12(b) and Fig. 12(c). We observe that both FFD and

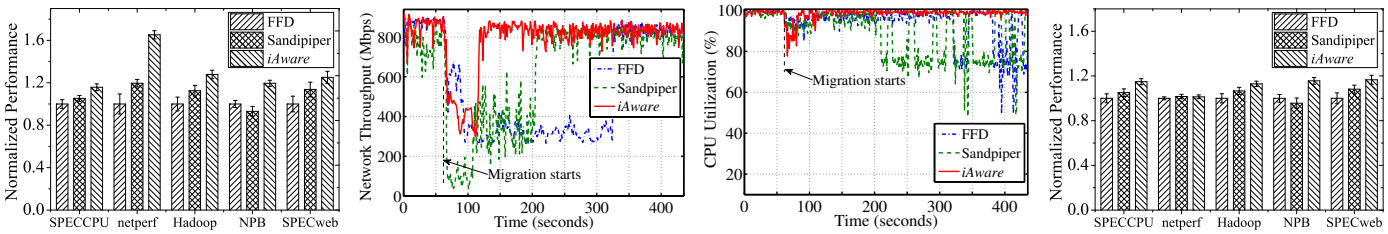


Fig. 12: Workload performance comparison with FFD, Sandpiper and *iAware* VM migration strategies. We show the aggregated performance (normalized by using FFD as a baseline) in a cloud platform with (a) a shared network and (d) a dedicated migration network. We also show the performance of VMs over time on both the migration source and destination PMs, in terms of (b) network throughput (*i.e.*, netperf) and (c) CPU utilization (*i.e.*, NPB) during and after VM migration.

Sandpiper severely degrade the network throughput and CPU utilization of VMs compared to *iAware*.

The rationale is that, the VMs for migration selected by FFD and Sandpiper (*e.g.*, W_{1L}) have a large VM size and a high memory dirtying rate. Moreover, the migration destination PMs selected by FFD and Sandpiper (*i.e.*, P_3, P_7) have constrained CPU, memory and network resources. Such migration decisions would incur more severe CPU, memory and network I/O interference on both the source and destination PMs, and thus affect VM performance for a longer duration than *iAware*. In contrast, *iAware* chooses to migrate W_3 on P_2 to P_8, which leads to the least CPU, network I/O and memory bandwidth contention during and after VM migration.

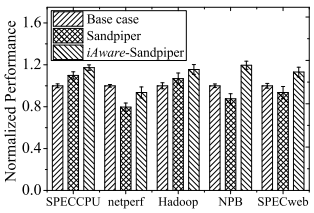
In addition, we examine the effectiveness of *iAware* in the cloud platform with a dedicated migration network. Both FFD and Sandpiper make the same migration decisions as in the cloud platform with a shared network, since they do not distinguish these two platforms. Using the migration interference model Eq. (3) in Sec. 3.1, *iAware* is adaptive to the cloud platforms with and without a dedicated migration network. As explained in Sec. 2.1, the dedicate migration network could eliminate the migration interference on the network I/O resource. In such a scenario, it would cause the least performance interference for *iAware* to migrate W_3 (*i.e.*, a CPU-moderate workload) on P_2 to P_9, which has constrained network resources and a sufficient amount of CPU resources. Fig. 12(d) shows the normalized workload performance in a cloud platform with a dedicated migration network. We observe that the performance gain for netperf application is almost eliminated, compared to that in the platform with a shared network shown in Fig. 12(a). Such an observation is consistent with our analysis in Sec. 2.1. In contrast, the performance gain for the other workloads still exists, as migration interference on the CPU resource and co-location interference are both minimized by the *iAware* strategy.

5.2.2 Strategy Cooperation for Load Balancing and Power Saving

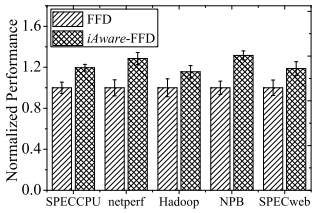
Moreover, we examine whether *iAware* can cooperate with existing VM migration policies in a complementary manner. We first conduct an experimental case study for achieving load balancing by incorporating *iAware* into Sandpiper [4] (denoted as *iAware*-Sandpiper). To allow multiple VM live migrations, we extend the running durations of workloads to 1050 seconds. Given that one migration process lasts for $60 - 200$ seconds as shown in Sec. 2.1 and the sleep time of *iAware* after migration is empirically set as 60 seconds in our platform, the running time of 1050 seconds can support up to $4 - 8$ (*i.e.*, $\lfloor \frac{1050}{200+60} \rfloor = 4, \lfloor \frac{1050}{60+60} \rfloor = 8$) live migrations, which meets our requirement to examine the effectiveness of *iAware* in load balancing and power saving scenarios. Essentially, our experiment results can be generalized to the other scenarios for VM migration in datacenters, such as hardware update and server maintenance.

We employ the hotspot detection strategy from Sandpiper [4] to trigger VM migration, according to the setting of CPU usage threshold (*i.e.*, 75%). Under the initial VM placement in Fig. 10, the CPU consumption on P_1 and P_2 exceeds the CPU usage threshold, and thus one or multiple VMs on P_1 and P_2 need to be migrated for load balancing on CPU usage in our cluster. We choose those VMs with their PCPU utilization greater than 75% on P_1 and P_2 as the set of potential candidate VMs for migration $\mathbb{V}$. Accordingly, the original Sandpiper chooses to migrate W_{1L} on P_2 to P_7, and W_1 on P_1 to P_8, respectively, while *iAware*-Sandpiper chooses to migrate W_4 on P_2 to P_8, and W_4 on P_1 to P_9, sequentially. Hence, the total rounds of migration under both *iAware*-Sandpiper and the original Sandpiper are 2, which shows that the iteration rounds of our strategy can be well-controlled by the exit conditions as elaborated in Sec. 4.

Fig. 13(a) compares the normalized workload performance with the original Sandpiper, *iAware*-Sandpiper strategies and the base case without VM migration. We observe that *iAware*-Sandpiper outperforms the original Sandpiper by 7%-32%. However, the performance of netperf application under Sandpiper and *iAware*-Sandpiper becomes worse than that in the base case. The reason is that the network-intensive workload (*i.e.*, netperf) is significantly affected by VM live migration process, as we have analyzed in Sec. 2.1. The performance of NPB workload under *iAware*-Sandpiper is much better than that with the original Sandpiper and the base case, as *iAware*-Sandpiper migrates the two VMs W_4 (*i.e.*, NPB) to the least loaded PMs P_8 and P_9, respectively, which incurs the least contention on CPU, cache and memory bandwidth resources. After these VM migrations have been performed, the standard deviations of CPU



(a)



(b)

Fig. 13: Workload performance comparison: (a) under Sandpiper and *iAware*-Sandpiper VM migration strategies, for achieving load balancing on CPU usage in the datacenter; (b) with FFD and *iAware*-FFD VM consolidation strategies, for achieving power saving in the datacenter.

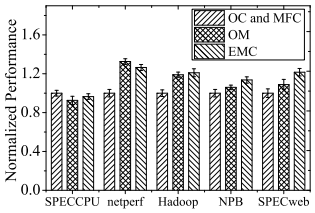


Fig. 14: Performance comparison of representative benchmark workloads using the *iAware* VM migration strategy with various design choices, *i.e.,* OM, EMC, OC and MFC.

utilization of all PMs in our cluster under Sandpiper and *iAware*-Sandpiper have become 0.1634 and 0.1422, respectively. This implies that *iAware*-Sandpiper can balance the CPU utilization of all PMs across the cluster as well as the original Sandpiper, while mitigating an additional performance loss during and after VM migration.

Furthermore, *iAware* is also able to cooperate with existing VM consolidation policies for achieving power saving, such as the classic FFD [7] algorithm. Specifically, we incorporate *iAware* into FFD (denoted as *iAware*-FFD) and compare the workload performance with the original FFD and *iAware*-FFD. In particular, this power saving scenario can be viewed as a special case of *iAware*, as all the VMs hosted on under-utilized PMs should be migrated. Then, *iAware*-FFD does not need to choose the cost-effective VMs for migration. It only needs to choose the cost-effective destination PMs among available PMs.

Under the initial VM placement in Fig. 10, both FFD and *iAware*-FFD choose to migrate the VMs on P_7 to other PMs, as P_7 is the least loaded PM in the cluster according to $\frac{1}{(1-PU_c)\cdot(1-PU_m)\cdot(1-PU_n)}$ [4]. Suppose that all PMs except the source PM P_7 could serve as the candidate destination PMs under FFD and *iAware*-FFD. *iAware*-FFD migrates W_3 to P_8, W_4 to P_9 and another W_4 to P_8, sequentially, with the least estimated performance interference. In contrast, FFD migrates W_4 to P_1, another W_4 and W_3 to P_3, sequentially, as P_1 and P_3 are the first-fit PMs for these VMs to be migrated. The total rounds of migration under both FFD and *iAware*-FFD are 3 as the three VMs on P_7 need to be migrated for power saving.

Fig. 13(b) compares the normalized workload performance with FFD and *iAware*-FFD strategies. We observe that all the workloads achieve much better performance under *iAware*-FFD than that with the original FFD by 15%-30%. This is because FFD is oblivious to VM performance interference and greedily consolidates VMs on a PM, which makes P_1 and P_3 over-subscribe more VMs. Accordingly, the workloads on the two PMs will take more time to complete. In particular, the performance of NPB and netperf applications is much better than the other workloads. The rationale is that, FFD migrates two VMs of W_4 (*i.e.,* NPB) to the most loaded PMs P_1 and P_3, respectively, which would incur severe VM performance interference on CPU and network I/O resources. In contrast, *iAware*-FFD seeks to alleviate such performance loss during VM consolidation by selecting PMs with the least estimated performance interference for W_3 and W_4.

To examine the *power efficiency* of the cluster handled by FFD and *iAware*-FFD VM consolidation strategies, we estimate the power consumption of clusters as a roughly linear function of the average CPU utilization u_t of PMs, which is empirically verified by the realistic statistics in Google datacenters [38]. Specifically, the power function is given by $P(u_t) = F(n) + n \cdot (P_{peak} - P_{idle}) \cdot (2u_t - u_t^r)$, where n is the number of PMs in a datacenter; the factor r is an empirical constant equal to 1.4; F_n, P_{peak} and P_{idle} are constant values. As captured by our resource tracing tool, the average CPU utilization of PMs after VM migrations with FFD and *iAware*-FFD is 0.5116 and 0.5456, respectively. Accordingly, the variable part $(2u_t - u_t^r)$ of the power consumption $P(u_t)$ with FFD and *iAware*-FFD is 0.6319 and 0.663, respectively, which implies that the cluster power consumption under *iAware*-FFD tends to be slightly higher than that under FFD. As a result, *iAware*-FFD can achieve better workload performance with an acceptable cost of power consumption, as compared to greedy VM consolidation strategies.

5.2.3 Flexible Design Choices of iAware

In addition, *iAware* can provide *flexible design choices* for cloud service providers by fine tuning the relative weights of VM migration interference versus VM co-location interference, denoted as a and b in Eq. (10) of Sec. 3. By treating the VMs on P_2 as the potential candidate VMs for migration, we examine the migration decisions and workload performance under the *iAware* strategy with four representative design choices: (1) EMC: $a = 0.5, b = 0.5$, equally considering VM migration and co-location interference; (2) OM: $a = 1, b = 0$, only considering VM migration interference; (3) OC: $a = 0, b = 1$, only considering VM co-location interference; (4) MFC: $a = 0.3, b = 0.7$, jointly considering VM migration and co-location interference while putting an emphasis on the latter. We observe that *iAware* migrates W_3 on P_2 to P_8 and W_5 on P_2 to P_7 under EMC and OM configurations, respectively. For both OC and MFC configurations, *iAware* migrates W_4 on P_2 to P_9.

Fig. 14 shows that different choices of a and b are suitable for different types of workloads: (1) OC and MFC configurations achieve slightly better performance of CPU-intensive workloads (*i.e.,* SPECCPU) than OM and EMC configurations. (2) The performance of network-intensive workloads (*i.e.,* netperf) under the OM configuration outperforms that under the EMC, OC and MFC

configurations by 6%-32%. (3) EMC configuration improves the performance of workloads that are both CPU and network-intensive (*i.e.*, Hadoop, NPB, SPECWeb) by 3%-13% compared with OM, OC and MFC configurations. These design implications can be adopted by cloud service providers to adjust the *iAware* strategy to adapt to various types of application workloads on demand.

5.2.4 Overhead of iAware

Finally, we evaluate the runtime overhead of *iAware* in terms of CPU and network I/O consumption on both the central server(s) and VMs in our *Xen* virtualized cluster. First, we periodically record the total amount of network communication traffic on the central server(s), which is around 35600 bytes every 10 seconds. This is marginal for a 1 Gbps network switch in a datacenter. Second, we examine the CPU usage of our prototype, incurred by the *iAware* strategy in Algorithm 1 and our resource tracing tool as elaborated in Sec. 5. Specifically, we compare the performance of benchmark workloads with and without our *iAware* prototype which disables the VM migration function. We find that the benchmark scores of CPU-intensive workloads (*i.e.*, mcf of SPEC-CPU2006 [21]) are affected by around one second due to the CPU overhead of *iAware*. These collectively show that *iAware* incurs marginal CPU and network I/O overhead for strategy computation and information collection of multi-dimensional resource utilization of PMs and VMs.

5.3 Validating Scalability of *iAware*

To examine the scalability of *iAware* and obtain complementary insights, we also conduct trace-driven simulations using the realistic WorldCup98 Web trace [39]. By replaying $50,000$ seconds of such trace starting from 1998/06/01 22:00:01 to 1998/06/02 11:53:21 on a small VM instance hosting the RUBiS Web server [40], we record the resource utilization per second. Then, we extract each continuous 10 seconds of such measured resource utilization to represent the resource consumption for different VMs. With these input into the central server of our *iAware* prototype, we increase the scale of the datacenter by: (1) setting the average number of VMs hosted on a PM (*i.e.*, N_{vm}) to 5 or 16 and varying the number of PMs from 7 to $1,000$, (2) setting N_{vm} to 100 and varying the number of PMs from 5 to 50. Thus, we not only check the running time of the *iAware* strategy by varying the number of potential candidate VMs for migration from 10 to $5,000$, but also sample the memory consumption of the *iAware* strategy by varying the total number of VMs in the datacenter from 10 to $5,000$.

As shown in Fig. 15(a), the running time of *iAware* increases quadratically as the number of potential candidate VMs for migration increases when $N_{vm} = 5$ or 16, while the running time of *iAware* is linear to the number of potential candidate VMs for migration when N_{vm} increases to 100, which is consistent with our complexity analysis in Sec. 4. Specifically, *iAware* can be finished within 1 second when the datacenter scales up to $2,500$ VMs. Fig. 15(b) shows that the memory consumption of the *iAware* strategy in the three scenarios are almost linear to the scale of the datacenter, as *iAware* requires additional memory to store the collected information of resource utilization of PMs and VMs for its computation. In more detail, the *iAware* strategy only consumes around 3 MB memory at the central server, even when the datacenter scales up to $5,000$ VMs.

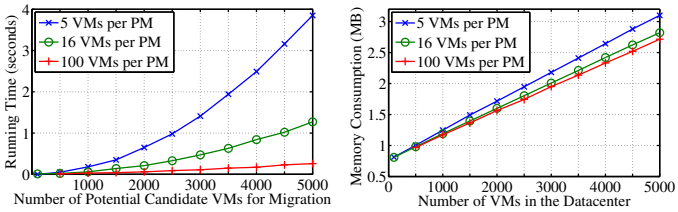


Fig. 15: Scalability of the *iAware* strategy in Algorithm 1: (a) running time of the *iAware* strategy under different numbers of potential candidate VMs for migration, and (b) memory consumption of the *iAware* strategy under different numbers of VMs in the datacenter, where the number of VMs varies from 100 to $5,000$ in the scenarios from 5 VMs per PM to 100 VMs per PM, respectively.

6 Related Work

6.1 VM Migration Interference

There have been works on measuring and studying VM migration interference in CPU and memory-intensive scenarios. For instance, [41] quantified VM migration interference as a linear function of the total amount of memory allocated to the VMs to be migrated. [31] conducted offline measurements of VM migration interference for representative multi-tier Web applications. Complementary to such works, our study comprehensively measures and analyzes VM migration interference under CPU, cache and memory, and network-intensive scenarios, with and without a dedicated migration network (Sec. 2.1). In particular, we carry out an in-depth examination of the multi-dimensional resource consumption of VM migration in *domain-0*, which has been seldom studied in existing works. Based on the measurements, our multi-resource demand-supply model identifies a set of fine-grained factors associated with the CPU, memory, network I/O contention and migration time (Sec. 3.1), which are more specific and complete than the existing resource contention model [8]. Moreover, the *iAware* model estimates VM migration interference online, while prior models (*e.g.*, [8]) mostly work offline and require pre-runs of application workloads and VM migration.

6.2 VM Co-location Interference

There have been several studies on VM co-location interference by VM resource contention. For instance, [5] illustrated the severity of co-location interference among three VMs. [42] examined the relationship between the co-location interference of two VMs and their resource utilization. A recent study [6] quantified the co-location interference of two VMs by the correlation of their

resource utilization, for the consolidation of scientific workloads. While such existing works analyzed VM co-location interference at a relatively small scale, *i.e.,* 2-3 VMs, our work further measures the co-location interference among multiple co-located VMs and utilizes critical factors such as the network interrupts inside VMs, VCPU queueing time, and cache miss ratios in both PMs and VMs (Sec. 2.2), in order to achieve the simple yet effective estimation of VM co-location interference.

There were also works on guaranteeing performance isolation across co-located VMs on shared resources. For example, [33] alleviated cache and memory bandwidth interference by compensating the impacted VMs for reserved CPU resource. [43] summarized the VM isolation techniques on the network resource, such as installing multiple network adapters, implementing static bandwidth allocation for VMs in the *domain-0* [28], or in network adapters and switches. Different from these works, *iAware* alleviates the shared memory and network bandwidth interference with optimized placement of VMs, rather than relying on resource compensating and bandwidth capping techniques, which either require reserved CPU (or network adapter) resources or adapt poorly to a dynamic number of VMs in datacenters [43]. Another approach [11] restricted CPU utilization of some victim VMs in *domain-0* to guarantee the network performance of other co-located VMs. Commercial virtualization solution of Microsoft allocated the network bandwidth among a dynamic number of VMs, by incorporating a rate controller into *Hyper-V* [16]. Orthogonal to these VM performance isolation solutions, we focus on mitigating the performance impact during and after VM migration.

6.3 VM Migration and Consolidation Strategies

There exist a number of VM migration and consolidation strategies for load balancing [4] and power saving [5]–[7]. Yet, few of them devoted adequate attention to the overall VM performance interference across migration source and destination PMs, during and after the migration of VMs. They either considered VM migration interference [8], [41] or co-location interference [5], [6], [12] in an isolated manner (as discussed in Sec. 6.1 and Sec. 6.2). In contrast, the *iAware* strategy seeks to mitigate overall VM performance interference by jointly measuring, analyzing and minimizing the migration and co-location interference during and after VM migration.

A latest work on VMware DRS [36] also considered the VM migration cost on the migration source and destination PMs, and the resource contention cost. However, it did not explicitly present a model on such performance cost. Our work further differs from DRS in that: (1) DRS only considers the migration cost on CPU and memory resources. (2) DRS makes a certain migration decision as long as its performance benefit exceeds its cost, while *iAware* identifies the migration decision with the minimum estimated overall performance interference.

7 Conclusion

In this paper, we have presented our design and implementation of *iAware*, a lightweight *interference-aware* VM live migration strategy to avoid violations of performance SLAs in the cloud. *iAware* jointly measures, analyzes and mitigates both VM migration and co-location interference in a holistic manner on the source and destination servers during and after VM migration. Based on a *Xen* virtualized cluster platform, we show extensive experiments of representative benchmark workloads, conducted to practically reveal the essential relationships between VM performance interference and key factors from servers and VMs. Such an empirical understanding further guides us to develop a simple multi-resource demand-supply model to estimate and minimize both VM migration and co-location interference. Extensive experiments and complementary large-scale simulations have demonstrated that *iAware* can qualitatively estimate VM performance interference, and improve I/O and network throughput and execution durations, by 65% for network-intensive workloads and by 16%-28% for workloads that are CPU, memory and network-intensive, as compared to the FFD and Sandpiper algorithms. In addition, the runtime overhead of *iAware* in terms of CPU consumption and network I/O is very well contained, even as it scales to thousands of VMs. Finally, we advocate that *iAware* cooperates well with existing VM migration or consolidation policies in a complementary manner, so that load balancing or power efficiency can be achieved without sacrificing performance.

Acknowledgments

The corresponding author is Fangming Liu. The research was supported in part by a grant from NSFC under grant No.61370232, by a grant from National Basic Research Program (973 program) under Grant of 2014CB347800. Prof. Bo Li's work was supported in part by a grant from RGC under the contract 615613, a grant from NSFC/RGC under the contract N_HKUST610/11, and a grant from ChinaCache Int. Corp. under the contract CCNT12EG01.

References

[1] J. Guo, F. Liu, H. Tang, Y. Lian, H. Jin, and J. C. Lui, "Falloc: Fair Network Bandwidth Allocation in IaaS Datacenters Via a Bargaining Game Approach," in *Proc. of IEEE ICNP*, 2013.

[2] Amazon Elastic Compute Cloud (Amazon EC2). [Online]. Available: http://aws.amazon.com/ec2/

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proc. of NSDI*, May 2005.

[4] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," in *Proc. of NSDI*, Apr. 2007.

[5] 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," in *Proc. of HotCloud*, Jun. 2011.

[6] Q. Zhu, J. Zhu, and G. Agrawal, "Power-aware Consolidation of Scientific Workflows in Virtualized Environments," in *Proc. of SC*, Nov. 2010.

[7] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Proc. of Middleware*, Dec. 2008.

[8] S.-H. Lim, J.-S. Huh, Y. Kim, and C. R. Das, "Migration, Assignment, and Scheduling of Jobs in Virtualized Environment," in *Proc. of HotCloud*, Jun. 2011.
[9] A. Koto, H. Yamada, K. Ohmura, and K. Kono, "Towards Unobtrusive VM Live Migration for Cloud Computing Platforms," in *Proc. of APSys*, Jul. 2012.
[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. of SOSP*, Oct. 2003.
[11] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in Xen," in *Proc. of Middleware*, Nov. 2006.
[12] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," in *Proc. of ATC*, Jun. 2013.
[13] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and Energy Modeling for Live Migration of Virtual Machines," in *Proc. of HPDC*, Jun. 2011.
[14] D. Breitgand, G. Kutiel, and D. Raz, "Cost-Aware Live Migration of Services in the Cloud," in *Proc. of HotICE*, Mar. 2011.
[15] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Virtual Machine," in *Proc. of SOCC*, Oct. 2011.
[16] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *Proc. of NSDI*, Mar. 2011.
[17] Z. Zhou, F. Liu, Y. Xu, R. Zou, H. Xu, C. Lui, and H. Jin, "Carbon-aware Load Balancing for Geo-distributed Cloud Services," in *Proc. of IEEE MASCOTS*, 2013.
[18] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On Arbitrating the Power-Performance Tradeoff in SaaS Clouds," in *Proc. of IEEE INFOCOM*, 2013.
[19] W. Deng, F. Liu, H. Jin, and C. Wu, "SmartDPSS: Cost-Minimizing Multi-source Power Supply for Datacenters with Arbitrary Demand," in *Proc. of IEEE ICDCS*, 2013.
[20] W. Deng, F. Liu, H. Jin, B. Li, and D. Li, "Harnessing Renewable Energy in Cloud Datacenters: Opportunities and Challenges," *IEEE Network Magazine*, 2013.
[21] SPEC CPU2006. [Online]. Available: http://www.spec.org/cpu2006/
[22] Netperf. [Online]. Available: http://www.netperf.org/
[23] Hadoop. [Online]. Available: http://hadoop.apache.org/
[24] NASA Parallel Benchmark. [Online]. Available: http://www.nas.nasa.gov/Resources/Software/npb.html
[25] SPEC Web2005. [Online]. Available: http://www.spec.org/web2005/
[26] Credit Scheduler. [Online]. Available: http://wiki.xen.org/wiki/Credit_Scheduler
[27] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," in *Proc. of VEE*, Mar. 2008.
[28] S. K. Barker and P. Shenoy, "Empirical Evaluation of Latency-sensitive Application Performance in the Cloud," in *Proc. of MMSys*, Feb. 2010.
[29] Oprofile. [Online]. Available: http://oprofile.sourceforge.net/
[30] Xenprof. [Online]. Available: http://xenoprof.sourceforge.net/
[31] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "A Cost-Sensitive Adaptation Engine for Server Consolidation of Multitier Applications," in *Proc. of Middleware*, Dec. 2009.
[32] "Vmware vsphere vmotion architecture, performance and best practices in vmware vsphere 5," White Paper, VMware, Aug. 2011.
[33] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds," in *Proc. of Eurosys*, Apr. 2010.
[34] F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li, "Novasky: Cinematic-Quality VoD in a P2P Storage Cloud," in *Proc. of INFOCOM*, Apr. 2011.
[35] J. Guo, F. Liu, D. Zeng, J. C. Lui, and H. Jin, "A Cooperative Game Based Allocation for Sharing Data Center Networks," in *Proc. of IEEE INFOCOM*, 2013.
[36] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware Distributed Resource Management: Design, Implementation, and Lessons Learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.
[37] IxChariot. [Online]. Available: http://www.netiq.com/support/
[38] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," in *Proc. of SIGCOMM*, Aug. 2009.
[39] WorldCup98. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/WorldCup.html
[40] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, "Performance Comparison of Middleware Architectures for Generating Dynamic Web Content," in *Proc. of Middleware*, Jun. 2003.
[41] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems," in *Proc. of SOCC*, Oct. 2011.
[42] R. C. Chiang and H. H. Huang, "TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments," in *Proc. of SC*, Nov. 2011.
[43] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance Isolation for Cloud Datacenter Networks," in *Proc. of HotCloud*, Jun. 2010.

Fei Xu is a Ph.D candidate in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His current research interests focus on cloud computing and virtualization technology.

Fangming Liu is an Associate Professor in Huazhong University of Science and Technology, and he is awarded as the CHUTIAN Scholar of Hubei Province, China. He is the Youth Scientist of National 973 Basic Research Program Project of "Software-defined Networking (SDN)-based Cloud Datacenter Networks", which is one of the largest SDN projects in China. Since 2012, he has also been a StarTrack Visiting Faculty in Microsoft Research Asia. He received his Ph.D. degree in Computer Science and Engineering from Hong Kong University of Science and Technology in 2011. His research interests include cloud computing and datacenter networking, mobile cloud, green computing, SDN and virtualization technology. He has been a Guest Editor for IEEE Network Magazine, an Associate Editor for Frontiers of Computer Science, and served as TPC for IEEE INFOCOM 2013-2014.

Linghui Liu is currently a Master student in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests focus on cloud computing and virtualization technology.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering in Huazhong University of Science and Technology (HUST), Wuhan, China. He is now Dean of the School of Computer Science and Technology at HUST. He received his Ph.D. degree in Computer Engineering from HUST in 1994. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. His research interests include computer architecture, virtualization technology, cluster computing and grid computing.

Bo Li is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His current research interests include: large-scale content distribution in the Internet, datacenter networking, cloud computing, and wireless communications. He has been an editor or a guest editor for a dozen of IEEE journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004. He received his Ph.D. degree in the Electrical and Computer Engineering from University of Massachusetts at Amherst. He is a Fellow of IEEE.

Baochun Li is a Professor with the Department of Electrical and Computer Engineering at the University of Toronto, Canada. He received the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks.