Colocation as a Service
Strategic and Operational Services for Cloud Colocation[†]

VATCHE ISHAKIAN
visahak@cs.bu.edu
Computer Science Dept
Boston University, USA

RAYMOND SWEHA
remos@cs.bu.edu
Computer Science Dept
Boston University, USA

JORGE LONDOÑO
jmlon@cs.bu.edu
Computer Science Dept
Boston University, USA

AZER BESTAVROS
best@cs.bu.edu
Computer Science Dept
Boston University, USA

***Abstract*—By colocating with other tenants of an Infrastructure as a Service (IaaS) offering, IaaS users could reap significant cost savings by judiciously sharing their use of the fixed-size instances offered by IaaS providers. This paper presents the blueprints of a Colocation as a Service (CaaS) framework. CaaS strategic services identify coalitions of self-interested users that would benefit from colocation on shared instances. CaaS operational services provide the information necessary for, and carry out the reconfigurations mandated by strategic services. CaaS could be incorporated into an IaaS offering by providers; it could be implemented as a value-added proposition by IaaS resellers; or it could be directly leveraged in a peer-to-peer fashion by IaaS users. To establish the practicality of such offerings, this paper presents XCS – a prototype implementation of CaaS on top of the Xen hypervisor. XCS makes specific choices with respect to the various elements of the CaaS framework: it implements strategic services based on a game-theoretic formulation of colocation; it features novel concurrent migration heuristics which are shown to be efficient; and it offers monitoring and accounting services at both the hypervisor and VM layers. Extensive experimental results obtained by running PlanetLab trace-driven workloads on the XCS prototype confirm the premise of CaaS – by demonstrating the efficiency and scalability of XCS, and by quantifying the potential cost savings accrued through the use of XCS.**

I. INTRODUCTION

Motivation: *Cloud computing* in general and Infrastructure as a Service (IaaS) in particular have emerged as compelling paradigms for the deployment of distributed applications and services on the Internet due in large to the maturity and wide adoption of virtualization technologies. By relying on virtualized resources, users are able to easily deploy, scale up or down their applications seamlessly across computing resources offered by one or more infrastructure providers. More importantly, virtualization enables performance isolation, whereby each application is able to acquire appropriate fractions of shared fixed-capacity resources for unencumbered use subject to binding Service-Level Agreements (SLAs).

The value proposition of IaaS offerings [1], [2] is highly dependent on the efficient utilization of cloud resources [3]. For an IaaS provider, this necessitates a judicious mapping of physical resources to virtualized instances that could be acquired over prescribed, fixed periods (*e.g.*, daily or hourly). To be flexible, an IaaS provider must be able to offer a range of such instances so as to cater to a wide range of customer needs, spelled out as SLAs defined over the various resources of the instance (*e.g.*, CPU, memory, local storage, network bandwidth). To be practical, an IaaS provider must limit the set of instance choices available to customers. For example, as of February 2010, Amazon EC-2 offers seven instance types: three types of standard instances, two types of high-memory instances, and two types of high-CPU instances [1].

While varied, the range of instance types available to IaaS customers is unlikely to match their specific application needs. As a result, IaaS customers must "over provision" by acquiring instances that are sized to support peak utilizations. More importantly, since many applications exhibit highly-variable resource utilization over time (*e.g.*, due to diurnal workload characteristics), and given the overheads associated with resizing acquired instances, IaaS customers may end up over-provisioning over extended periods of time.

Scope: In an IaaS setting, more efficient instance utilization could be achieved by appropriately *colocating* applications from multiple IaaS customers on the same instance. We note that virtualization allows *both* colocation and performance isolation of applications by viewing such applications as independent Virtual Machines (VMs). Such VM colocation could be done in a multitude of ways: (1) It could be offered as a (distinguishing) feature by the *IaaS provider*. (2) It could be developed as a mechanism that allows an *IaaS reseller* (a third party that is neither the provider nor the customer) to leverage the efficiencies resulting from IaaS customer aggregation, allowing it to offer more economical IaaS offerings to customers who are willing to colocate with others. (3) It could be used in a peer-to-peer fashion to allow *IaaS customers* to form coalitions that benefit from colocation.

Whether leveraged by IaaS providers, resellers, or customers, the functionality we envision requires the development of specific colocation services. To that end, this paper introduces the concept of *Colocation as a Service* (CaaS) and develops a framework in support of that concept by providing the means for efficiently allocating resources in a dynamic IaaS environment. Our framework is made concrete through a prototypical implementation of CaaS on top of the Xen hypervisor.

Challenges: The problem of VM consolidation has been previously considered, but only from a singular perspective – that of the provider (*i.e.*, assuming that both the infrastructure and the VMs belong to a single party) [4], [5], [6], [7], [8], [9], [10]. When VMs belong to different, independent, and self-interested customers (as is the case in an IaaS setting), VM consolidation must be seen as a process that *must* involve all parties involved since VM consolidation will ultimately affect the prices extended by providers and resellers, and the cost accrued by customers.

[†] This research was supported in part by NSF awards #0720604, #0735974, #0820138, and #0952145.

Another consideration when consolidating IaaS VM workloads is the fact that IaaS environments may be highly dynamic due to the churn caused by arrival and departure of VMs, and/or the need of customers to change their own resource reservations in tandem with changes in the workloads that their VMs must handle. In such settings, complex sets of VM migrations must be performed in such a way that the impact on performance is minimized.

Contributions: This paper presents Colocation as a Service (CaaS) – a framework for enabling VM colocation in such a way that the infrastructure is (quasi-)optimally utilized, while ensuring that the interests of customers (in terms of the costs that accrue for the resources they acquire) are guaranteed. Our CaaS framework outlined in § III requires the development of two sets of services: (a) strategic "match making" services that aim to identify groupings (or coalitions) of VMs that reflect the best interests (in a selfish as opposed to a socially-optimal sense) of the various parties involved, and (b) operational services that efficiently realize the quasi-optimal configurations obtained through the use of the strategic services. CaaS strategic services can be seen as implementing the decision-making processes that ensure that *each* individual customer is able to acquire the IaaS resources it needs at the least possible cost. Operational services can be seen as implementing what it takes to enable and carry out (act upon) the strategic choices made by (or on behalf of) customers.

For our CaaS framework to be practical, we must demonstrate that the set of services it provides do not negatively impact the IaaS value proposition. CaaS strategic services are off-line, in the sense that they could be carried out as background, control-plane processes: while they may consume resources, they have no negative impact on the performance of customer VMs. CaaS operational services, on the other hand, are on-line, in the sense that they involve the manipulation of (or interaction with) the VMs themselves, *e.g.*, for performance profiling or migration purposes. Thus CaaS operational services could potentially have a negative impact on the performance of customer VMs. Indeed, an important contribution of this paper is to demonstrate that such services could be implemented in such a way so as to make such an impact negligible. In particular, in § V we present a scheduling algorithm that is able to identify and carry out a VM group migration plan efficiently.

As a proof of concept demonstrating practicality and efficiency, and to enable us to report concrete performance metrics, in § IV of this paper we also present XCS – a prototypical implementation of CaaS on top the Xen Hypervisor. Our experimental evaluation in § VI – quantifying the cost-benefit as well as the performance overheads of CaaS – is based on measurements obtained by deploying real applications using XCS, as well as measurements obtained from extensive trace-driven experiments (using PlanetLab traces).

II. Background and Related work

VM consolidation and colocation are very active research topics. The goal of minimizing the operational cost of a data center (in terms of hardware, energy, and cooling), as well as providing a potential benefit in terms of achieving higher performance at no additional cost has been considered by Jason *et al*[6]. Also, a lot of work has gone into studying the consolidation of workloads across various resources: CPU, memory and network. Wood *et al* [8] promote colocation as a way to minimize the actual memory utilization by sharing portions of the physical memory between multiple colocated VMs. Network-aware consolidations have been addressed in [4], [9], [5], [10]. In [10], the authors consolidate VMs in a way similar to bin packing, while providing a monitoring technique that triggers migrations to resolve hotspots. Colocation has also been explored as the means for reducing the power consumption in data centers, for example by Cardosa *et al* [7].

VM migration mechanisms have an impact on CPU and network capacity. Recent work [11], [12], [13], [14], [15], [16] considered ways to migrate VMs so as to minimize CPU and network overheads, with careful consideration so as not to violate a client SLAs. We note that these mechanisms are designed for the benefit of the infrastructure provider, and indeed are carried out without the knowledge or consent of VM owners. They are done for the sole purpose of reducing the total cost of operation of the data center by reducing cooling and electricity costs. In our system, the purpose of migration is radically different (and so would be the resulting VM configurations). In particular, we view CaaS *not* as an optimization framework to minimize the provider's operational costs, *but rather* as a framework that enables an efficient marketplace that empowers various parties – customers, providers, aggregators, and resellers – to maximize their utilities.

Our CaaS strategic services bring into consideration the economics of the system from the customer's (selfish) perspective. Many resource management systems have been developed for large-scale computing infrastructures [17], [18], [19], [20], [21] using various micro-economic models, such as commodity markets, auctions, double-auctions, and combinatorial auctions. One particular line of work that influenced our conception of CaaS and our implementation of XCS is that by Londoño, Bestavros, and Teng [22], in which VM colocation is seen as a special case of more general pure-strategies *Colocation Games*. In that setting, customer interactions is driven by the rational behavior of users, who are free to re-locate and choose whatever is best for their own interests. The colocation games model in [22] has two attractive properties. First, it is shown that the interaction among customers for VM colocation purposes – termed as the Process Colocation Game (PCG) in [22] – converges to a Nash Equilibrium (NE), *i.e.* a state where no customer in the system is incentivized to migrate. Second, it is shown that the Price of Anarchy (the ratio between the overall cost of all customers under the worst-case NE and that under a socially-optimal solution) is bounded. Moreover, the bound is the same as the best-known approximation ratio for a centrally computed solution.

III. Colocation as a Service

We assume an IaaS setting consisting of any number of possibly heterogeneous instances (servers), to which we refer as "Physical Machines" (PM). Each instance is characterized by a number of capacitated resources (*e.g.*, CPU, network, memory, and disk space) which constitute dimensions of the instance capacity vector. Workloads associated with an instance are similarly characterized by a multi-dimensional

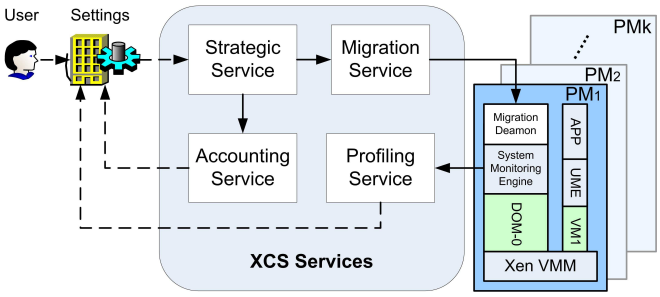

Fig. 1. CaaS Framework services (in the context of the XCS prototype).

utilization vector. Instances may have additional attributes (other than those associated with specific physical resources, *e.g.*, location, virtualization technology, *etc.*) that may also be considered as part of the customer workload SLA constraints. Without loss of generality, in this paper, we only consider dimensions related to physical resources.

As we alluded before, there are two types of services underlying a CaaS framework: strategic and operational. While strategic services may be implemented differently (*e.g.*, as a result of adopting different pricing schemes, or depending on whether CaaS is used by operators, resellers, or customers), operational services are assumed to be universal, in the sense that they would be useful (and necessary) under all possible settings.

CaaS services (as well as the data and control flows between them) are illustrated in Figure 1. Strategic services produce a desirable configuration which acts as the input to the migration service, which in turn issues migration commands to relocate VMs so as to realize that desirable configuration. In support of CaaS services, the virtualization infrastructure must support two functionalities – denoted in Figure 1 by the "System Monitoring Engine" (SME) and the "User Monitoring Engine" (UME). The SME is responsible for the collection of information about each customer's resource utilization, and is also responsible for the execution of the migration commands received from the migration service. The UME can be used by customers to gather additional, finer resolution information (*e.g.*, to measure application specific QoS characteristics, which may be needed for future provisioning purposes). All data gathered by either SME or UME for a particular VM is available to the VM's owner (the customer or a software agent thereof), who may utilize such data to adjust future resource reservations (*e.g.*, for the next reservation epoch). Based on the desirable configuration generated by the strategic services, a CaaS accounting service distributes accrued costs.

CaaS Strategic Services: Strategic services are invoked periodically, every *epoch* – *e.g.*, corresponding with the intervals used for instance reservation purposes. The input to these services consists of the users' workload (reservations) expressed in terms of an instance multi-dimensional resource utilizations (for CPU, memory, and network bandwidth, for example). The output of these services is the "desirable" VM configuration, reflecting what the service deems best for each customer in the system.

CaaS Operational Services: We discuss three types of operational services: migration services, accounting services, and monitoring and profiling services.

Migration: The migration service is needed at the beginning of any epoch in which a new VM configuration is requested. It takes as input the current as well as the desirable VM configuration and produces a migration plan, which it carries out by issuing migration commands to the constituent VMs.

Accounting: In addition to the cost that each user incurs as a result of their own use of resources, the various CaaS services themselves consume resources that must be accounted for as well (*e.g.*, in our XCS prototype, CaaS services are hosted on a VM that consumes resources just like any customer's VM would). The cost of CaaS services constitute an overhead that must be borne by the parties benefiting from CaaS services (provider, reseller, or customers). Clearly, there are many ways to appropriate the cost of such overhead. For instance, if customers are to collectively bear such costs, one option may be to charge only the users who use the strategic service. Another option would be to distribute the cost in proportion to the benefit that customers begets from relocation. Yet, a third option (and the one we adopt in our XCS prototype) would be to charge customers equally, independent of whether they benefit from the CaaS services or not.

Monitoring and Profiling: The monitoring service[1] is responsible for tracking CPU, network and memory utilization of each VM. It tracks the usage of resources over a predefined interval α. This service is offered through two engines. The System Monitoring Engine (SME) runs on the physical machine and monitors all colocated user instances on that physical machine, whereas the User Monitoring Engine (UME) provides monitoring from within each VM. While the SME does not require any modification of the VM (user application), it is only able to provide raw resource utilization data. The UME, on the other hand, allows tracking of application-specific metrics (*e.g.*, response times), allowing for finer management of SLAs.

IV. The XCS Prototype

In this section we present the Xen Colocation Service (XCS): a prototype implementation of our CaaS framework. We chose the open-source Xen platform [23] because it supports physical resource reservations, as well as live migration using iterative bandwidth adapting (pre-copy or post-copy) memory page transfer algorithms [24], [25].[2]

The XCS Cluster: Our XCS prototype consists of a cluster of 3.2 GHz quad-core hosts (the PMs), each of which with 3 GB of memory. The servers are connected to each other using a one gigabit Ethernet, and to a network file server containing the VMs' disk images. One of the PMs is dedicated to supporting the various XCS functionalities and services, while the remaining PMs are free to host VM instances of different sizes. All PMs run Xen 3.4.2, and all hosted VMs run different versions of Linux Fedora distribution.

Each PM in the cluster consists of a Virtual Machine Monitor (VMM) and at least one VM instance. VM instances are allocated specific fractions of the PM resources using

[1]The main purpose of this service is to provide users with the raw data that enables them to adjust their reservations. Users might adopt different techniques to analyze/process this data (*e.g.*, averaging or statistical time series analysis/prediction).

[2]We note that our CaaS framework could be realized atop other virtualization technologies such as VMWare [26], or hosted as a service on top of systems such as Kittyhawk [27].

Xen reservation APIs. In particular, for the CPU, we use the *cap* option of Xen. This option caps the CPU cycles that a domain is able to get through the Xen credit scheduler, which assigns the CPU bandwidth fairly across all domains without exceeding the assigned *cap* levels, even if the physical host has idle CPU cycles. For memory, we are able to allocate specific amounts of physical RAM to each VM. As for network bandwidth, and since the Xen API does not provide for an explicit allocation of network bandwidth, we have implemented a Linux packet filter to support it.

XCS's strategic and run-time support services are java based applications. The user and system monitoring engines (UME and SME) are implemented in python. The SME is set up to collect and report statistics about CPU, network, and memory utilizations every 5 seconds. The monitoring engine is also used to execute migration commands with the use of Xen's migration and resource reservation commands.

The XCS PCG Strategic Service: Recall that our CaaS framework calls for the implementation of a strategic service that is responsible for initiating any necessary reconfiguration of the VMs in the cluster. In our XCS prototype, we have chosen to implement a specific strategic service based on the *Process Colocation Game* (PCG) game-theoretic formulation in [22].[3] Under PCG, each PM has a fixed cost (*e.g.*, corresponding to what an IaaS provider would charge for it). This cost is split among all VMs (users) sharing the PM in proportion to their fractional use of the PM's resources.[4] This pricing is assessed periodically, every *colocation epoch* – taken to be 5 minutes in our XCS implementation. Under PCG, each VM (player) is able to entertain a "move" to a different PM, if such a move would reduce its cost (at the next epoch). So, if x_i is the utilization of the resource by user i and the total utilization of the resource is $U = \sum_i x_i$, then the cost for user i is $c_i = P \cdot x_i/U$; where P is a constant denoting the price of the resource.

As established in [22], such moves by the various players are guaranteed to converge to a fairly efficient Nash Equilibrium (NE), constituting a new configuration of colocated VMs on the cluster.[5]

To support PCG within our XCS prototype, we have implemented APIs that allow users to specify the various resource reservations they require (see Figure 1). Users can change such settings at any time (*e.g.*, as a result of changes in the workload, resulting in degraded performance as reported by the profiling services), but such changes are not effective until the strategic services are invoked again at the end of the current colocation epoch to effect the colocation in the following epoch.

The XCS strategic service we have implemented can be seen as carrying out the PCG game every epoch. For a given VM (player), this is done by evaluating if there is a *"better response"* for such a player – *i.e.*, a colocation that reduces the overall cost of the player – in the upcoming epoch. This cost reduction may be required to be above some pre-defined threshold to avoid relocations that are not justifiable (relative to the overhead of underlying migrations). By letting players take (random) turns in evaluating and executing their better responses, the PCG game eventually (and typically very rapidly) converges to a Nash Equilibrium – when no cost-reducing moves are possible for any player (VM) in the system. As depicted in Figure 1, at this point, the XCS strategic service reports this new configuration to the migration service, which realizes it for the next epoch.

When new users (VMs) join the XCS cluster, they are assigned exclusive PMs, and they are charged accordingly for the remainder of the current epoch. By invoking the XCS strategic services, such users are able to participate in the next PCG epoch, and thus are able to find cost-effective colocations. User departures result in the deallocation of the corresponding VMs (and associated reservations), thus excluding such users from being part of the next PCG epoch.

The XCS DTM/UTM Migration Services: The XCS migration service is responsible for efficiently realizing the colocation configuration obtained through the XCS strategic service (*i.e.*, the NE outcome of the PCG). In particular, given a configuration for the current epoch, the migration service must determine the set of VMs that must be migrated as well as the ordering of such migrations so as to realize the configuration requested by the strategic service in the most efficient way. For example, as illustrated in Figure 2, consider a current configuration in which VM_1 and VM_2 are colocated on the same PM, while VM_3 is by itself. If for the next epoch, the strategic service requests the colocation of VM_2 with VM_3 (instead of VM_1), then it will be necessary to migrate VM_2 from the first PM to the second, or else to migrate VM_1 from the first PM to the second and migrate VM_3 from the second PM to the first. Which of these two options to follow is a decision that the migration service must determine and execute. To that end, we have developed two migration services: A Data Transfer Minimization (DTM) migration service and a User Transfer Minimization (UTM) migration service. DTM aims to minimize the amount of data that needs to be transferred in the system, whereas UTM aims to minimize the number of users (VMs) that are migrated. In the next section, we describe the algorithms underlying both DTM and UTM, showing that they are within a constant factor of the optimal.

The XCS Monitoring and Profiling Services: The XCS monitoring service uses a similar approach to those presented in [4], [13], [30], [31]. The monitoring engines send their data to the profiling service whose purpose is to generate a profile of the users' resource utilization. Users may opt to use this information to change their resource reservations, which will only be effective during the next reconfiguration epoch. We implemented profiling models based on the Exponential Weighted Moving Average (EWMA) and percentiles profiling (95 percentile), with the option of changing the user reservation only upon user consent.

The SME runs as a service in Domain0 and provides the system with information about the volume of the user workload – specifically processor, network and memory utilization. To track CPU usage of VMs, we utilize the xenmon [32] monitoring tool because of its low CPU utilization. Network monitoring uses the Linux /proc/net/dev interface, which allows us to monitor the number of bytes sent and received on each Xen bridged interface. Monitoring a VM's

[3]The XCS prototype could be easily adapted to support other choices.

[4]*Shapley cost* pricing [28] has the desirable property of being budget balanced while satisfying a strong notion of fairness between VMs (users).

[5]For PMs with homogeneous capacities and unit prices (which is the case in our setting), the Price of Anarchy (PoA) is $3/2$ [22], which is quite efficient since there is no approximation algorithm for the *on-line* bin-packing problem with an approximation ratio better than $3/2$ [29].

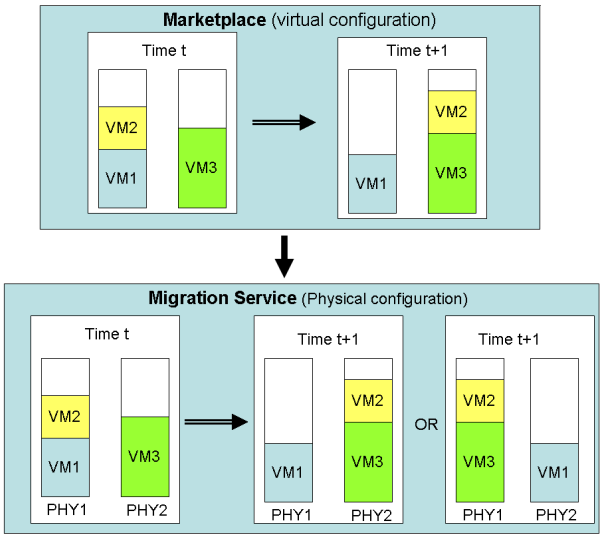

Fig. 2. Mapping the Virtual Configuration into a Physical Configuration.

main memory is non trivial, since Xen only provides a reservation bound on the amount of each VM's main memory usage. To that extent, we use the working set size as an indicator. To estimate the working set size of each VM, we use the technique defined in [13], [30], where the reads and writes to each swap partition are possible to monitor as they reside on the network and can be tracked. This technique allows us to infer accurately information about the memory usage of the system during high load periods.

The UME runs inside the VM and provides two options for gathering CPU, memory, and network utilization: through the `/proc` interface or through `dstat` process [33] monitoring tool. Since users have full control over their VMs, they are able to employ their own monitoring and profiling scripts. The profiling service API is designed to accept input from a possibly-user-modified monitoring service.

The XCS Accounting Service: We have implemented an accounting service that apportions the cost of using all XCS services on all system users, equally. Here we note that we have considered other choices for how to apportion the overhead costs of XCS over all users,[6] but opted to stick to this simple approach, which has the added advantage of being highly scalable to a large number of users.

V. Efficient Migration in XCS

In this section, we present the efficient migration strategies adopted in XCS.

VM Bundles: We elaborate on the example configuration in Figure 2. Assume that the memory footprint of VM_2 (*i.e.*, its size) is 1, whereas that of VM_1 and of VM_3 is 1,000. Migrating VM_2 alone would cost 1, but migrating both VM_1 and VM_3 (swapping them) would cost 2,000. Thus, a sensible approach would be to avoid migrating VMs with larger footprints. While such an approach would work for the case illustrated in Figure 2, it will be rather inefficient for other cases. In particular, if we replace each of VM_1 and VM_3 with 2,000 VMs, each of which with size 0.5, then an algorithm that chooses not to migrate VMs with larger footprints will end up migrating 4,000 VMs of size 0.5, instead of a single VM of size 1. This demonstrates that the cost of migration using such a simple heuristic can be arbitrarily large, when compared to the optimal solution. This leads us to the idea of VM bundles, which we define next.

Given two VM configurations – one for the current epoch and another for the next epoch – a *VM bundle* is defined to comprise all VMs that are colocated on the same PM in *both* configurations. VM bundles (or simply bundles) define the entities that migration heuristics must handle.

Basic Definitions and Notation: Let $S(x)$ be the memory footprint of bundle x. Let $M(P)$ be the cost of migrating a set of bundles P, which equals the size of all the bundles in P that need to relocate from their current PMs to new PMs. We define $\ell(x)$ to be the PM where bundle x is located (in the current/old configuration) and $\ell'(x)$ to be the PM where bundle x should be relocated (in the next/new configuration).

Consider bundle x in the configuration illustrated in Figure 3. We denote by $\{\alpha_k\}$, the set of bundles that will be colocated with x in the new configuration,[7] and we denote by $\{\beta_k\}$ the set of bundles that were colocated with x in the previous configuration – *i.e.*, $\ell'(\alpha_k) = \ell'(x)$ and $\ell(\beta_k) = \ell(x)$. Let A_k denote the set of bundles colocated with α_k in the old configuration, and let B_k denote the set of bundles colocated with β_k in the new configuration – *i.e.*, $\ell(\alpha_k) = \ell(A_k)$ and $\ell'(\beta_k) = \ell'(B_k)$.

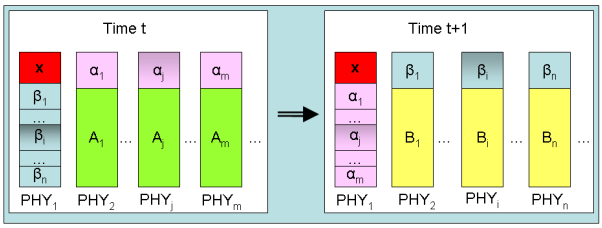

Fig. 3. DTM keeps the biggest bundle x and relocates all α_k, β_k bundles.

Data Transfer Minimization (DTM): To minimize the total data migrated, we use the greedy Algorithm 1 (over bundles) shown below. The algorithm works recursively by pinning the bundle x with the largest footprint (Step 1) to the PM it is currently in, thus ensuring that this bundle will not move (Step 2). Then, the bundles to be colocated with x in the new configuration, *i.e.*, α_k are moved into the same PM (Steps 3-5). Once the location of all bundles that used to be colocated with x in the old configuration is determined (as a result of the recursive call in Step 7), these bundles are moved to their new PM (Steps 8-10).

An important aspect of our DTM algorithm relates to the resources needed to migrate VMs, since the spatial capacities of existing PMs may not be able to accommodate such swapping. Thus, we distinguish between *spatially constrained*

[6]Other strategies such as charging only the users who migrate would be counter intuitive to use since our system tries to minimize the number of migrations, and an efficient migration service (such as DTM and UTM) might end up migrating users who might not have requested to migrate. Dividing the cost over the players who benefit from the system was another option we considered. Given the selfish nature of system users, such an approach would result in introducing complexities (by making the strategic and accounting services inter-dependent – *e.g.*, the pricing we used for PCG would not hold).

[7]Conveniently, we use k for enumeration and omit quantifiers over k.

Algorithm 1 DTM(P)

1: $x \leftarrow \arg_{y \in P} \max S(y)$
2: $\ell'(x) \leftarrow \ell(x)$
3: **for all** α_k colocated with x in the new configuration **do**
4: $\quad \ell'(\alpha_k) \leftarrow \ell(x)$
5: **end for**
6: $P' \leftarrow P - (\{x\} + \bigcup_{\forall k}\{\alpha_k\} + \bigcup_{\forall k}\{\beta_k\})$
7: DTM(P')
8: **for all** β_k colocated with x in the old configuration **do**
9: $\quad \ell(\beta_k) \leftarrow \ell'(B_k)$ (determined by the recursive call)
10: **end for**

settings in which the PMs are memory constrained (*i.e.*, migration may require the use of intermediate PMs) and *spatially unconstrained* settings in which the PMs are not memory constrained (*i.e.*, migration does not require the use of intermediate PMs). Figure 4 illustrates such a case wherein no physical machine can hold 3 VM's at the same time, necessitating the use of an extra (or intermediate) PM.

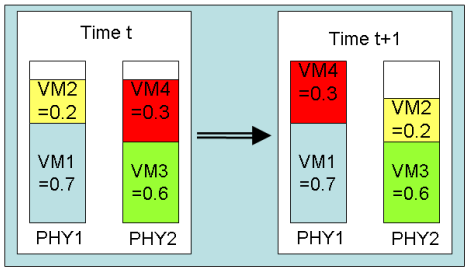

Fig. 4. Example: A temporary PM is needed to swap VM2 and VM4

This DTM greedy algorithm is not optimal. Taking for example the configuration in Figure 2, if VM_2 has slightly larger memory footprint it won't move thus causing both VM_1 and VM_3 to move. The sum of memory footprint of VM_1 and VM_3 would be slightly less than twice the footprint of VM_2, in this case our heuristic is slightly less than twice as optimal.

The following theorem establishes a constant-approximation bound on DTM's performance relative to an optimal algorithm.

Theorem 1: In spatially unconstrained settings, the DTM algorithm results in at most twice the amount of data transfer incurred by an optimal algorithm, whereas in spatially constrained settings, it results in at most four times the amount of data transfer.

Proof Sketch: A complete proof is provided in the appendix. In unconstrained settings, we observe that at each iteration of Algorithm 1, the optimal solution may keep two smaller bundles in a PM (instead of the largest bundle chosen by DTM). Thus, by pinning down the largest bundle in the PM, we accrue the cost of migrating the other two bundles, which are smaller in size by definition. This yields the 2-approximation result for unconstrained settings. In constrained settings, we further observe that in the worst case each DTM migration may require two moves (through an intermediary PM), another factor of two blow-up in the amount of data transferred. This yields the 4-approximation result for constrained settings. ■

User Transfer Minimization (UTM): Rather than minimizing the amount of data movement resulting from a reconfiguration, UTM minimizes the number of VMs affected (disrupted) by the reconfiguration. The UTM algorithm is identical to that of DTM, except that the "size" of a bundle is taken to be the number of VMs that constitute the bundle (as opposed to the memory footprint of the bundle). The performance bounds for UTM are also similar to those of DTM – a 2-approximation for unconstrained settings and a 4-approximation for constrained settings.

Concurrent Migration Scheduling: Our base implementation of DTM (UTM) produces migration requests sequentially to minimize the intermediary space needed to hold VMs as they are moved in an out of PMs. While efficient in space, performing migrations sequentially can be a time consuming process, especially for large clusters.[8] To minimize migration time, it would be natural to issue migration commands in parallel. However, since migration can have an effect on both the source and destination PMs (in terms of resources consumed), we must ensure that concurrently executing migrations do not conflict.

The problem of parallelizing migrations is similar to a scheduling problem for storage systems, which was proven to be NP-complete [34] by reducing it to the graph edge coloring problem. In [34], the authors present heuristics to solve the problem under various constraints. For XCS migrations services, we follow a similar approach by developing a service that maximizes the number of concurrent migrations, while ensuring that no source, destination, or intermediary PM is involved in more than one VM migration at a time (*i.e.*, either sending or receiving, but not both).

VI. Performance Evaluation

In this section we present results from extensive experimental evaluations of our XCS prototype. Our main motivation is to establish the feasibility of CaaS as incarnated in our XCS prototype by: (1) contrasting the performance of various XCS design choices, *e.g.*, DTM versus UTM, (2) show that the impact of deploying XCS on the performance of hosted VMs is minimal, (3) quantify the potential monetary gains that users of XCS services may stand to achieve.

DTM versus UTM: We use trace-driven workloads to contrast the performance of DTM versus UTM. The traces we use were derived from publicly available PlanetLab traces of CoMon [35]. PlanetLab is an example of a hosting infrastructure that allows researchers to submit tasks that utilize various resources from the PlanetLab servers. The traces we used give us snapshots of PlanetLab server capacities, as well as the utilization of the slices assigned to the various tasks (users) colocated on each server. The main advantage of this data set is that it gives us a realistic distribution of *typical* task utilizations on a fairly large scale.

We used the PlanetLab data set to synthesize a workload in which user (VM) arrivals are Poisson with a rate λ, and in which user (VM) session times follow either an exponential or a powerlaw distribution with a mean δ. Each user (VM) session acquires a slice with resource utilizations that are drawn from the Planet Lab data set. In our experiments, we set the arrival rate to be $\lambda = 10$ users/epoch, and we set the

[8]Notice that prolonging the time it takes to realize a new configuration does not necessarily imply performance degradation for the hosted VMs, since for a given VM, any significant impact on performance will be due to live migrations involving that VM.

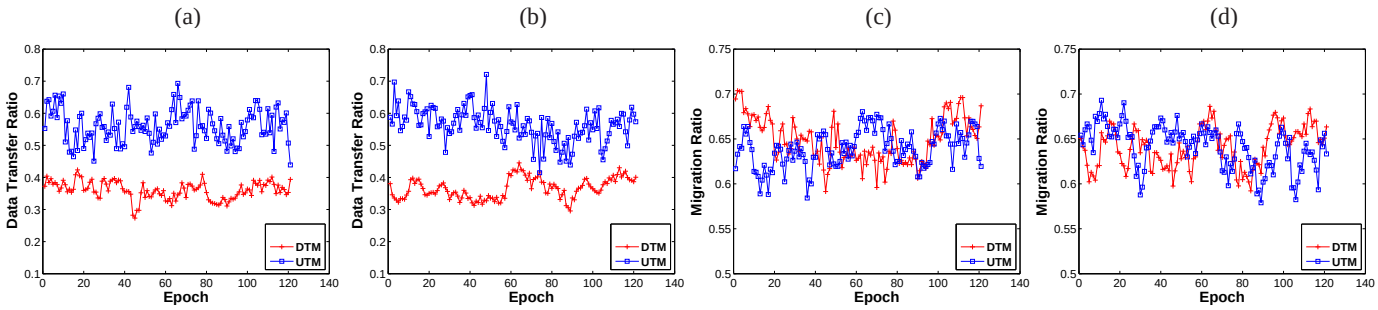

Fig. 5. Performance of DTM and UTM algorithms: Under exponential (a & b) and powerlaw (c & d) session times.

average session times to $\delta = 10$ epochs (for an average of 100 VMs in the system). We also assume the existence of only one intermediary PM to be used to perform swappings if needed (this is a conservative assumption).

We ran the experiment for 150 epochs, with the first 30 epochs used for warm-up. Every epoch (of five minutes), the XCS strategic service is invoked on behalf of all users in the system, and the resulting colocation configuration is realized using either DTM or UTM. Figures 5(a) and 5(b) show the percentage the data migrated in the system (by normalizing the amount of data transferred by the total amount of data in the system). As expected, these results show that DTM incurs much less data transfer overhead than UTM. Figures 5(c) and 5(d) show the percentage of the VMs migrated (by normalizing the number of migrations by the total number of VMs in the system). Surprisingly, the performance of both UTM and DTM is quite similar. The results in Figures 5 show that the results under exponential and powerlaw session times are quite similar. Overall, these results confirm that the DTM algorithm is superior to the UTM algorithm.

Impact of XCS Migrations on VM Performance: We consider the effect of frequent live migrations on the performance of a *migrating* VM as well as on the performance of *non-migrating* VMs, which might be impacted by overheads due to data transfers or other overheads underlying live migration. To do so, we use a baseline VM running the (TPC-W) web application benchmark [36]. Our choice of the TPC-W benchmark is motivated by the fact that I/O-bound applications are more prone to disruption of service due to migrations than CPU-Bound applications. The TPC-W benchmark models an Amazon-bookstore-like web application that provides workload generators that simulate multiple concurrent browser clients accessing the application. Our TPC-W benchmark uses MySQL 5.05 and runs on Tomcat version 5. All experiments are repeated multiple times, and all results are reported with 95% confidence.

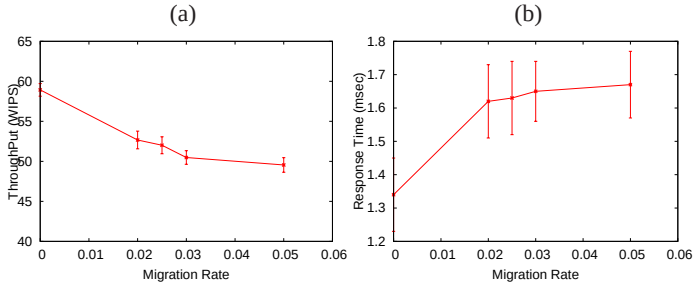

Fig. 6. Impact of migration on throughput (a) and response time (b).

To measure the impact of XCS migration services on the migrated VM, we use a setting consisting of two PMs. We run a single VM that hosts the TPC-W web application on one of the two PMs, and we perform migrations of that VM from the PM it is running on to the other PM under a variety of conditions. We use the TPC-W workload generator to create 350 clients which are run (on external hosts) against the TPC-W server for 300 seconds, allowing for server throughput and response time metrics to be collected.[9]

Figure 6(a) shows the average throughput of the TPC-W server, measured in Web Interactions per Second (WIPS). Figure 6(b) shows the corresponding average response time observed by clients, measured in milliseconds. The migration rate (on the horizontal axis) is measured in Hertz and ranges from one migration every 100 seconds, to one migration every 20 seconds. As indicated by the results in Figure 6, and as expected, as the TPC-W server is migrated more frequently, its throughput is decreased (by as much as 15%) and its response time is increased (by as much 30%). These results are quite encouraging since the rate with which a single VM is likely to migrate is likely to be much less than the rates we used. With an epoch of 5 minutes, the maximum migration rate for a VM is once every 300 seconds.

We now turn our attention to the impact of XCS on VMs that are not involved in migrations (*e.g.*, due to contention over system resources, such as network bandwidth, CPU, and memory). We do so using a setting involving two VMs on two PMs. One of the VMs is pinned to one of the PMs whereas the other is made to relocate back and forth between the two PMs every colocation epoch. The non-migrating VM runs the TPC-W web application benchmark (as before), whereas the migrating VM is configured to use various percentages of the PM resources (to evaluate the impact of contention over various resources).

In particular, we report results for two extreme settings. In the first, the migrating VM is CPU-bound, *i.e.*, it hosts an application that uses 100% of the CPU cycles allocated to it by XCS. In the second, the migrating VM is network-I/O-bound, *i.e.*, it hosts an application that uses 100% of the network bandwidth allocated to it by XCS. In both of these cases, we perform XCS migrations at different rates. And, as before, we use the workload generator provided with TPC-W to measure performance (throughput and response time) with 350 clients for 300 seconds.

Figures 7(a) and 7(c) show the average throughput and response times of the TPC-W server in the presence of an I/O-bound migrating VM, whereas Figures 7(b) and 7(d)

[9]The impact of VM live migration on VM performance is well documented in the literature, *e.g.*, [24], [37].

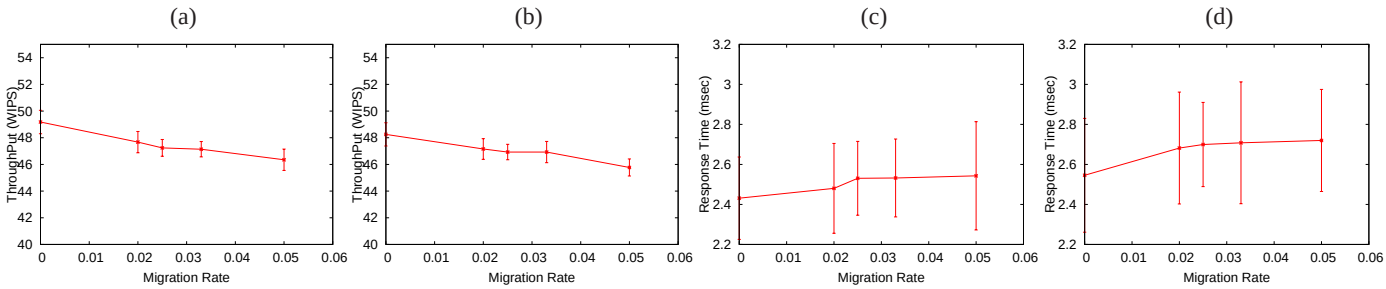

Fig. 7. Impact of XCS services on non-migrating TPC-W VM: Effect on throughput (a & b) and on response time (c & d).

show the same results for a CPU-bound migrating VM. As with our results for migrating VMs, we note that the impact of XCS migration on non-migrating VMs is also more pronounced as migration rates increase. However, we note that the impact in this case (even with fairly high migration rates) is minimal – limited to a 5% decrease in throughput and a 4-7% increase in response time.

Overall Cost Savings: We use our XCS prototype to measure the cost savings that are possible to achieve per user. To do so, we run the system with a random set of 100 users (VMs) selected from the PlanetLab workload. We assume that if users opt not to use our system, then they will end up paying a unit cost. Using our system, the XCS system calculates the cost of each user (including the amortized cost of running all XCS services – strategic and operational).

Figure 8(a) shows the cumulative distribution function (CDF) of the *cost savings* accrued by users (the difference between the unit cost of using a PM exclusively and the cost of colocation through the XCS system). These results suggest that (at least for PlanetLab-like workloads) more than 50% of the users achieve savings upwards of 50% – with the average cost saving for all users being 33%. Our results also show that the amortized costs of the XCS services are insignificant compared to the actual savings achieved by most users (VMs). In particular, the average cost per user is 0.64, of which only 0.01 accounts for the amortized cost of running the various XCS services.

To appreciate the cost savings achievable through the use of XCS, we also calculate the *cost inflation ratio* (CIR). For a given user, CIR is the ratio of the *actual* cost borne by the user to the *Utopian* cost for that user, where the Utopian cost is the minimal possible cost – reflecting only the cost of the resources that the user actually uses. Utopian costs are achievable only if a perfect packing of the VMs into PMs is possible. Figure 8(b) shows the CDF of the CIR for all users: Almost 40% of all users end-up with costs that are indistinguishable from Utopian costs (*i.e.*, with a CIR approaching 1) and over 95% of all users achieve a CIR less than 1.5, which is the best approximation ratio known for bin packing.

System Scalability: The main overheads in XCS are those attributed to supporting strategic services (namely the better-response computation needed for PCG) and supporting migration services (namely the data transfers underlying live migrations). For strategic services, supporting up to 500 users in real-time was shown to be quite feasible [22]. Beyond that, finding better-responses may become computationally expensive and reaching NE take a long time. For migration using DTM, our experiments suggest that supporting hundreds of

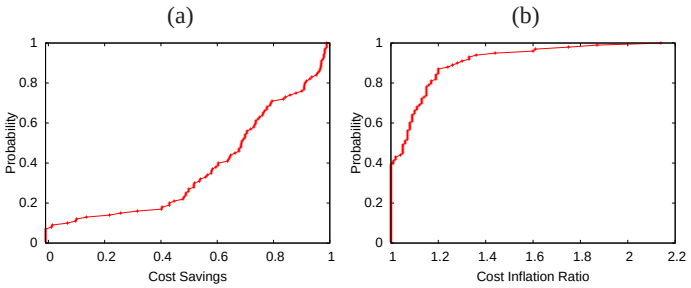

Fig. 8. CDF of user cost savings (a) and inflation ratio (b).

users is quite feasible with respect to data transfer overheads. For extremely large clusters with high churn rates (either high arrival/departure rates or high reservation adjustment rates), the total amount of data transfers in support of XCS migrations might become an impediment to scalability. Even beyond these large scales, it is possible to boost scalability by partitioning users into clusters — based on broad criteria (*e.g.*, based on geographical proximity or based on similarity in session lifetimes or workload characteristics). These clusters could be managed using independent XCS services.

VII. Conclusion

Significant cost savings could be realized by supplying cloud tenants with the means to efficiently colocate their workloads on cloud resources. In this paper, we have shown the feasibility and promise of the *Colocation as a Service* (CaaS) concept. We presented the blueprints of a CaaS framework, which we instantiated on a Xen virtualization platform by implementing and evaluating a *Xen Colocation Services* (XCS) prototype. Our work on XCS made use of specific design and implementation choices; our current work extends XCS to accommodate different choices as well as to enable new functionalities, which we exemplify below.

Our choice of a game-theoretic formulation for strategic services anticipates a "peer-to-peer" CaaS deployment, in which the customers (peers) are autonomous and self-interested (selfish), and in which a single provider offers cloud resources (which are assumed plentiful) at fixed prices. Alternative deployment scenarios – *e.g.*, in which one or more providers adopt a dynamic pricing scheme based on supply and demand, or in which third parties may act as resellers – would necessitate making different choices with respect to the strategic services to be implemented in a CaaS offering, *e.g.*, using auction or optimization techniques.

In the work presented in this paper, we have restricted our attention to colocation on resources that admit multiplexing through the reservation of (additive) resource capacities.

In many settings, however, efficient resource multiplexing by colocated tenant may depend highly on a more granular workload specification. For example, if services to be co-hosted are of a periodic, real-time nature, then the identification of groupings of tenants that can be efficiently colocated would require the development of new functionalities and services, such as those envisioned in our recent work on the colocation of periodic real-time systems [38].

APPENDIX
PROOF OF THEOREM 1

Proof: The basic idea underlying the proof is the observation that, at each time step, the optimal solution may choose to keep only two other bundles instead of the biggest one which DTM chooses to keep. All other bundles (which were or will be colocated with the biggest bundle) would have to move anyway. Thus, by choosing the biggest bundle, DTM forces only two bundles, smaller in size by definition, to migrate. Thus our algorithm is a 2-approximation.

Figures 3 and 9 show two mutually exclusive migration decisions. In the first (Figure 3), bundle x gets to stay at the the same PM and bundles $\alpha_k, \forall k$ and $\beta_k, \forall k$ migrate. The cost of migration is:

$$M_1(P) = \sum_{k=1}^{m} S(\alpha_k) + \sum_{k=1}^{n} S(\beta_k) + M(P')$$

where $M(P')$ is the cost to migrate the rest of the bundles as mentioned in the notation section.

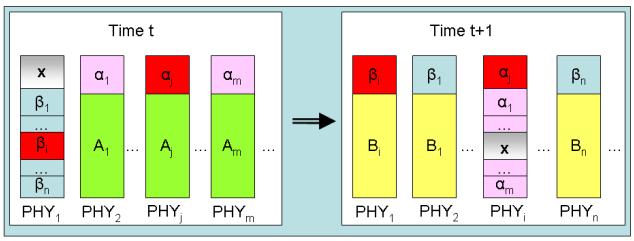

Fig. 9. Option 2: Relocate bundle x along with bundles $\alpha_k, \beta_k, \forall k$ except α_j and β_i.

In the second option (Figure 9), x will be the one migrating. This option is general enough to cover all other possibilities as we only have two options, to keep x or to migrate x. By definition only one of the β bundles, if any, denoted β_i is allowed to stay in the same PM, while all other β bundles will migrate along with x. Only one α bundle, if any, denoted α_j will stay while all the other α bundles will migrate. The cost of migration in this case is

$$M_2(P) = S(x) + \sum_{k=1k\neq j}^{m} S(\alpha_k) + \sum_{k=1k\neq i}^{n} S(\beta_k) + M(P'_c)$$

Notice that $M(P'_c)$ is a constrained version of $M(P')$ as all the bundles in the set B_i will have to migrate in, because β_i will stay in the same PM. Also all bundles in the set A_j will have to migrate, because α_j will stay in the same PM. This is not the case in the first option as one of the bundles of each of the sets B_i and A_j is allowed to stay in the same PM. Thus, if we use the same migration algorithm, denoted Alg, on both P' and P'_c we are certain that

$$M_{\text{Alg}}(P') \leq M_{\text{Alg}}(P'_c)$$

DTM *always* chooses the first option. Thus,

$$M_{\text{DTM}}(P) = M_1(P)$$

Now, assume the existence of an *Oracle* (OPT) that knows the optimal way to migrate the set of bundles P by returning the minimum of the two options:

$$M_{\text{OPT}}(P) = \min\{M_1(P), M_2(P)\}$$

If OPT chooses the first option, then the cost for migrating $\{\alpha_k \forall k, \beta_k \forall k\}$ is the same as DTM. Thus

$$M_{\text{OPT}}(P) = \sum_{k=1}^{m} S(\alpha_k) + \sum_{k=1}^{n} S(\beta_k) + M_{\text{OPT}}(P')$$

By induction, if we assume that

$$M_{\text{DTM}}(P') \leq 2 * M_{\text{OPT}}(P')$$

then

$$\sum_{k=1}^{m} S(\alpha_k) + \sum_{k=1}^{n} S(\beta_k) + M_{\text{DTM}}(P') \leq$$
$$2 * (\sum_{k=1}^{m} S(\alpha_k) + \sum_{k=1}^{n} S(\beta_k)) + 2 * M_{\text{OPT}}(P')$$

which means $M_{\text{DTM}}(P) \leq 2 * M_{\text{OPT}}(P)$.

If OPT chooses the second option (to migrate x and keep α_j and β_i), then by definition $S(\alpha_j), S(\beta_i) \leq S(x)$, otherwise they would have been chosen as x. The maximum penalty for choosing the first option would be when $S(\alpha_j) = S(\beta_i) = S(x) - \epsilon$. Starting from the induction case where we assume that:

$$M_{\text{DTM}}(P') \leq 2 * M_{\text{OPT}}(P')$$
$$M_{\text{OPT}}(P') \leq M_{\text{OPT}}(P'_c)$$
$$M_{\text{DTM}}(P') \leq 2 * M_{\text{OPT}}(P'_c)$$

$$\sum_{k=1}^{m} S(\alpha_k) + \sum_{k=1}^{n} S(\beta_k) + M_{\text{DTM}}(P') \leq$$
$$S(a_j) + S(\beta_i) + 2 * (\sum_{k=1k\neq j}^{m} S(\alpha_k) + \sum_{k=1k\neq i}^{n} S(\beta_k)) +$$
$$2 * M_{\text{OPT}}(P'_c)$$

$$M_{\text{DTM}}(P) \leq 2 * S(x) + 2 * (\sum_{k=1k\neq j}^{m} S(\alpha_k) +$$
$$\sum_{k=1k\neq i}^{n} S(\beta_k)) + 2 * M_{\text{OPT}}(P'_c)$$

$$M_{\text{DTM}}(P) \leq 2 * M_{\text{OPT}}(P)$$

By iteratively choosing x as the biggest bundle in the set P, resulting in the set P' such that $|P'| < |P|$. Eventually P will be empty. The base case is when the remaining set is empty. In this case $M_{\text{DTM}}(\phi) = M_{\text{OPT}}(\phi) = 0$.

Thus in spatially unconstrained setting, MDT results in 2-approximation of the optimal.

In spatially constrained settings, we might want to move a VM to a destination PM but there is no free space in its memory for it.

We can solve any swapping problem in our DTM by simply migrating all the VMs that need to migrate, initially, to temporary machines then migrating them back to their final destination. In this case we double the amount of data migrated. On the other hand the best case for optimal solution is when it doesn't need to do an VM swapping. Thus the amount of the data migrated in the optimal case remains the same. Thus our DTM algorithm with swapping all migrating machines is at worst 4 times the optimal solution with swapping no machine. Thus, in this case DTM is 4-approximation at worst. ■

REFERENCES

[1] Amazon.com, Inc., "Amazon Elastic Computing Cloud (Amazon EC2)," http://aws.amazon.com/ec2/, Jan 2010.

[2] "Eucalyptus project," http://eucalyptus.cs.ucsb.edu/, 2009.

[3] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[4] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for i/o performance." in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA: ACM, 2009, pp. 101–110.

[5] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proc. of the IEEE/ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009.

[6] J. Sonnek and A. Chandra, "Virtual Putty: Reshaping the Physical Footprint of Virtual Machines," in *USENIX/HotCloud'09*, 2009.

[7] M. Cardosa, M. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Proc. of IFIP/IEEE Integrated Network Management 2009*, 2009.

[8] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers," in *VEE '09*, 2009.

[9] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms," in *VEE '07*, 2007.

[10] G. Khanna, K. A. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *NOMS*, 2006, pp. 373–381.

[11] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 25–25.

[12] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 377–390, 2002.

[13] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI*, 2007.

[14] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven server migration for internet data centers," in *Proceedings of IWQoS*, 2002.

[15] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *VEE '07*. New York, NY, USA: ACM, 2007.

[16] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proc. of the 18th ACM international symposium on High performance distributed computing*, 2009.

[17] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik, "G-commerce: Market formulations controlling resource allocation on the computational grid," *IPDPS*, 2001.

[18] J. Gomoluch and M. Schroeder, "Performance evaluation of market-based resource allocation for grid computing: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 16, no. 5, pp. 469–475, 2004.

[19] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[20] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid." IEEE Computer Society Press, 2000, pp. 283–289.

[21] A. AuYoung, P. Buonadonna, B. N. Chun, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat, "Two auction-based resource allocation environments: Design and experience," in *Market Oriented Grid and Utility Computing*, R. Buyya and K. Bubendorfer, Eds. Wiley, 2009, ch. 23.

[22] J. Londoño, A. Bestavros, and S.-H. Teng, "Collocation Games And Their Application to Distributed Resource Management," in *USENIX/HotCloud'09*, 2009.

[23] 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 *SOSP '03*, 2003.

[24] C. C. Keir, C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI*, 2005, pp. 273–286.

[25] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, 2009.

[26] VMWare, http://www.vmware.com.

[27] J. Appavoo, V. Uhlig, and A. Waterland, "Project kittyhawk: building a global-scale computer: Blue gene/p as a generic computing platform," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 1, pp. 77–84, 2008.

[28] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, September 2007.

[29] V. V. Vazirani, *Approximation Algorithms*. Springer, March 2004.

[30] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Geiger: monitoring the buffer cache in a virtual machine environment," in *ASPLOS-XII*, 2006.

[31] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proc. of the ACM/IFIP/USENIX International Conference on Middleware*, 2006.

[32] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," HP Labs, Tech. Rep. HPL-2005-187, 2005.

[33] Dstat, "Versatile resource statistics tool," http://dag.wieers.com/home-made/dstat/, Jan 2010.

[34] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes, "On algorithms for efficient data migration," in *SODA '01*, 2001.

[35] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, 2006.

[36] Wayne D. Smith, "TPC-W: Benchmarking An Ecommerce Solution." http://www.tpc.org/information/other/techarticles.asp.

[37] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *CloudCom*, 2009, pp. 254–265.

[38] V. Ishakian, A. Bestavros, and A. Kfoury, "A Type-Theoretic Framework for Efficient and Safe Colocation of Periodic Real-time Systems," CS Dept, Boston University, Tech. Rep. BUCS-TR-2010-002, 2010.