Large-Scale VM Placement with Disk Anti-Colocation Constraints Using Hierarchical Decomposition and Mixed Integer Programming

Ye Xia, Mauricio Tsugawa, Jose A. B. Fortes, and Shigang Chen, *Fellow, IEEE*

Abstract—As computational clouds offer increasingly sophisticated services, there is a dramatic increase in the variety and complexity of virtual machine (VM) placement problems. In this paper, we consider a VM placement problem with a special type of anti-colocation requirements—disk anti-colocation—which stipulate that, for every VM assigned to a PM (physical machine), its virtual disks should be spread out across the physical disks of the PM. Once such a requirement is met, the users of the VM can expect improved disk I/O performance. There will also be improvement in fault tolerance and availability. For scalable solutions, we propose a method that combines hierarchical decomposition with mixed integer programming (MIP), where the basic building blocks are independent, small MIP subproblems. We provide experimental results to demonstrate the effectiveness of the proposed method. We show that it is scalable and achieves high performance with respect to the optimization objective.

Index Terms—Cloud computing, datacenter, virtual machine placement, resource management, mixed integer programming

1 Introduction

Over the last ten years computational clouds have become widely used by large and small enterprises as their most cost-effective means to deploy IT services. The value proposition offered by computational clouds has evolved to go beyond cost-effective on-demand hosting of IT resources and elasticity. The added value now includes the ability to offer entire IT systems as a service that can quickly adapt to changing business environments (versus being statically configured) and are automatically optimized in response to changes in either the environment or the workload [1], [2], [3]. These new capabilities-agility and continuous optimization—put pressure on the resource management component of the cloud. There is an urgent need of effective resource management that can enable those new capabilities while minimizing datacenter costs. Ineffective resource management will result in wasted datacenter resources and excessive energy consumption, thus significantly reducing the return on investment [4].

One of the important datacenter resource management problems is virtual machine (VM) placement [5], [6], [7], which is to assign a set of VMs requested by customers to the physical machines (PMs) in the datacenters so that certain cost, profit or performance objective is optimized, subject to the PMs' resource capacity constraints and possibly network bandwidth constraints. As clouds offer increasingly sophisticated services with the intended agility and performance, there is a dramatic increase in the variety and complexity of VM placement problems. Part of the reason is that system deployment requested by customers may contain complex relationships among the system components, such as resource grouping and hierarchy, various colocation or anti-colocation constraints, topological relationships, and workflow dependencies. In particular, services that contain *anti-colocation* requirements have the generic form that a set of requested resources should not be colocated in a sense that depends on the precise specification. For instance, to improve the availability of its service, a customer may require some of its VMs not to be placed on the same physical server or the same server rack [3]. The anti-colocation requirements have not been adequately addressed in the placement solutions of any prior work.

This paper focuses on a special type of anti-colocation requirements—disk anti-colocation. Most VM types offered by public clouds such as Amazon EC2 [8] have multiple virtual disks per VM. When a customer requests such a VM, he may be interested in the following disk anti-colocation requirement: No physical disk of the PM to which the VM is assigned should contain more than one of the VM's virtual disks. That is, the VM's virtual disks should be spread out across the physical disks of the PM. There are important use cases where disk anti-colocation is desirable, which we will elaborate next.

Cloud users often care a great deal about disk IO performance, as evidenced by many Internet discussions [9], [10], [11]. Since directly attached storage (DAS)—including local disks—has numerous advantages over network-based storage, such as higher IO throughput, lower latency, more predicable IO performance (less storage sharing), lower cost

- *Y. Xia and S. Chen are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611. E-mail: {yx1, sgchen}@cise.ufl.edu.*
- *M. Tsugawa and J.A.B. Fortes are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611. E-mail: {tsugawa, fortes}@acis.ufl.edu.*

Manuscript received 14 Mar. 2016; revised 25 Aug. 2016; accepted 3 Oct. 2016. Date of publication 7 Oct. 2016; date of current version 12 Apr. 2017. Recommended for acceptance by B. He.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2615933

and lower complexity [9], [12], [13], they are the preferred storage option for many high-valued, critical applications. These include NoSQL databases like Cassandra and MongoDB, online transaction processing systems, massively parallel processing (MPP) data warehousing, Hadoop/MapReduce storage nodes, log or data-processing applications, etc. [9], [10], [13], [14].

However, even with DAS, the disk IO can still be a performance bottleneck. This can happen even for a VM that is assigned with a dedicated PM—in particular, if the requested virtual disks are all mapped to a single physical disk and the IO capacity of that disk is exceeded. Many PMs have multiple, up to dozens of, directly attached physical disks with very high combined IO capacity [14]. With disk anti-colocation, the disk IO bottleneck may be avoidable. Amazon and industry specialists recommend that multiple virtual disks can be grouped together into a RAID configuration to increase aggregate IO throughput or improve fault tolerance [12], [13]. However, the RAID configuration can only be fully effective if the virtual disks are mapped to different physical disks. Putting the requested virtual disks onto the same physical disk makes little sense, as it does not exploit the parallelism and redundancy provided by multiple physical disks. Disk anti-colocation is also important in another scenario that improves reliability without RAID. Hadoop handles reliability in software through data duplication across servers and it can allow individual volumes to fail before the node goes down [15]. However, for a Hadoop cluster in a virtualized environment, if all the virtual disks of a VM are assigned to the same physical disk, then the failure of that physical disk is more serious than the failure of a single virtual disk. It will take out all the virtual disks of the VM, rendering the VM useless, and it will be more difficult for the cluster to repair the damage.

To solve our VM placement problem with disk anti-colocation, we advocate the use of mixed integer programming (MIP) [16] formulations and algorithms. The MIP approach should complement other approaches frequently used for datacenter resource management, including specialized combinatorial algorithms and heuristic algorithms. The rationales are the following. First, VM placement problems are different for different customers; for the same customer, the problems can change constantly since the resource demand and performance goal can vary significantly over time. We have to expect that our problem is not a single static one, but a representative of a class of dynamic problems. For continuous optimization, the resource management decisions need be continuously carried out. Hence, it is not possible to design specialized algorithms for all these problem variants in time. Second, to support the envisioned datacenter capabilities, some of the problem variants can be very complex and it is not an easy task to design good heuristic or specialized algorithms. MIP-based formulations are general and flexible in capturing all sorts of constraints and objectives, as is well-known in related fields involving large-scale, complex resource management or scheduling [17], [18], [19]. Once a problem is formulated, the standard MIP algorithms can be applied directly, thus minimizing the algorithm development time. There is no need to craft different specialized algorithms for different variants of the problem. Furthermore, since the MIP algorithms look for optimal solutions, they will out-perform heuristic solutions in terms of the achieved objective values. The performance gain can be significant (see experimental results in Section 4).

The main challenge for the MIP approach is that the MIP algorithms have long computation time for large problem instances. Typical strategies to cope with that challenge include finding better MIP algorithms, using more powerful computers to run the algorithms, and developing clever problem formulations. Even with all the above, MIP algorithms can only solve what might be considered small to medium problem instances in our application setting, good enough for several server clusters in a datacenter. To model problems for a large datacenter in its entirety, an MIP formulation may involve billions of variables and/or constraints, and there is no hope to solve them optimally within acceptable time. In such cases, we observe that a hierarchical decomposition method can be used to break a large problem into many independent subproblems, which can be solved in parallel by separate management servers. Each of the subproblems will be sufficiently small and solvable quickly using MIP algorithms.

In this paper, we will explore such hierarchical decomposition approach within the MIP framework on our VM placement problem. The goal is to solve large instances of the problem suitable for large datacenters. We next summarize the main contributions of the paper.

1) We identify and investigate a difficult and practically useful class of datacenter resource management problems that involve disk anti-colocation requirements. We will see that the problem is very difficult, harder than typical VM placement problems with colocation/anti-colocation constraints among the VMs (as apposed to among the disks). To the best of our knowledge, there have been no systematic studies about such a problem. There are no known specialized algorithms or heuristic algorithms that are both fast and yield high-quality solutions.
2) For scalable solutions, we propose a method that combines hierarchical decomposition with MIP, where the basic building blocks are independent, small MIP subproblems. Prior studies on datacenter resource management generally do not use MIP formulations and algorithms. There exist several prior studies that propose hierarchical resource management [20], [21], [22]. However, they are about different problems and for different purposes, and they are not combined with MIP.
3) We provide experimental results to demonstrate the effectiveness of the proposed method. We show that it is scalable and achieves high performance with respect to the optimization objective. The results should be interesting to researchers who wish to know the effectiveness and scalability of MIP-based algorithms for VM placement problems.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we present an MIP formulation for our VM placement problem with disk anti-colocation requirements. We describe our solution approach, which combines hierarchical decomposition and MIP. In Section 4, we present experimental results to

demonstrate the effectiveness of the approach. In Section 5, we draw conclusions and discuss additional issues.

2 Related Work

Prior studies on VM placement generally avoid MIP formulations all together. In the cases where MIP formulations are used, they are usually used to describe the problem; the algorithms are not based on MIP. Instead, the effort is usually on developing specialized combinatorial algorithms such as multi-dimensional bin-packing [23], [24], graph algorithms [6] or sophisticated heuristics [25]. Those algorithms are tailored to the special problems that the authors study, usually relying on certain structures of the problems. In our assessment, they cannot be adapted easily to our problem, due to the addition of the disk anti-colocation requirements, which pose constraints of a different kind. Packing is still there in our problem with respect to the vCPU and memory resources. However, at the disk level, there is tension between packing and anti-colocation; packing too tightly may make it difficult to satisfy the disk anti-colocation requirements.

Practical cloud systems usually adopt less sophisticated heuristics, such as round-robin, first-fit or first-fit-decrease, as evidenced by open-source middleware stacks [26], [27], [28]. While simple heuristics may find solutions quickly, they can also be underachieving in terms of performance. In particular, when a problem is sufficiently complex or have difficult constraints, intuitions that are needed to develop sound heuristics may fail. The anti-colocation constraints in our problem are difficult, since disk assignments are intertwined with VM assignments. It is not easy to design a heuristic algorithm that always has good performance (see Section 4.4 for more discussion).

We propose to stay with MIP as far as possible. To find solutions for large problem instances, we propose to use a hierarchical decomposition method to replace a flat, large MIP problem by many much smaller problems, each of which is far easier to solve and can be solved in parallel by separate management servers. The authors of [20] also propose an idea of hierarchical decomposition on a VM placement problem with the objective of reducing datacenter network traffic (see also [25]). However, they do not use MIP algorithms, but develop a specialized algorithm tailored for their particular problem. Thus, their focus is quite different from ours.

Our earlier work [29] (published as work-in-progress) presents a hierarchical architecture for general datacenter resource management problems. There, the main contributions are the motivations and design of the architecture, which provides a flexible way of grouping various resource requirements from the customers. The subject of this paper is to solve a particular VM placement problem with disk anti-colocation constraints. In the process, a simple version of that architecture is used. But, the focus here is the problem itself, the design of the two-level assignment subproblems, and detailed computational experiences for solving that problem. The idea of grouping the resource requirements and grouping hardware resources has also been explored in [21], [22]. However, those studies do not overlap with our VM placement problem. Nor do they explore the MIP approach.

We next review several marginally related recent studies. The authors of [30] consider a VM placement problem under traffic demand variations in time. They formulate the problem as to minimize the maximum network cut load while satisfying other resource constraints. They develop two heuristic placement algorithms with different tradeoffs between solution quality and complexity. The authors of [31] present a VM placement problem that takes into account the time-varying resource requirements from the VMs, with the objective of reducing the number of PMs used. A greedy heuristic algorithm is proposed that places the VMs sequentially. Each VM is placed into the PM that will become the "most fully utilized" after the VM's placement, where the utilization is averaged over time and over different resource types. The authors of [32] formulate a combinatorial optimization problem of joint VM placement and routing in datacenters. They develop a heuristic randomized algorithm based on the Markov chain Monte Carlo method. In [33], a more general version of routing and VM placement problem is formulated and solved with an online randomized algorithm. The authors of [34] develop a stochastic control algorithm for online VM placement under VM arrival and departure dynamics. The authors of [20] study a VM placement problem with the objective of reducing datacenter network traffic (see also [25]). They develop a specialized hierarchical algorithm based on the problem structure. The work in [3] describes how datacenters can offer complex services to the cloud customers, such as entire IT as a service with VM colocation and anti-colocation requirements. It does not have a clearly formulated optimization problem. For resource placement, it takes an incremental and randomization approach with immediate random placement of the requested resources at the time of the request arrivals, followed by subsequent gradual adjustment. Overall, the above studies are different from ours in both the problems and solution approaches.

There is also a stream of literature on short-term dynamic resource re-allocation based on workload monitoring, prediction and adaptive control [35], [36], [37], [38], [39], [40]. The proposals in that literature typically are restricted to (1) managing much smaller systems, e.g., a single PM with multiple VMs or a small-scale server cluster, and (2) having limited problem complexity, e.g., no anti-colocation constraints. Such restrictions make the resource allocation problems much simpler and thereby allow algorithms to reallocate resources more frequently based on more recent workload information. Aspects of the ideas and algorithms in that stream of literature may be added to our framework to act on a shorter timescale.

3 VM Placement with Disk Anti-Colocation

We will show an MIP formulation of the VM placement problem with disk anti-colocation requirements. It illustrates that the MIP formulation is capable of describing complex constraints. However, we will see that even for a basic version of this problem, the MIP formulation has an exceedingly large number of variables and constraints. Standard MIP algorithms cannot solve large problem instances within acceptable time. One can imagine that more complicated versions will be more difficult solve. We will provide a scalable algorithm which combines hierarchical decomposition and MIP.

3.1 Problem Formulation

We consider a problem of assigning N VMs to M PMs with disk anti-colocation requirements. Here, N and M can both be fairly large, e.g., thousands or more. Each VM has the following resource requirements: memory, number of vCPUs, number of local disk volumes (virtual ones) and their respective sizes.

Each PM has certain memory capacity, number of vCPUs that it can support, and number of local disks and their respective sizes. These local disks may be in the PM or directly attached (either by local disk interface or fast, dedicated network interface such as fiber channels). Recall that the physical disks that we are considering are high-performance disks, e.g., local or directly attached SSDs (denoted as lssd). They can be quite expensive, not in great abundance, and therefore, need to be managed for efficient use.

3.1.1 Constraints

We first give an overview of the constraints for our problem.

- There are the usual capacity constraints for each resource: With respect to each resource (vCPUs, memory), the total amount of resource required by all the VMs assigned to any PM cannot exceed the resource capacity of the PM.
- The next set of constraints is quite special, which make our problem different from the usual VM placement problems. When a requested VM i has multiple virtual disks, there is a disk *anti-colocation constraint*: No physical disk of the PM (to which VM i is assigned) can contain more than one of VM i's requested virtual disks. The motivations for such a constraint have been given in Section 1.
- A final set of constraints is that the capacity of each physical disk must be no less than the aggregate size of all virtual disks assigned to it.

We next give the details. Let the sets of VMs and PMs be denoted by $\mathcal{V}$ and $\mathcal{P}$, respectively. Without loss of generality, let $\mathcal{V} = \{1, 2, \ldots, N\}$ and $\mathcal{P} = \{1, 2, \ldots, M\}$. For each VM i, let α_i be the number of vCPUs required and let β_i be the memory requirement (in GiB).[1] For each VM i, a set of virtual disks is requested and the set is denoted by $R_i = \{1, \ldots, |R_i|\}$. For each of the requested virtual disks $k \in R_i$, let ν_{ik} be the requested disk volume size (in GB).

For each PM j, let C_j be the number of vCPUs it can support, M_j be the amount of memory (in GiB), and $D_j = \{1, \ldots, |D_j|\}$ be the set of available physical disks. The sizes of the physical disks are denoted by S_{jl} (GB) for $l \in D_j$.

For each $i \in \mathcal{V}$ and each $j \in \mathcal{P}$, let x_{ij} be the binary assignment variable, which takes the value 1 if VM i is assigned to PM j and 0 otherwise. The binary variables y_{ikjl} are used for disk assignment: y_{ikjl} is set to 1 if VM i is assigned to PM j and the requested virtual disk k, where $k \in R_i$, for VM i is assigned to the physical disk l of PM j, where $l \in D_j$; it is set to 0 otherwise. The following constraints are required:

$$y_{ikjl} \leq x_{ij}, \quad i \in \mathcal{V}, j \in \mathcal{P}, k \in R_i, l \in D_j \tag{1}$$

$$\sum_{j \in \mathcal{P}} \sum_{l \in D_j} y_{ikjl} = 1, \quad i \in \mathcal{V}, k \in R_i \tag{2}$$

$$\sum_{j \in \mathcal{P}} x_{ij} = 1, \quad i \in \mathcal{V} \tag{3}$$

$$\sum_{k \in R_i} y_{ikjl} \leq 1, \quad i \in \mathcal{V}, j \in \mathcal{P}, l \in D_j \tag{4}$$

$$\sum_{i \in \mathcal{V}} \sum_{k \in R_i} \nu_{ik} y_{ikjl} \leq S_{jl}, \quad j \in \mathcal{P}, l \in D_j. \tag{5}$$

$$\sum_{i \in \mathcal{V}} \alpha_i x_{ij} \leq C_j, \quad j \in \mathcal{P} \tag{6}$$

$$\sum_{i \in \mathcal{V}} \beta_i x_{ij} \leq M_j, \quad j \in \mathcal{P}. \tag{7}$$

The following explains some of the constraints:

- (1) ensures that the requested virtual disks for VM i may be assigned to the physical disks of PM j only if VM i is assigned to PM j.
- (2) ensures that every requested virtual disk must be assigned to exactly one physical disk.
- (3) ensures that every VM is assigned to exactly one PM.
- (4) ensures that VM i cannot have more than one of its virtual disks assigned to the same physical disk; (1) and (4) together enforce the disk anti-colocation constraints.
- (5) is the disk capacity constraint.
- (6) and (7) are the resource capacity constraints posed by the number of vCPUs and the total memory size of each PM j.

We will check some subtler implications. Suppose $y_{ikjl} = 1$. By (1), we have $x_{ij} = 1$; hence, VM i must be assigned to PM j. Suppose VM i requests more than one virtual disk. Suppose $k' \neq k$. Then, the virtual disk k' cannot be assigned to a PM other than j; otherwise, $x_{ij'} = 1$ for some $j' \neq j$, which violates (3). Furthermore, by (4), k' must be assigned to one of PM j's disks other than l.

Remark. VM placement problems encountered in practice will likely contain many different performance-cost considerations. One of the key points is that MIP formulations can capture subtle or complex requirements quite easily, such as the disk anti-colocation requirements, VM/storage colocation and anti-colocation constraints, resource grouping and hierarchy, other topological constraints, network constraints, and workflow dependencies. Multiple performance or cost objectives can be handled either by optimizing a linear combination of the objectives or by selecting one objective to optimize and putting the rest as constraints. Practical VM placement problems may contain several sets of difficult constraints, each of which may require a different technique to cope with. Our problem formulation contains one such difficult component, disk anti-colocation, and we will provide one set of techniques to address it. The research community has examined several other difficult components, such as network constraints. The techniques in this

1. 1 GiB (gibibyte) is equal to 2^{30} bytes, which is 1,073,741,824 bytes; 1 GB (gigabyte) is equal to 10^9 bytes.

paper may serve as a building block for a complete solution to practical problems.

3.1.2 Costs and Optimization Objective

The optimization objective will ultimately be decided by the cloud provider. For concreteness, we assume that a fixed operation cost is incurred for a PM as long as the PM is used by some VMs (that is, some VMs are assigned to the PM). Specifically, when a PM j is turned on to serve some VMs, there is a fixed cost $\hat{c}_j$ associated with running the PM; when the PM is off, there is zero cost. The operation cost may include the average energy cost when a machine is running and typical maintenance cost. The optimization objective is to minimize the total operation cost.

Let z_j be a 0-1 variable indicating whether PM j is used by some VMs. To ensure that $z_j = 1$ if and only if $x_{ij} = 1$ for some $i \in \mathcal{V}$, we add the following two constraints, where B is a large enough constant (it is enough to take $B = N$)

$$z_j \leq \sum_{i \in \mathcal{V}} x_{ij}, \quad j \in \mathcal{P} \tag{8}$$

$$Bz_j \geq \sum_{i \in \mathcal{V}} x_{ij}, \quad j \in \mathcal{P}. \tag{9}$$

In normal situations, an assignment should be feasible so that no VMs will be rejected; otherwise, the cloud provider will usually increase its datacenter capacity. When no VMs are rejected, the total payment by the customers is fixed. In that case, a sensible optimization objective is to minimize the total operation cost, which leads to profit maximization. Thus, our optimization objective is the following:

$$\min_{x,y,z} \sum_{j \in \mathcal{P}} \hat{c}_j z_j. \tag{10}$$

3.1.3 Possible Extensions of the Model

The model can be enriched in many ways. Variants and refinements are the subject of ongoing work. With respect to the optimization objective, we can consider maximizing the profit under complex revenue and cost structures. On the revenue side, a customer's payment may depend on the received performance level of his workload, on the types of PMs his VMs are placed at, or on the degree of isolation of his VMs from other customers' VMs. The costs may include load-dependent costs (e.g., the energy cost is higher for higher CPU load) and the costs of other equipments such as networking devices. Multiple objectives from both the provider and the customers (e.g., thermal dissipation, customers' performance objectives) can be incorporated into the formulation by either forming a weighted sum of all the objectives or by treating all but one of the objectives as constraints (see [5] for a related treatment).

The model can clearly be extended to include network storage of various types. It can also be extended to include local and network bandwidth constraints. Those additional constraints depend on the customers' needs and the cloud provider's policies, and they vary across customers/providers and change over time. Given the absence of the details, we do not include those additional constraints in this paper. We expect that disk anti-colocation is a class of distinct constraints. It is worthwhile to single it out for a focused investigation.

3.1.4 Problem Complexity

As is typical for MIP problems, we will measure the complexity of the problem by the number of variables and the number of constraints, which can be counted easily. For simplicity, suppose each VM requests R virtual disks and each PM provides D physical disks. Then, the number of y variables is $N \times M \times R \times D$ and the number of x variables is $N \times M$. The number of constraints of the form in (1) is $N \times M \times R \times D$. To make the matter more concrete, suppose 1,000 VMs are to be assigned to 1,000 PMs, i.e., $N = 1{,}000$ and $M = 1{,}000$. Suppose each VM requests $R = 2$ virtual disks and each PM provides $D = 4$ physical disks. Then, the number of y variables is $1{,}000 \times 1{,}000 \times 2 \times 4 = 8{,}000{,}000$ and the number of x variables is 1,000,000. The number of constraints of the form in (1) is 8,000,000. This is a very large MIP problem, exceeding what typical MIP software can handle. A large datacenter may have 1,000,000 PMs servicing more than 1,000,000 VMs. Even if at each assignment instance, only 10 percent of them participate in the assignment, the number of y variables exceeds $100{,}000 \times 100{,}000 \times 2 \times 4 = 8 \times 10^{10}$ and the number of constraints also exceeds that number. Thus, a different kind of algorithm is needed for large problem instances. The examples also show that the disk anti-colocation requirements are the main source of difficulty.

3.2 Two-Level Hierarchical Decomposition Algorithm

For large problem instances, we propose a hiearchical decomposition algorithm to break a large problem into many small independent MIP subproblems that can be solved in parallel by different management servers. The outline of our decomposition algorithm is given in Algorithm 1, which may be viewed as a framework capturing a class of algorithms. The choice for the first-level assignment problem is one of the major factors that distinguish different algorithms within that framework. In the subsequent detailed explanation, we will describe our choice.

Algorithm 1. Outline of Two-Level Decomposition Algorithm

1: divide the VMs into K_v packs; divide the PMs into K_p swads
2: assign the packs to the swads by solving an MIP problem # *first-level assignment*
3: **for** each swad with assigned packs **do**
4: perform a VM-to-PM assignment by solving the MIP problem in Section 3.1 # *a second-level assignment*
5: **end for**

3.2.1 Divisions into Packs and Swads

The algorithm first divides the VMs into K_v groups and the PMs into K_p groups (line 1). Using the terminologies introduced in [29], a VM group is called a *pack*; a PM group is called a *swad*. The parameters K_v and K_p should be chosen such that (i) each of the swads should have plenty of resources to accommodate at least one pack, and (ii) the sizes of

the packs and swads are not too large (each having less than several hundred VMs or PMs, respectively) so that each second-level assignment problem can be solved quickly. Conditions (i) and (ii) are basic, necessary conditions for the two-level decomposition algorithm to work.

There are many ways to make the divisions. For concreteness, we recommend evenly and randomly divide the VMs into packs and the PMs into swads. That is, each pack is formed by selecting N/K_v VMs randomly and each swad is formed by selecting M/K_p PMs randomly. This is the strategy we took for our experiments. Each pack or swad should be sized so that each second-level assignment problem can be solved quickly, for instance, 20-100 VMs per pack and 20-100 PMs per swad. Our computational experiences have shown that this recommended approach works well (see Section 4.4). See Sections 3.2.4, 4.3.3 and 4.4.4 for more comments. If there are other constraints or requirements that lead to packs/swads of different sizes, for the purpose of solving our VM placement problem, we can still make *logical* packs/swads of the sizes that we are considering here, and add extra constraints to represent the needed relationship (e.g., colocation) between some logical packs or between some logical swads.

3.2.2 First-Level Pack-to-Swad Assignment (Line 2)

In the first level of assignment, we assign the K_v packs to the K_p swads by solving an MIP problem described below. For each swad, we aggregate the total number of vCPUs, the total amount of memory, the total number of disks (lssd) and the total amount of disk storage space over all the PMs in the swad; we also record the maximum number of vCPUs, the maximum amount of memory, and the maximum number of disks of any PM in the swad. For each pack, we also tabulate the same requested quantities. In particular, the maximum amount of a resource requested by a pack is defined to be the maximum amount requested by any VM in the pack.

Then, we solve an MIP problem that minimizes the number of swads used, subject to the following constraints with respect to the vCPU, memory and disk space resources: For each swad, (i) the total usage of a resource is no greater than the total capacity of the same resource in the swad; (ii) the maximum amount of a resource requested by any pack assigned to the swad is no more than the maximum capacity of the same resource provided by any PM in the swad.

For instance, for each swad j, let C_j^{tot} be the total number of vCPUs of all the PMs in the swad, and let C_j^{max} be the maximum number of vCPUs of any PM in the swad. For each pack i, let α_i^{tot} be the total number of vCPUs requested by all the VMs in the pack, and let α_i^{max} be the maximum number of vCPUs requested by any VM in the pack. Let $\hat{x}_{ij}$ be a 0-1 assignment variable that takes the value 1 if and only if pack i is assigned to swad j. Then, the constraints with respect to the vCPU resource are

$$\sum_{i=1}^{K_v} \alpha_i^{tot} \hat{x}_{ij} \le C_j^{tot}, \quad j = 1, \ldots, K_p \tag{11}$$

$$\max_{1 \le i \le K_v} \alpha_i^{max} \hat{x}_{ij} \le C_j^{max}, \quad j = 1, \ldots, K_p. \tag{12}$$

There is a similar set of constraints for the memory resource, and there is another set for the total disk space.

The constraints associated with the requested number of disks are somewhat different. We define a safety margin, $0 < \theta \le 1$, and we require that the total number of disks requested by all the packs assigned to a swad is no more than θ times the total number of disks provided by the swad. The reason for doing so is that the disk anti-colocation constraints can be difficult to satisfy in the second-level assignment problems. By reducing θ in the first-level assignment (if needed), we can spread out the packs more across the swads to gain more room for maneuver. The constraints associated with the maximum number of requested disks are as usual. That is, the maximum of the number of disks requested by any pack assigned to a swad is no more than the maximum number of disks provided by any PM in the swad.

3.2.3 Second-Level VM-to-PM Assignments

A second-level assignment is performed for each of the swads that has some packs assigned to it by the first-level assignment (line 4). For each such swad, we collect all the VMs in all the packs that are assigned to the swad, and we collect all the PMs in the swad. We then find optimal assignments of the VMs to the PMs using the MIP formulation provided in Section 3.1, with the same objective of minimizing the total operation cost of the PMs. For each of the swads, the minimum cost is given by the MIP solution. The cost is equal to zero if no packs are assigned to the swad. The overall cost of the entire decomposition algorithm is the sum of all the minimum costs for all the swads.[2]

3.2.4 Discussion

There is a single MIP problem to solve in the first-level assignment. There are up to K_p MIP problems to solve for the second-level assignments. However, the second-level problems are completely independent from each other. They can be solved by different computing servers in parallel. Thus, the two-level decomposition method is scalable. The first-level assignment problem is far less difficult than each of the second-level problems, since there are no disk anti-colocation constraints in the first level. For instance, we can solve a $1{,}000 \times 1{,}000$ pack-to-swad assignment problem very quickly (see Section 4.4). If further scalability is needed for very large datacenters, the two-level decomposition algorithm can be extended naturally to more levels of decomposition. For instance, the $K_v \times K_p$ pack-to-swad assignment problem can be solved using a two-level decomposition algorithm.

By varying K_v and K_p, there is a trade-off between the computation time and the total achievable cost. As the sizes become larger, the two-level decomposition algorithm will achieve a lower cost, whereas the computation time for the second-level problems will be longer. Conversely, the

2. There is still a possibility that some of the second-level assignments are infeasible for some swads, even after we require the resource usage constraints are satisfied in the aggregate and in the maxima when we conduct the first-level pack-to-swad assignment. One remedy is to put safety margins on all the stringent resources (like the use of θ) and find suitable values for the margins by binary search. When the constraints are violated by small amounts, one can simply move some of the VMs to unused swads. For the experiments that we will present, all the second-level assignments turn out to be feasible.

TABLE 1
VM Types

VM Type	vCPU	Memory (GiB)	Storage (all SSD; GB)
m3.medium	1	3.75	1×4
m3.large	2	7.5	1×32
m3.xlarge	4	15	2×40
m3.2xlarge	8	30	2×80
c3.large	2	3.75	2×16
c3.xlarge	4	7.5	2×40
c3.2xlarge	8	15	2×80
c3.4xlarge	16	30	2×160
c3.8xlarge	32	60	2×320
r3.large	2	15.25	1×32
r3.xlarge	4	30.5	1×80
r3.2xlarge	8	61	1×160
r3.4xlarge	16	122	1×320
r3.8xlarge	32	244	2×320
i2.xlarge	4	30.5	1×800
i2.2xlarge	8	61	2×800
i2.4xlarge	16	122	4×800
i2.8xlarge	32	244	8×800

smaller the pack/swad sizes are, the shorter the computation time is; however, the achievable cost will be higher, because the boundaries between the swads or packs put extra restrictions on where each VM can be placed. Also, for a given swad size, there is a maximum pack size beyond which the first-level assignment problem becomes infeasible. To get substantial cost reduction and acceptable computation time, the pack size should not be too small either. For our problem, computational experiences have shown that it works well when K_v and K_p are comparable in values, taken in the range from 20 to 100.

4 Solution and Simulation Results

This section presents experimental results to demonstrate the effectiveness of the proposed two-level decomposition algorithm in terms of scalability and performance improvement. The proposed algorithm is compared with an aggressive heuristic algorithm.

4.1 Setup

We follow the VM and PM setup in Amazon's EC2 as close as we can [8]. We take a subset of the allowed VM types (classes) of Amazon's EC2. Their resource requirements are shown in Table 1. We use a subset of the PM types in Amazon EC2, which are listed in the first column of Table 2. Cloud providers generally don't disclose the detailed capabilities of all their PMs. The amount of resources that each type of PMs is equipped with (shown in Table 2) is largely our guess based on the information revealed on Amazon's web site. Given the diversity of physical hardware that vendors offer, the amount of resources listed in Table 2 can also be understood as a plausible sample (see the remark below). The operation costs (in the 5th column) are not exactly known, but are based on our estimate.[3] The costs are normalized with the lowest operation cost chosen to be 100.

3. The large cost increase when the number of disks exceeds 4 reflects the cost of running separate DAS (directed attached storage) devices.

TABLE 2
PM Types

PM Type	vCPU	Memory (GiB)	Storage (all SSD; GB)	Operation Costs (normalized)
s1	8	16	1×256	100
s2	8	32	1×512	120
s3	8	64	2×512	200
s4	8	64	4×512	300
m1	16	32	2×512	600
m2	16	64	4×512	700
m3	16	128	$4 \times 1{,}000$	900
m4	16	256	$8 \times 1{,}000$	1,500
m5	16	256	16×512	1,800
l1	32	256	$4 \times 1{,}000$	2,500
l2	48	512	$8 \times 1{,}000$	3,500
l3	64	1,024	$4 \times 1{,}000$	5,000
l4	80	2,048	$16 \times 1{,}600$	7,000
l5	120	4,096	$4 \times 1{,}000$	9,000
l6	120	4,096	$24 \times 1{,}600$	12,000

Since the problem is linear, it doesn't matter what the chosen normalization base cost is. If the base cost is chosen to be κ instead of 100, the optimal cost is simply $\kappa/100$ times of the optimal cost under the base cost 100.

Remark. There is a diverse array of server and storage systems for datacenters in the market. Different cloud providers use different systems, ranging from simple, low-cost servers to specially-designed systems for datacenters, and they may use a combination of them. A low-cost server may be only slightly more powerful than a high-end consumer machine, such as a 4-core, single processor machine, with 16 GiB or memory and 1512-GB SSD drive (and 1 Gbps networking interface). On the other end of the spectrum, a Cisco UCS B460 M4 server has four Intel Xeon E7-8800 v2 processors, which provides 60 processor cores, 96 DDR3 memory DIMM slots (total 6.0 TiB of memory using 64-GiB memory modules), four drive bays for hard disks or SSDs (up to 4.8 TB of internal storage), and 320 Gbps of overall Ethernet throughput. The number of disk drives that can be installed inside a server is usually quite limited, e.g., up to 4. However, servers can be attached to disk arrays using some form of "direct" connections, such as the SAS interface or direct fiber channel connections. Such directly attached storage (DAS) can provide storage-access performance at the level of internal disks. Thus, DAS can be considered as a form of local storage. With DAS, each server can have 16, 24, 128 local disks, depending on the system setup and cost.

For Amazon EC2, each vCPU corresponds to a hyperthread of a physical core [41]. In our experiments, we assume the PMs all support two hyperthreads per physical core. Hence, each physical core counts as 2 vCPUs. As an example, Amazon EC2 uses Intel Xeon E5-2680 processors for the c3 class of VMs. Each Xeon E5-2680 processor has 8 cores and supports a total of 16 threads. A PM with one such processor offers 16 vCPUs. As another example, the aforementioned Cisco UCS B460 M4 server offers 120 vCPUs.

All the experiments were done on a machine with 2 cores at 2.13 GHz and 4 GB memory. One can expect a speedup by a factor of 5-10 if a more powerful machine is used.

4.2 Target of Comparison: Randomized Heuristic Algorithm

Since we are not aware of prior studies on exactly our problem (due to the disk anti-colocation requirements), we developed our own heuristic algorithm as a target for performance comparison. In order for it to be a useful benchmark for comparison, we make it aggressive in the sense that it sacrifices speed for better performance in the achievable cost. By doing so, its cost performance can serve as a lower bound for a class of other possible heuristic algorithms that put speed at a higher priority.

Our heuristic algorithm is motivated by the general ideas of online randomized algorithms [3], [42], [43] but should achieve much lower costs than the latter due to two exhaustive search steps. The algorithm is summarized in Algorithm 2.

Algorithm 2. Randomized Heuristic Algorithm

```
1: permute the VM_list
2: for each VM in VM_list do
3:    scan used_PM_list to find a PM that can accommodate
      the VM
4:    if such a PM is found then
5:       assign the VM to the PM
6:    else
7:       scan unused_PM_list to find a PM that can
         accommodate the VM
8:       if such a PM is found then
9:          assign the VM to the PM; move the PM to
            used_PM_list
10:      else
11:         exit    # the problem is infeasible
12:      end if
13:   end if
14: end for
```

The algorithm first randomly permutes the list of all the requested VMs; this emulates the random arrival order of the VM requests. For each VM in the permuted list, an attempt is made to assign the VM to a PM. The PMs are organized into two lists—the list of *used* PMs, which are those PMs already with some assigned VMs, and the list of *unused* PMs.

To place a VM, the list of used PMs is searched exhaustively first. If no PM in the used list can accommodate the VM, then the list of unused PMs is checked.[4] Note that, in our model, a PM incurs a constant operation cost regardless of how many VMs are assigned to it. The algorithm's greedy behavior of checking used PMs first matches the assumption of constant operation costs.[5] Before scanning a PM list, the PM list is randomly permuted to add some load-balancing aspect. The first PM in the list that can accommodate the VM is selected. This aspect is known as the the first-fit heuristic, which is an often used heuristic [3], [42], [43].

For each scanned PM (in lines 3 and 7), our heuristic algorithm checks whether it is possible to assign the currently considered VM to that PM. For vCPU or memory, it is enough to check whether the remaining number of vCPUs or the remaining memory is sufficient for the VM. For disk assignment, the algorithm exhaustively enumerates different disk assignment possibilities and uses the first one that is feasible. As discussed earlier, the disk anti-colocation constraints are the difficult ones to satisfy. Our heuristic algorithm is aggressive in finding a feasible disk assignment. Other practical heuristic algorithms are unlikely to conduct an exhaustive search, since the number of possible disk assignments can sometimes be very large. With exhaustive search, our algorithm will achieve a lower total operation cost.

Remark. One can consider several other alternative algorithms. There is a popular heuristic algorithm known as first-fit-decrease. The VMs are sorted in a decreasing order of the resource requirements. For each VM in the list, the PM list is searched. A VM is placed in the first PM that can accommodate it. Such an algorithm makes sense if the objective is to be able to place all the VMs. On the other hand, if the number of PMs is abundant and the objective is to minimize some cost, then the sorting of the VMs is not necessarily helpful. The results depend on the order in which the PM list is scanned. Another heuristic idea is to sort each PM list in an increasing order of the operation costs and scan the PMs according to that order instead of a random order. This way, cheaper PMs are used with a higher priority. Whether such sorting is beneficial depends on the details of the problem instance, i.e., all the parameters, such as the costs, the resource configurations of the PMs and the requested configurations of the VMs.

4.3 Comparison between Flat Optimization and Randomized Heuristics

In this part of the experiments, we will show problem instances that can be solved by directly using the MIP solver Gurobi [44] *without* hierarchical decomposition, an approach that we call *flat optimization*. We will compare flat optimization with our randomized heuristic algorithm. The intention is to demonstrate that (1) the optimization approach achieves much lower costs than sophisticated heuristics; (2) however, the size of solvable instances is rather limited. Later in Section 4.4, we will show how hierarchical decomposition can help to solve large instances while maintaining the advantage over the heuristic algorithm in the achievable cost.

The results for experiments 1 and 2 are summarized in Table 3.[6]

4.3.1 Experiment 1-70 VMs and 50 PMs

We experimented with a problem of assigning 70 VMs to 50 PMs. The 70 VMs are of the following mix of types—m3.

4. For a large datacenter, a scalable online algorithm cannot afford to search through all the used PMs or unused PMs for each VM request. A typical strategy is to sample a few used PMs and, if that does not work out, pick a unused PM with sufficient resources randomly. Our heuristic algorithm should do better in the achievable objective value. A more sophisticated algorithm is to keep track of an ordered list of all the PMs according to certain criterion and assign the VM to the first one on the list that fits. In that case, exhaustive search is needed and scalability is limited, as in our heuristic algorithm.

5. The discussion also suggests that, if the cost model is different, the current heuristic algorithm may not work well. For instance, if the operation costs of the PMs depend on the load placed on the PMs in some complicated way, then trying to pack VMs to used PMs first may not be the right approach.

6. Since the heuristic algorithm is not sufficiently scalable, there is no need to collect the computation time. See later comment in Section 4.4.1.

TABLE 3
Summary of Results: Flat Optimization versus Heuristics

Experiments		Flat Optimization	Heuristics
1	Cost	4,540	9,913
	Run Time (s)	106	
2	Cost	45,300	65,105
	Run Time (s)	3,756	

TABLE 5
Flat Optimization Computation Time

Num. of VMs	Num. of PMs	Average Run Time (seconds)
20	20	7.8
40	40	75
70	50	106
77	70	3,756
90	75	4,885

medium: 36; m3.large: 14; m3.xlarge: 10; m3.2xlarge: 10. The 50 PMs are of the following mix—s1: 7; s2: 7; s3: 10; s4: 7; m1: 5; m2: 5; m3: 5; m4: 2; m5: 2. Judging by the VM and PM numbers, this is a small instance. However, the MIP formulation involves 17,950 binary variables and 26,120 constraints, which make it non-trivial for any MIP software. The instance is solved by Gurobi in 106 seconds, yielding the optimal cost 4,540.

The randomized heuristic algorithm can solve the problem more quickly. Since it involves randomization, we collected the results of 50 runs, which all together took several seconds. The average cost of the 50 runs is 9,913, more than twice the optimal cost. The minimum, maximum and standard deviation of the costs are 6,980, 12,000, and 1,074, respectively. As we argued, when compared with other possible randomized heuristics, our heuristic algorithm is designed to do quite well in terms of the achievable cost. The gaps from the optimal cost are expected to be wider for other randomized heuristic algorithms.

4.3.2 Experiment 2-77 VMs and 70 PMs

In this experiment, 77 VMs are to be assigned to 70 PMs (see Table 4). Although the numbers of VMs and PMs are not so different from the previous instance, the mixes of the VM and PM types are quite different. Here, we have a fuller mix of almost all types of VMs and PMs. The instance has 55,380 binary variables and 80,825 constraints, quite a bit larger than the previous instance. The problem took Gurobi about 3,756 seconds (about 63 minutes) to solve, which is much longer than for the previous instance. The optimal assignment has a cost of 45,300.

TABLE 4
VM and PM Setup

VM Type	No. of VMs	PM Type	No. of PMs
m3.medium	5	s1	5
m3.large	5	s2	5
m3.xlarge	5	s3	5
m3.2xlarge	5	s4	5
c3.large	5	m1	5
c3.xlarge	5	m2	5
c3.2xlarge	5	m3	5
c3.4xlarge	5	m4	5
c3.8xlarge	5	m5	5
r3.large	5	l1	5
r3.xlarge	5	l2	5
r3.2xlarge	5	l3	5
r3.4xlarge	5	l4	5
r3.8xlarge	5	l5	5
		16	0
i2.xlarge	2		
i2.2xlarge	2		
i2.4xlarge	3		
i2.8xlarge	0		

We ran the heuristic algorithm 50 times. The resulting average cost is 65,105 and the standard deviation is 3,683. The minimum and maximum costs are 58,400 and 76,200, respectively. On average, the heuristic algorithm results in 44 percent higher cost than the optimal algorithm. Percentage-wise, the heuristic algorithm is doing better here than for the previous instance. Part of the reason is that, even in the optimal algorithm, 38 out of the 70 PMs are used and there is not a lot of room for cost saving, in terms of percentage of improvement. Nevertheless, the cost saving *in value* by the optimal algorithm is still much more than that for the previous instance.

4.3.3 Additional Results and Comments

Table 5 summarizes the computation time for flat optimization on several other instances with various VM and PM mixes. The computation time depends on all the parameters of a problem instance, including the numbers of VMs and PMs, the resource specifications of different VM and PM types, and the mixes of the types. It is difficult to give a concise characterization of that dependency. But, generally speaking, the standard MIP algorithms cannot solve problems with more than several hundred VMs and PMs in under an hour on ordinary computers. Improvement in the computation speed is possible with the use of more clever problem formulations, customized algorithms, and more powerful computers.

4.4 Main Results: Two-Level Hierarchical Decomposition

The major experiments are to assign 1,000 VMs to 1,000 PMs of different types using our two-level decomposition algorithm.[7] We split the VMs into $K_v = 25$ packs and the PMs into $K_p = 25$ swads randomly. Each pack has 40 VMs and each swad has 40 PMs. The results of the experiments are summarized in Table 6.

4.4.1 Mix 1

The mixes of the VMs and PMs are described in Table 7. The safety margin is chosen to be $\theta = 0.7$. The two-level decomposition algorithm achieves a total cost 82,540. This number

7. Although the numbers of VMs and PMs are chosen to be identical, not all the PMs are used in the results of the experiments, and therefore, VM consolidation still occurs.

TABLE 6
Summary of Results: Two-Level Decomposition versus Heuristics

Experiments		Two-Level Decomp.	Heuristics
Mix 1	Cost	82,540	150,573
	Run Time (s)	1281;75 per swad	
Mix 2	Cost	487,840	601,914
	Run Time (s)	3,366;280.5 per swad	
Mix 1; Smaller Pack/Swad Sizes	Cost	98,040	150,573
	Run Time (s)	202;7.8 per swad	

should be compared with the randomized heuristics, which has an average (over 50 runs) total cost 150,573, a standard deviation 4,951, a minimum cost 140,060 and a maximum cost 165,840. The two-level decomposition algorithm achieves about half the cost as that of the randomized heuristics. We conclude that the cost improvement can be significant. As explained earlier, our target of comparison—the randomized heuristics—is quite aggressive and other randomized heuristics will likely do worse than it.

We next discuss the algorithm running time. A total 17 swads are used after the first-level pack-to-swad assignment. A used swad is assigned 1 to 2 packs. The computation for the first-level assignment takes very little time, on the order of a few seconds, due to the small problem size at this level. In general, the first-level assignment does not pose scalability challenges.

For the second-level VM-to-PM assignments, the total running time is 1,281 seconds, which is the aggregate for 17 different computations for the 17 used swads. The average running time is therefore 75 seconds per swad. Note that the 17 different assignment problems are completely independent and can run in parallel on different computers. There is variability in the running times for different swads, due to different problem sizes and the inherent variability of how the feasibility set is explored by the MIP algorithm.

TABLE 7
1,000 VMs and 1,000 PMs-Mix-1

VM Type	No. of VMs	PM Type	No. of PMs
m3.medium	500	s1	150
m3.large	200	s2	150
m3.xlarge	150	s3	150
m3.2xlarge	150	s4	150
c3.large	0	m1	100
c3.xlarge	0	m2	100
c3.2xlarge	0	m3	100
c3.4xlarge	0	m4	50
c3.8xlarge	0	m5	50
r3.large	0	l1	0
r3.xlarge	0	l2	0
r3.2xlarge	0	l3	0
r3.4xlarge	0	l4	0
r3.8xlarge	0	l5	0
		16	0
i2.xlarge	0		
i2.2xlarge	0		
i2.4xlarge	0		
i2.8xlarge	0		

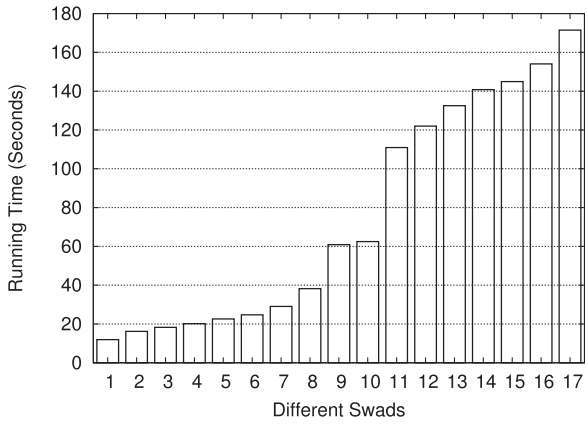


Fig. 1. Sorted running times for VM-to-PM assignments for different swads; mix-1.

The running times are shown in Fig. 1, sorted in increasing order. Overall, we see that the second-level assignments are where the computation complexity lies. To get a solution within a prescribed time budget, the size of each such MIP problems needs to be limited, which can be achieved by controlling the pack/swad sizes.

The heuristic algorithm takes a fairly long time, hundreds of seconds per run. At the minimum, the computation time scales as $O(NM)$, where N is the total number of VMs and M is the total number of PMs. The disk anti-colocation requirements pose greater scalability challenges as the number of disks involved increases. For instance, a type-l6 PM has 24 physical disks and a VM of type i2.8xlarge needs 8 virtual disks. The number of disk assignment possibilities is $\frac{24!}{(24-8)!} = 29{,}654{,}190{,}720$. For that case, the enumeration strategy becomes impractical.[8] The precise situation depends on how disk assignments are implemented in the heuristic algorithm. For instance, one may formulate and solve MIP problems for disk assignments. However, there is a large number of such problems, up to $N \times M$ of them, since there is one whenever a VM is checked against a PM for possible placement. Overall, the heuristic algorithm is not a sufficiently scalable algorithm.[9] But, whenever it works, it should achieve good performance with respect to the optimization objective.

We next make additional comments about the experimental results. Fig. 2 shows the utilization of various resources by the optimal solution for the first-level assignment. The four curves correspond to four different types of resources: vCPU, memory, the number of disks (lssd) and the total disk size. The utilization of the 'number of lssd' is the highest, ranging from 40 percent to close to 70 percent. Given that the safety margin is set at $\theta = 0.7$, we see that the optimal solution tends to saturate that constraint. The next highest utilization is that of the vCPU, ranging from 25 to 50 percent. The total lssd size and the memory are underutilized, at around 10 and 20 percent, respectively. Thus,

8. The disk numbers in our experiments are chosen such that cases with a large number of disk assignment possibilities are avoided.

9. It is possible to make the heuristic algorithm a parallel one by dividing the problem into many smaller ones. Doing so will reduce the computation time but at the expense of yielding a higher operation cost. For the purpose cost comparison, the non-parallel version is more appropriate.

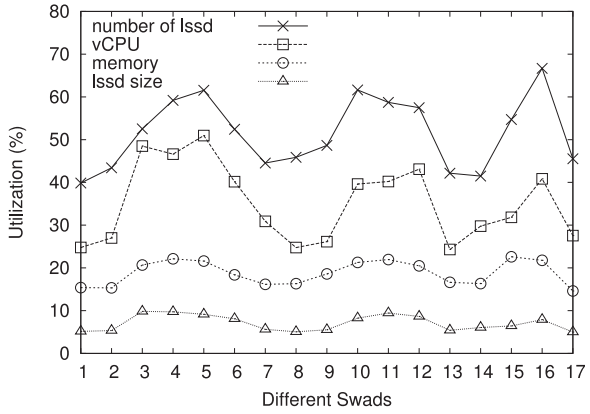


Fig. 2. Resource utilization for different swads; mix-1.

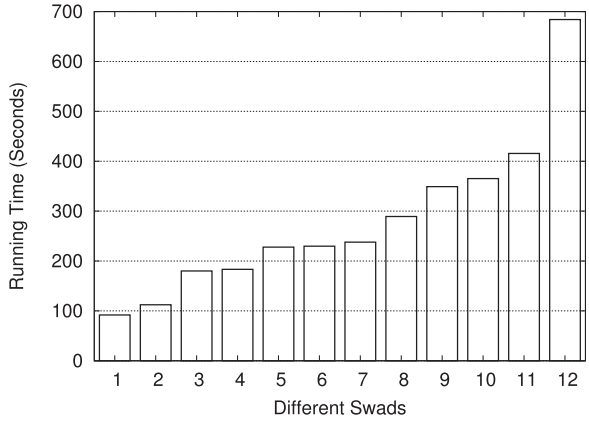


Fig. 3. Sorted running times for VM-to-PM assignments for different swads; mix-2.

the vCPUs and the total number of disks tend to be the resource bottleneck whereas the memory and disk space tend to be abundant.

The low utilization of some resources is in part due to the chosen pack and swad sizes. For instance, while the resource utilization at a swad may be low, bringing in another pack to it causes a big jump in the total resource requirements, likely exceeding the total amounts provisioned by the swad. The other part of the reason is the imbalance in the supply and demand of various resources. For instance, the disk-size-to-vCPU ratios of many VM types are relatively low, compared with the same ratios of most PM types. However, the imbalance may change as customers request different mixes of VM types.

4.4.2 Mix 2

The mixes of the VMs and PMs are described in Table 8. The safety margin is set to $\theta = 0.7$.

The two-level decomposition algorithm achieves a total cost 487,840. Out of the 25 swads, 12 of them are used. The total algorithm running time is 3,366 seconds, or 280.5 seconds per swad. The running time for each of the second-level VM-to-PM assignments (for the used swads) is shown in Fig. 3. The resource utilization results are shown in Fig. 4.

TABLE 8
1,000 VMs and 1,000 PMs-Mix-2

VM Type	No. of VMs	PM Type	No. of PMs
m3.medium	200	s1	100
m3.large	100	s2	100
m3.xlarge	100	s3	100
m3.2xlarge	100	s4	100
c3.large	50	m1	100
c3.xlarge	50	m2	100
c3.2xlarge	50	m3	50
c3.4xlarge	50	m4	50
c3.8xlarge	50	m5	50
r3.large	50	l1	50
r3.xlarge	50	l2	50
r3.2xlarge	50	l3	50
r3.4xlarge	50	l4	50
r3.8xlarge	50	l5	50
		16	0
i2.xlarge	0		
i2.2xlarge	0		
i2.4xlarge	0		
i2.8xlarge	0		

We ran the randomized heuristics 50 times, which took hours. The average cost of the heuristic algorithm is 601,914 and the standard deviation is 5,079; the minimum and the maximum costs are 589,900 and 613,520, respectively. The heuristic algorithm is about 23 percent more costly than the decomposition algorithm.

4.4.3 Mix 1 with Smaller Pack/Swad Sizes

Here, we want to show that decreasing the sizes of the packs and swads can reduce the computation time drastically. The mixes of the VMs and PMs are as described in Table 7. The safety margin is $\theta = 0.7$. The 1,000 VMs are divided into 50 packs and the 1,000 PMs are divided into 50 swads. Each pack has 20 random VMs and each swad has 20 random PMs.

The first-level assignment attempts to assign the 50 packs to the 50 swads. The result shows that 26 swads are used. The computation time is negligible.

Each second-level assignment attempts to assign 20 or more VMs (on average, $1{,}000/26 \approx 38$) to 20 PMs. The total running time is 202 seconds. The average running time is therefore 7.8 seconds per swad. We see that the running times are much smaller than the case in Section 4.4.1 (where the total is 1,281 and the average is 75 seconds, respectively).

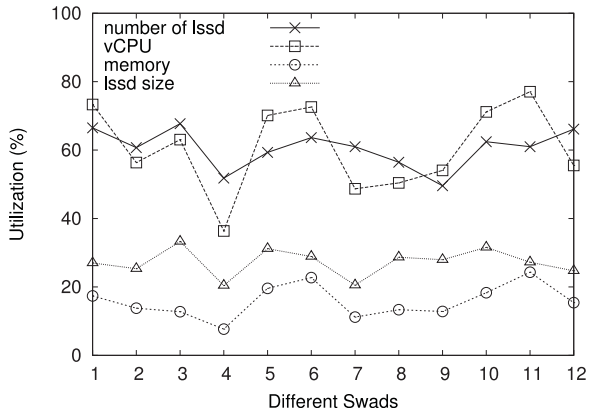


Fig. 4. Resource utilization for different swads; mix-2.

TABLE 9
Controlling Number of Swads Used by θ

θ	No. of Swads Used	No. of PMs Used	Total Cost
		Mix 1	
0.5	24	346	71,720
0.6	19	336	78,980
0.7	17	326	82,540
0.8	12	306	95,740
		Mix 2	
0.5	22	438	443,260
0.6	13	361	474,440
0.7	12	346	487,840

The total cost achieved by two-level decomposition is 98,040, still a big improvement over the randomized heuristics, which leads to a cost of 150,573.

4.4.4 Effects of the First-Level Assignment

There is a complex relationship between the assignment problems at the two levels. The resulting total cost depends crucially on how the MIP problem is formulated at the first level (for pack-to-swad assignment). For instance, it may appear reasonable that, in order to reduce the total operation cost, the first-level assignment problem should aim at reducing the number of swads used. We can control that number by varying the parameter θ. The results after the first and second-level assignments are shown in Table 9. As θ decreases, the constraints associated with the number of disks become more stringent in the first-level assignment and consequently, each swad is assigned fewer packs on average and more swads are used. However, after the second-level assignments, the total cost in fact decreases as θ decreases. Also, the number of PMs used after the second-level assignments increases as θ decreases. One explanation is that, as more swads are used, there are more second-level assignment instances (one for each used swad), and hence, there are more opportunities to improve the total cost. Although more swads and more PMs are used as θ decreases, cheaper PMs tend to be used and more expensive PMs tend to be avoided, resulting in a lower total cost. As θ continues to decrease, the first-level assignment problem will eventually become infeasible.

The above observations hold for the particular performance objective and cost structure. It should not be generalized without rigorous reasoning or extensive experiments. For instance, if every PM has the same operation cost, the total cost will be proportional to the number of PMs used. Then, based on the data in Table 9, the total cost would have increased as θ decreases.

In the end, θ needs to be tuned based on computational experiences. For instance, given a single instance of the VM placement problem, the two-level decomposition algorithm can be invoked multiple times under different values of θ. The θ value that gives the lowest total cost may be selected and the VM and disk placement will be made accordingly. In a real operating environment, the manager of the datacenter may also pick a suitable value for θ based on past experiences.

How to define the first-level assignment problem is a tough issue, which requires further research. On the positive side, there is an easy practical approach to address the issue, which is to experiment with different formulations and look at the total cost achieved. This approach is possible because the hierarchical decomposition method is scalable and the result for each experiment can be computed quickly.

4.4.5 Scalability

The two-level decomposition method is scalable with the help of parallelism. Suppose the basic building blocks of two-level decomposition are 100×50 VM-to-PM assignment problems and suppose each takes 2 minutes to solve. A system with 2 million VMs and 1 million PMs has 20,000 100×50 such assignment problems, which takes 40,000 minutes computation time. If 400 management servers are used to manage the datacenter, the running time on each is 100 minutes. Over a 24-hour day, there can be 14 rounds of complete re-assignment. The 400 management servers represent an overhead of $400/1{,}000{,}000 = 0.04$ percent, which is low.

It is unlikely that every VM needs to be re-assigned every 100 minutes. The numbers of VMs and PMs that need to be considered at each assignment period are likely to be drastically smaller than 2 or 1 million, respectively. Even a reduction by a factor of 10, i.e., 200,000 VMs and 100,000 PMs, can bring the total computation time down to 4,000 minutes or 10 minutes per management server. In practical systems, the variability of the problem sizes at the second level can be exploited. Some VM-to-PM assignment problems may be small, e.g., 40×20, which can be solved in seconds. On the other hand, larger VM-to-PM problems (e.g., 100×50) can be computed less frequently, such as once every few hours.

With smarter algorithms, it is hopeful that the computation time of each 100×50 VM-to-PM assignment problem can be cut down to sub-minutes. A factor of 10 reduction will have significant overall impact. Finally, we can always make most of the second-level problems smaller to speed up the overall computation. But, the achievable cost will be higher.

5 Conclusion and Discussion

In this paper, we study a new VM placement problem with difficult disk anti-colocation constraints. For solutions, we propose a hierarchical decomposition method combined with MIP formulations and algorithms. With simulation and numerical experiments, the proposed approach is shown to be effective. It achieves lower costs than an aggressive heuristic algorithm, and it is scalable with the help of parallel management servers. The proposed approach is extensible to other datacenter management problems. It can help to enable complex and system-oriented cloud services, enhance customer agility, and at the same time, improve datacenter resource efficiency or costs.

Customers' requests for VMs arrive at the datacenter dynamically. To handle the dynamic arrivals, we propose to wrap the two-level decomposition algorithm with an outer algorithm that combines periodic re-optimization with immediate placement. If the request is for a batch of

VMs, e.g., 1,000 VMs, as is often the case for enterprise customers, the two-level decomposition algorithm is executed immediately to place the set of requested VMs. If the request contains one or several VMs, it will be immediately placed using a simple online heuristic algorithm, such as a first-fit algorithm or a randomized algorithm. Periodically, a collection of VMs that have already been placed is selected to participate in periodic re-optimization. Their new placement is computed based on the two-level decomposition algorithm, subject to the consideration of migration constraints or costs.

We have seen that, in the two-level decomposition algorithm, the first-level assignment does not take much computation time. The computation challenge comes from the second-level VM-to-PM assignment problems. Ultimately, the sizes of these second-level problems need to be limited. Any future research that can improve that limit will be worthwhile. Improvement may come from more clever problem formulations, customization of the MIP algorithms, or new classes of algorithms.

REFERENCES

[1] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software defined infrastructures," *IBM J. Res. Develop.*, vol. 58, no. 2/3, pp. 2–2, Mar.–May 2014.

[2] C. Mega, T. Waizenegger, D. Lebutsch, S. Schleipen, and J. M. Barney, "Dynamic cloud service topology adaption for minimizing resources while meeting performance goals," *IBM J. Res. Develop.*, vol. 58, no. 2/3, pp. 1–10, Mar.–May 2014.

[3] W. C. Arnold, D. J. Arroyo, W. Segmuller, M. Spreitzer, M. Steinder, and A. N. Tantawi, "Workload orchestration and optimization for software defined environments," *IBM J. Res. Develop.*, vol. 58, no. 2/3, pp. 1–12, Mar.–May 2014.

[4] J. Koomey and J. Taylor, "New data supports finding that 30 percent of servers are 'Comatose', indicating that nearly a third of capital in enterprise data centers is wasted," 2015. [Online]. Available: http://anthesisgroup.com/wp-content/uploads/2015/06/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf

[5] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," *Proc. IEEE Online Green Commun. Conf.*, 2010, pp. 179–188.

[6] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," *Proc. IEEE INFOCOM*, 2011, pp. 71–75.

[7] H. Jin, D. Pan, J. Xu, and N. Pissinou, "Efficient VM placement with multiple deterministic and stochastic resources in data centers," in *Proc. IEEE Global Commun. Conf.*, 2012, pp. 2505–2510.

[8] Amazon, "Amazon EC2 Instances," [Online]. Available: http://aws.amazon.com/ec2/instance-types/

[9] A. Lê-Quôc, "Top 5 ways to improve your AWS EC2 performance," Datadog. [Online]. Available: http://www.datadoghq.com/pdf/top_5_aws_ec2_performance_problems_ebook.pdf

[10] G. Costa and D. Marti, "Choosing EC2 instances for NoSQL," ScyllaDB, Feb. 2016 [Online]. Available: http://www.scylladb.com/2016/02/26/best-amazon-ec2-instance-nosql/

[11] S. Newman, "A systematic look at EC2 I/O," [Online]. Available: http://blog.scalyr.com/2012/10/a-systematic-look-at-ec2-io/, 2012.

[12] *AWS Storage Services Overview-A Look at Storage Services Offered by AWS*, Amazon AWS, Nov. 2015. [Online]. Available: https://d0.awsstatic.com/whitepapers/AWS%20Storage%20Services%20Whitepaper-v9.pdf

[13] L. Poland, "What is the story with AWS storage?" DataStax Enterprise, Feb. 2014. [Online]. Available: http://www.datastax.com/dev/blog/what-is-the-story-with-aws-storage

[14] K. ODell, "How-to: Select the right hardware for your new Hadoop cluster," Cloudera, Aug. 2013. [Online]. Available: http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/

[15] J. Holoman and K. ODell, "How-to: Deploy Apache Hadoop clusters like a boss," Cloudera, Jan. 2015. [Online]. Avaialble: http://blog.cloudera.com/blog/2015/01/how-to-deploy-apache-hadoop-clusters-like-a-boss/

[16] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Hoboken, NJ, USA: Wiley, 1999.

[17] M. Goetschalckx, C. J. Vidal, and K. Dogan, "Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms," *Eur. J. Oper. Res.*, vol. 143, no. 1, pp. 1–18, 2002.

[18] J. Blazewicz, M. Dror, and J. Weglarz, "Mathematical programming formulations for machine scheduling: A survey," *Eur. J. Oper. Res.*, vol. 51, no. 3, pp. 283–300, 1991.

[19] D. Wedelin, "An algorithm for large scale 0–1 integer programming with application to airline crew scheduling," *Ann. Operations Res.*, vol. 57, no. 1, pp. 283–301, 1995.

[20] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[21] C. Guo, et al., "Secondnet: A data center network virtualization architecture with bandwidth guarantees," *Proc. ACM CoNEXT*, 2010, Art. no. 15.

[22] L. Hu, K. D. Ryu, D. D. Silva, and K. Schwan, "v-Bundle: Flexible group resource offerings in clouds," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*., Jun. 2012, pp. 406–415.

[23] M. Chen, H. Zhang, Y. Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," *Proc. IFIP/IEEE Integr. Netw. Manage.*, 2011, pp. 594–601.

[24] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," *Proc. Comput. Meas. Group Conf.*, 2007, pp. 399–406.

[25] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Comput. Netw.*, vol. 57, no. 1, pp. 179–196, 2013.

[26] Apache CloudStack Project. [Online]. Available: https://cloudstack.apache.org/, 2016.

[27] OpenStack Project. [Online]. Available: http://www.openstack.org/

[28] Eucalyptus Systems. [Online]. Available: http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html, 2016.

[29] Y. Xia, M. Tsugawa, J. Fortes, and S. Chen, "Hierarchical mixed integer programming for pack-to-swad placement in datacenters (work-in-progress)," *Proc. 12th IEEE Int. Conf. Autonomic Comput.*, 2015, pp. 219–222.

[30] O. Biran, et al., "A stable network-aware VM placement for cloud systems," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 498–506.

[31] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1033–1041.

[32] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2876–2880.

[33] Y. Guo, A. L. Stolyar, and A. Walid, "Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 620–628.

[34] S. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 702–710.

[35] P. Padala, et al., "Automated control of multiple virtualized resources," *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 13–26.

[36] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," *Proc. 9th ACM/IFIP/USENIX Int. Conf. Middleware*, 2008, pp. 243–264.

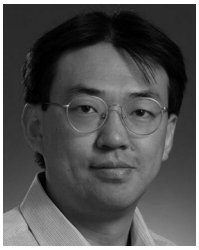
[37] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 62–73.

[38] J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," *Proc. 8th ACM Int. Conf. Autonomic Comput.*, 2011, pp. 225–234.

[39] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," *Proc. IEEE 14th Int. Symp. High Performance Comput. Archit.*, Feb. 2008, pp. 101–110.

[40] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, 2009, pp. 157–168.

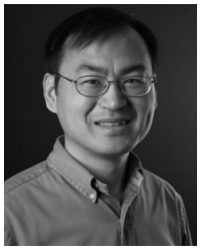
[41] M. Fielding, "Virtual CPUs with Amazon web services," Jun. 2014. [Online]. Available: http://www.pythian.com/blog/virtual-cpus-with-amazon-web-services/
[42] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise datacenters," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 331–340.
[43] X. Li, Z. Qian, S. Lu, and J. Wu, "Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center," *Math. Comput. Model.*, vol. 58, no. 5/6, pp. 1222–1235, 2013.
[44] Gurobi, Gurobi Web Site. [Online]. Available: http://www.gurobi.com/



Ye Xia received the BA degree from Harvard University, in 1993, the MS degree from Columbia University, in 1995, and the PhD degree from the University of California, Berkeley, in 2003, all in electrical engineering. He is an associate professor in the Computer and Information Science and Engineering Department, University of Florida, starting in Aug. 2003. Between June 1994 and August 1996, he was a member of the technical staff with Bell Laboratories, Lucent Technologies, New Jersey. His main research area is computer networking, including performance evaluation of network protocols and algorithms, resource allocation, wireless network scheduling, network optimization, and load balancing on peer-to-peer networks. He also works on cache organization and performance evaluation for chip multiprocessors. He is interested in applying probabilistic models to the study of computer systems.

Mauricio Tsugawa received the BS and MS degrees in electrical engineering from the University of So Paulo, and the PhD degree in electrical and computer engineering from the University of Florida. From 2009 to 2015, he was an assistant research professor with the University of Florida. His primary research interests include distributed computing, computer architecture, virtualization technologies and computer networks.

José A. B. Fortes received the BS degree in electrical engineering (Licenciatura em Engenharia Electrotécnica) from the Universidade de Angola, in 1978, the MS degree in electrical engineering from the Colorado State University, Fort Collins, in 1981 and the PhD degree in electrical engineering from the University of Southern California, Los Angeles, in 1984. He is the AT&T Eminent scholar and professor of electrical and computer engineering and computer science with the University of Florida where he founded and is the director of the Advanced Computing and Information Systems Laboratory. From 1984 until 2001 he was on the faculty of the School of Electrical Engineering, Purdue University, West Lafayette, Indiana. In 2001 he joined both the Department of Electrical and Computer Engineering and the Department of Computer and Information Science and Engineering, University of Florida as professor and BellSouth Eminent scholar. From July 1989 through July 1990 he served at the National Science Foundation as director of the Microelectronics Systems Architecture program. From June 1993 till January 1994 he was a visiting professor with the Computer Architecture Department, Universitat Politecnica de Catalunya in Barcelona, Spain. His research interests include the areas of distributed computing, autonomic computing, computer architecture, parallel processing and fault-tolerant computing. He has authored or coauthored more than 200 technical papers and has lead the Development and Deployment of Cloud and Grid-Computing Software used in several cyberinfrastructures for e-Science and digital government. His research has been funded by the Office of Naval Research, AT&T Foundation, IBM, General Electric, Intel, Northrop-Grumman, Army Research Office, NASA, Semiconductor Research Corporation and the National Science Foundation. He is a fellow of the Institute of Electrical and Electronics Engineers (IEEE) professional society and a fellow of the American Association for the Advancement of Science (AAAS). He was a distinguished visitor of the IEEE Computer Society from 1991 till 1995. He is on the editorial boards of the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *International Journal on Parallel Programming*. He is also a past member of the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *ACM Journal on Emerging Technologies in Computing Systems*, the *Cluster Computing: The Journal of Networks, Software Tools and Applications*, the *Journal of VLSI Signal Processing*, and the *Journal of Parallel and Distributed Computing*.



Shigang Chen received the BS degree in computer science from the University of Science and Technology of China, in 1993 and the MS and PhD degrees in computer science from the University of Illinois, Urbana-Champaign, in 1996 and 1999, respectively. He is a professor with the Department of Computer and Information Science and Engineering, University of Florida. After graduation, he had worked with Cisco Systems for three years before joining University of Florida, in 2002. He served as CTO for Chance Media Inc. during 2012-2014. His research interests include computer networks, Internet security, wireless communications, and distributed computing. He published more than 150 peer-reviewed journal/conference papers. He received IEEE Communications Society Best Tutorial Paper Award and NSF CAREER Award. He holds 12 US patents. He is an associate editor of the *IEEE/ACM Transactions on Networking*, and served as editors for a number of other journals. He served in various chair positions or as committee members for numerous conferences. He is a fellow of the IEEE and a distinguished lecturer of the IEEE Communication Society.