Hierarchical Mixed Integer Programming for Pack-to-Swad Placement in Datacenters: Concept and Analysis

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

***Abstract*—Computational clouds have evolved to go beyond cost-effective on-demand hosting of IT resources and elasticity. The added features now include the ability to offer entire IT systems as a service that can quickly adapt to changing business environments. The new trend introduces challenges in datacenter resource management, including scalability, system-orientation, and optimization supporting both datacenter efficiency and customer system agility and performance. The conventional VM-centric approach of resource management adopts a flat fine-grained model that results in problem formulations of enormous sizes. The difficulty in scalability hinders cloud providers' ability to improve resource efficiency and workload performance. In this paper, we introduce a pack-centric approach to datacenter resource management by abstracting a system as a pack of resources and considering the mapping of these packs onto physical datacenter resource groups, called swads. The assignment of packs/VMs to swads/PMs is formulated as an integer optimization problem that can capture constraints related to the available resources, datacenter efficiency and customers' complex requirements. Scalability is achieved through a hierarchical decomposition method. The new datacenter resource management framework is illustrated with a concrete resource allocation problem, which is interesting by itself and challenging. The problem formulation, our solution, and numerical experiments show the capability of integer programming formulations, the scalability of the hierarchical decomposition method, and the benefits of the overall framework.**

***Index Terms*—Cloud Computing, Datacenter, Virtual Machine Placement, Resource Allocation, Integer Programming**

I. 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. However, the value proposition offered by computational clouds has evolved to go beyond cost-effective on-demand hosting/management of IT resources and elasticity [1]. The added value now includes the ability to offer entire IT systems as a service (versus isolated resources that the customer needs to build into a system) 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. An example is an enterprise outsourcing its entire intranet to the cloud, instead of simply asking for a number of virtual machines (VM). The intranet consists of layers of networking and resource grouping that need to be implemented in datacenters.

These new capabilities - agility and continuous optimization - are enabled by building datacenters where resources and environments can be (re)configured programmatically on the basis of monitoring information and predictive models of behavior and workload. Industry has recently coined the term "software-defined" to refer to these new types of datacenters [1]–[5], built on the foundation of virtualization of computing, network, storage, software and all other datacenter resources. A parallel development is that some datacenters are rapidly increasing in scale, having up to several millions of servers, hundreds of thousands of cloud customers, and millions of end users of services provided by cloud customers [6].

The above-mentioned evolution of datacenters introduces new challenges in management as well as new opportunities [1]–[4]. The challenges are scalability, system-orientation, and optimization supporting both datacenter efficiency and customer system agility and performance. The opportunities are in considering systems as units of management therefore creating (1) hierarchical structure in resource management which contributes to scalability and (2) linkage between resources and business objectives which contributes to agility. In this paper, we pursue these opportunities by abstracting a system as a *pack* of (virtual) resources and considering the mapping of these packs onto physical datacenter resource groups (called *swads*) subject to constraints related to the physical datacenter topology and available resources, datacenter efficiency and customers' complex requirements. This is a major departure from most previous datacenter management approaches which rely on a VM-centric view rather than a system/pack-centric approach (hereon referred as pack-centric approach).

In VM-centric management of virtualized resources, each customer specifies a desired number of virtual machines (VMs)[1] as well as the resource requirements for each VM, including CPU, memory, storage, I/O throughput, and possibly bandwidth between VM pairs, defined in deterministic terms [7]–[9] or stochastic terms [10], [11]. A cloud provider's datacenters have a large number of server blades (physical machines, PM) mounted on racks and connected through layers of switches that form the datacenter network [12]. One of the important problems for datacenter resource management (a.k.a. allocation) is to map VMs to PMs such that certain

Shigang Chen and Ye Xia are with the Department of Computer and Information Science and Engineering, University of Florida.

Jose A. B. Fortes and Mauricio Tsugawa are with the Department of Electrical and Computer Engineering, University of Florida.

[1]We focus on datacenters where provisioning is based on resource virtualization. This paper's results are also extendable to the management of other types of datacenters (e.g., those using bare-metal provisioning).

cost, profit or performance objectives are optimized, subject to server resource constraints and network bandwidth constraints.

The above VM-centric problem is already quite difficult to solve for a large datacenter. Instead of optimal resource management, most deployed or proposed management schemes tackle specific aspects of the problem and usually adopt sub-optimal, heuristic solutions (see Section II for discussion of related work). The problem becomes even more complicated when datacenters attempt to support entire IT systems as a service with the intended agility and performance, for the following reasons. (1) A system deployment requested by a customer may contain complex relationships among the system components, such as resource grouping and hierarchy, various colocation or anti-colocation constraints, topological relationships, or workflow dependencies and traffic patterns. (2) The resource demand from each customer can vary significantly over time; thus, the resource allocation decisions need be continuously carried out. (3) Customers may not know the fine-grained system requirements at the individual VM level at all time, but may have partial information at certain aggregate levels and/or at a coarser timescale. (4) The resource allocation outcome often needs to achieve complex goals in multiple areas, such as the datacenter profit, energy consumption and thermal management, resource utilization, migration costs, and customers' workload performance (e.g., throughput, latency).

All the above boil down to two difficult issues - (1) how to incorporate all the complexity into workable problem formulations for high-quality resource allocation decisions, and (2) how to solve the problems in a timely manner. To address the first issue, we propose to use mixed integer programming formulations [13], which are general and flexible, thus ideally suited to characterize customers' complex requests of all sorts (see the IBM Connections example in Section III-D and the problem in Section IV), as well as complex objectives. The second issue is important, because, for a large datacenter, an integer programming formulation can easily involve trillions of variables as our later examples suggest, and there is no hope to solve it optimally within acceptable time. We propose a hierarchical decomposition method to break the large, hard problem into many subproblems, each of which will be sufficiently small and solvable quickly by integer programming algorithms. The decomposition is possible in part because of our abstraction of packs and swads. The subproblems have one-to-one correspondence to the swads in the swad hierarchy. They can mostly be solved independently from each other, making parallel computation possible by using multiple management servers. Unlike many prior work where various specialized algorithms are invented for special problems, the use of generic integer programming algorithms means that there is no need to craft different specialized algorithms for customers' constantly-changing requirements.

We next summarize the main contributions of this paper. (1) We propose a pack-centric approach to datacenter resource management, which is capable of supporting system-oriented services. (2) We adopt mixed integer programming formulations and algorithms for datacenter resource management problems since they are capable of capturing and solving complex, changing, sometimes vague requirements and constraints, thus providing the envisioned agility of the next-generation datacenters. Since optimization is involved, such formulations are a good starting point for improved performance with respect to the datacenter's and customers' objectives. (3) For scalable solutions, we propose hierarchical decomposition of the resource allocation problem in accordance with the pack and swad hierarchies that are often natural in the system-oriented, pack-centric datacenters.

The rest of the paper is organized as follows. In Section II, we discuss additional related work. In Section III, we discuss the pack-centric approach, including an overview of the pack and swad hierarchies, hierarchical decomposition and periodic re-optimization. In Section IV, we apply the integer programming formulation to an important datacenter resource management problem, which is to assign a large number of VMs to a large number of PMs under a disk exclusivity requirement. The problem illustrates how the integer programming formulation is capable of capturing complex constraints. It also shows the scalability challenges in such problems. In Section V, we show how to solve the above problem using hierarchical decomposition and we present experimental results to demonstrate the effectiveness of the approach. In Section VI, we draw conclusions and discuss additional issues.

II. Related Work

Prior studies on datacenter management generally avoid integer programming formulations all together. In the cases where integer programming is used, the focus is usually on developing more specialized combinatorial algorithms such as multi-dimensional bin-packing [14], [15], graph algorithms [10] or sophisticated heuristics [16], which are only applicable to special problems with the structures required by those algorithms. The resource allocation problems that we consider have varied, far more complex constraints, often unknown ahead of the time. The aforementioned specialized algorithms will not apply. Practical cloud systems usually adopt less sophisticated heuristics, such as first-fit and round-robin, as evidenced by open-source middleware stacks [17]–[19]. While simple heuristics can be highly scalable, they can also be underachieving in terms of performance.

We propose to stay with integer programming formulations as far as possible. To find solutions, we propose to use a hierarchical decomposition method to replace a flat, large integer programming problem by many much smaller subproblems, each of which is far easier to solve and can be solved in parallel by separate solvers. The authors of [20] also propose an idea of hierarchical decomposition, but on a specific problem with the specific objective of reducing datacenter network traffic by intelligent placement of the VMs (see also [16]). They also assume a particular network structure. Consequently, their algorithm is tailored for that particular situation. In contrast, our pack-centric approach is a more thorough-going hierarchical framework for datacenter resource management. Combined with integer programming formulations, it is also a more general and flexible framework. Our hierarchical decomposition is tied to the pack/swad hierarchies, and hence, is systematic.

A recent work [4] overlaps with this paper in considering how a software-defined datacenter can offer system-oriented complex services to the cloud customers. The level of complexity in service offerings and performance objectives is comparable to what we envision. However, the approaches of the two are quite different. While we propose to explicitly formulate integer optimization problems periodically and in batches and solve the problems by integer programming algorithms, [4] does not have a clearly formulated optimization problem. For resource placement, [4] 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 of the placement based on the observed performance. The adjustment is also through randomization, although the parameters for random sampling get modified over time (the details of how to do this are missing in [4]). Colocation and anti-colocation requirements are satisfied by sequential placement of the resources, with the resources placed earlier constraining the placement of the later resources. Our periodic batch optimization approach will bring improved resource efficiency and workload performance over any incremental, heuristic approaches. Finally, in between batch optimization events, our framework also incorporates the immediate random placement aspect in [4] for simple requests; complex requests are also handled immediately but by optimization.

The authors of [21] formulate a combinatorial optimization problem of joint VM placement and routing in datacenters. They develop a heuristic randomized algorithm by applying the Markov chain Monte Carlo method. In [22], a more general version of routing and VM placement problem is formulated and solved with an online randomized algorithm (see also [23]). Due to their specificity and limited scalability, these algorithms are unlikely to be directly applicable to large, complex datacenter management problems that we envision. Our decomposition method will break a large, complex optimization problem into much smaller subproblems of optimization. For each subproblem, we can always apply standard integer programming algorithms. Non-standard algorithms, such as randomized algorithms or approximation algorithms (e.g., see [24]), can also be used; but they will have to be developed from scratch due to the significant differences in the problems.

There are related proposals on short-term dynamic resource re-allocation based on workload monitoring and prediction [25]–[35]. Their common assumption is that the workload dynamic can be captured by linear system models or other statistical models. Then, adaptive control algorithms are developed. A comprehensive discussion can be found in the survey paper [36]. These proposals typically are restricted to (1) managing much smaller systems, e.g., a single PM with multiple VMs or a small-scale server cluster, (2) with limited performance objectives, e.g., energy saving while providing the required service quality to the workload, (3) and with limited workload complexity, e.g., no colocation constraints and no network requirement. Such restrictions make the resource allocation problems much simpler and thereby allow algorithms to re-allocate 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 (than our re-optimization period) for some of the simpler subproblems.

III. Pack-Centric Architecture of Datacenter Resource Management

The new pack-centric framework supports a variety of sophisticated, complex, system-oriented cloud-computing services. It also allows natural hierarchical grouping of resource requests, which can then be mapped to the hierarchy of physical resource groups in a datacenter. The hierarchies enable a scalable, hierarchical decomposition approach for solving large resource allocation problems.

A. Definitions and Examples of Pack and Swad

1) Pack Hierarchy: We propose a new abstraction called *pack*, which is a set of *VMs, smaller packs and other (virtual) resources* that should be placed as a group in a datacenter for the purpose of resource sharing or performance enhancement. This recursive definition allows a customer to organize its resource requirement in a hierarchical structure, as illustrated by Fig. 1, which shows a scenario of a multinational corporation outsourcing its IT infrastructure to the cloud. The corporation has a branch in London, a branch in Shanghai, and its headquarters in San Jose, corresponding to three packs. The headquarters pack further consists of a firewall VM and three lower-level packs, describing the resource requirements by the management department, finance department, and engineering department, respectively. Continuing further with the example, let's suppose the pack for the London branch requires 100 VMs for various databases, servers and other computing tasks. The workload on these VMs and the resource requirement cannot be fully predicted since the VMs may be assigned to different tasks as needed in the future. Instead of committing a fixed amount of each resource to each VM, it makes more sense to specify the combined requirements for each resource for the whole pack and let the VMs share the common pool of resources dynamically as needed. A natural way to facilitate resource sharing is to place these VMs as a group in a cluster of colocated physical servers. A controller can be implemented to monitor in real time the resource usage by the pack and dynamically adjust the resource distribution among the VMs.

2) Swad Hierarchy: We define a *swad* as a set of PMs, other physical resources (e.g., network storage) or lower-level swads in a cloud system. The resource capacity of a swad is equal to the sum of the capacities of its components, possibly excluding a certain percentage of resources that may be set aside to support elasticity (which allows a VM or pack to dynamically scale up its resources in real time). In the example of Fig. 2 with a fat-tree topology [37], a swad corresponds to all the servers of a rack, shown by the nodes labeled with "swad" in the right figure. We then recursively group a number of lower-level swads into a higher-level swad, as illustrated by the nodes labeled with "SWAD", giving rise to a hierarchical structure (tree), with the whole cloud system as a swad at the root. While the fat-tree topology in Fig. 2 has two levels (pod and rack), the swad hierarchy may have an arbitrary number of levels.

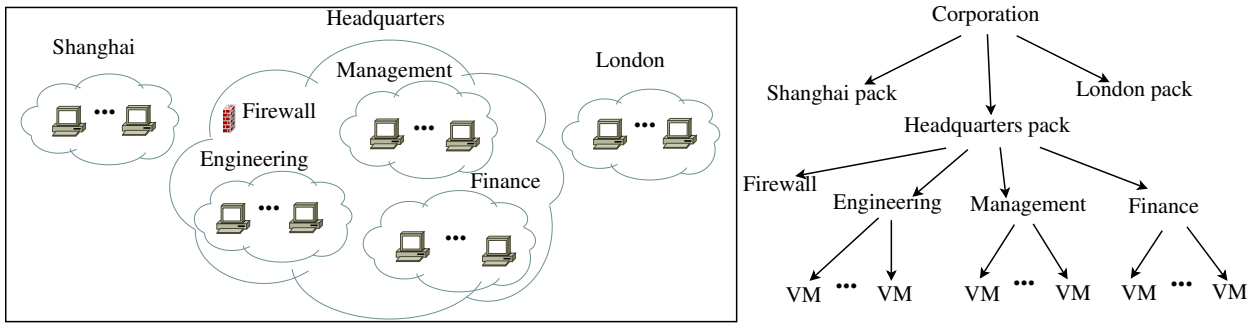

Fig. 1. VMs and other virtual resources can be organized through a recursive, hierarchical pack structure determined by administrative boundaries, locations and resource sharing requirement.

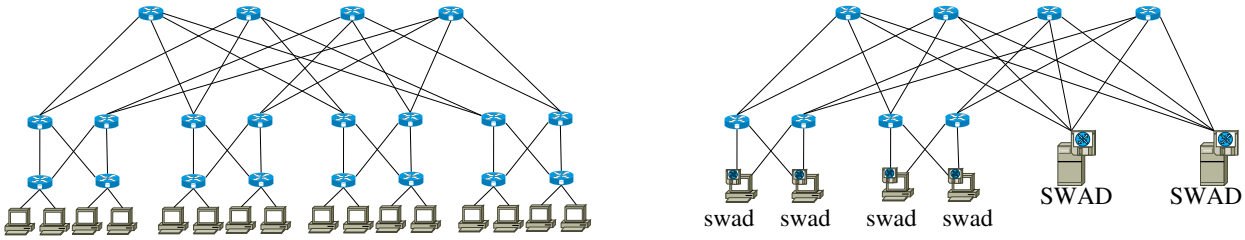

Fig. 2. Swad-based hierarchical abstraction of a cloud system

B. Construction of Pack and Swad Hierarchies

The swad hierarchy is established by the cloud provider alone according its resource management policies, proximity of physical resources and various other constraints (see Fig. 3).

Example: Suppose the cloud system has several geographically separated datacenters. The entire cloud system can be abstracted as a level-0 swad. Each datacenter becomes a level-1 swad. Suppose each datacenter has a number of zones, where each zone corresponds to a highly interconnected region in the datacenter. Then, each zone can be a level-2 swad. Each rack in a zone can be a level-3 swad. The PMs on a rack are level-4 nodes, which are leaves of the swad hierarchy (tree).

The pack hierarchy is established by combining the customers' pack specifications and the cloud provider's considerations. A customer's request may be already in the form of a pack hierarchy, such as the example of Fig. 1. The cloud provider collects such pack requests from the customers and establishes the final pack hierarchy (see Fig. 4). Each customer's pack hierarchy will be a subtree in the final pack hierarchy. The provider can group multiple pack hierarchies from the customers into a higher-level pack. The grouping process continues upward recursively to form the final pack hierarchy. There is a great deal of flexibility in constructing the pack hierarchy. The construction ultimately depends on many factors, including the provider's resource management policy, customers' need for pack isolation and/or colocation, security arrangement, the details of the swad hierarchy and other constraints in the datacenter. For instance, packs from customers with business ties may be placed under a common parent pack to increase the likelihood of colocation for better communication performance.

A comprehensive study on how to construct the swad and pack hierarchies is outside the scope of this paper. Subsequently, we assume both hierarchies are given. The swad and pack hierarchies each form a tree. The depth of a node in a tree is called the *level* of the node, with the root node at level 0 by convention. The leaves of the pack hierarchy can be VMs or packs; the leaves of the swad hierarchy can be PMs or swads.

C. Hierarchical Decomposition - Pack-to-Swad Assignments

Aside from the benefits that packs allow system-oriented cloud services and increased customer agility, the other main use of the pack and swad hierarchies is to break a large resource allocation problem of enormous complexity into a series of much smaller subproblems that are far easier to solve quickly and can be mostly solved in parallel[2]. There is an assignment subproblem associated with each swad on the swad hierarchy, which assigns some packs/VMs to the child swads/PMs of the focal swad.

Starting from the root of the swad hierarchy, the assignment is performed recursively. First, the root of the pack hierarchy is assigned to the root of the swad hierarchy, assuming the swad has sufficient resource capacity to support the pack. The next step is to assign the children of the root pack to the children of the root swad. The assignment process then continues downward along the swad hierarchy, in either depth-first or breath-first traversal order. In general, consider a swad at level i, where $i = 0, 1, \ldots$. Assume that some level-i packs have been assigned to the swad in question. For that swad, the next assignment to do is to assign the child packs/VMs of

[2]All subproblems at the lowest levels can be solved independently from each other. As the experimental results in Section V show, the computation time for these subproblems dominates the overall computation time.

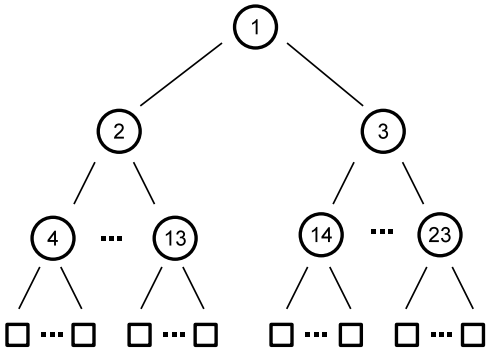

Fig. 3. An example of a swad hierarchy prepared by the cloud provider. Circles are swads; squares are PMs.

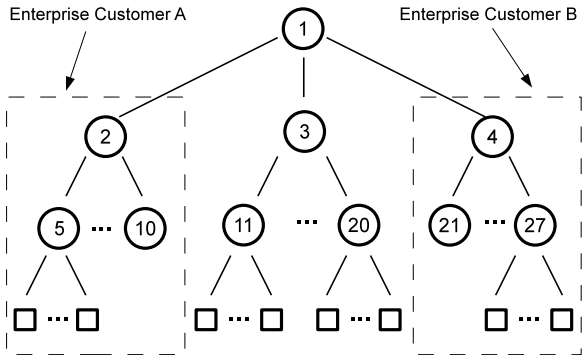

Fig. 4. An example of a final pack hierarchy. Circles are packs; squares are VMs. Enterprise customers A and B specified their own pack hierarchies, as shown in dashed boxes. Packs 11 to 20 are created by the cloud provider; each aggregates a set of VMs requested by individual customers.

the those level-i packs to the child swads/PMs of the level-i swad.

Small-scale VM-to-PM assignments show up as special cases of the above decomposition process. Consider a swad at height 1 and suppose all its children are PMs. Suppose all the packs assigned to the swad contain only VMs as their children. Then, the next assignment problem associated with the swad is a VM-to-PM assignment.

Example. Consider the assignment problem associated with the pack hierarchy in Fig. 4 and swad hierarchy in Fig. 3. The first step is trivial: Pack 1 is assigned to swad 1. The next step is to assign packs 2, 3 and 4 to swads 2 and 3. Suppose the result of the assignment (by solving an optimization problem chosen by the provider) is that packs 2 and 3 are assigned to swad 2, and pack 4 is assigned to swad 3. Next, consider swad 2. The assignment problem is to assign packs $5-20$ to swads $4-13$. Similarly, for swad 3, the assignment problem is to assign packs $21-27$ to swads $14-23$. Now, suppose packs $11-14$ are assigned to swad 13 in the assignment step associated with swad 2 (based on another optimization problem). The assignment problem associated with swad 13 is to assign all the VMs in packs $11-14$ to the PMs in swad 13.

To summarize, hierarchical decomposition transforms the complex problem of global resource management into a hierarchically structured one, where the resource requirements and performance objectives can be enforced iteratively from higher-level swads/packs to lower-level swads/packs through decentralized monitoring and allocation systems.

D. Periodic Re-Optimization by Integer Programming

Another key idea of the new datacenter management framework is to perform periodic re-optimization and, based on that result, update the assignment of packs/VMs to swads/PMs. The optimization problems are formulated as integer programming problems to capture complex constraints and customer requirements, such as various colocation or anti-colocation constraints, topological relationship among components, or even workflow precedence relationship.

IBM Connections Example: We will illustrate complex customer requirements by an example taken from [4], which is a (virtual) system for deploying IBM Connections, a business collaboration platform. In the example, the deployment requires four (virtual) server clusters and each cluster contains four virtual servers (i.e., VMs), with the requirement that (i) the VMs in a cluster must be spread across at least two racks, (ii) with no more than two VMs on each rack and (iii) no two VMs in a cluster can share the same PM. The deployment also requires two front-end HTTP servers, to be placed on separate racks for fault tolerance and high availability. Other components include a VM running the management server, a VM running the security server, an external database server and an external NFS server. Various VMs need to be connected with desired network bandwidth and latency and protected by firewall rules.

Periodic re-optimization may be triggered by timers or by events (e.g., such as when the estimated resource efficiency is below a threshold).

When re-optimization is due: The following actions are taken.

- A subset of packs/VMs is selected and a subset of PMs/swads is selected to participate in re-optimization. The selection depends on the cloud provider's policy and the states of the packs/VMs and swads/PMs, e.g., whether a PM is running large workload and cannot be interrupted.
- A large integer optimization problem is formulated for optimal assignment of the packs/VMs to swads/PMs, taking into account both the customers' and the provider's objectives and various constraints. The optimization problem covers all the packs/VMs and all the swads/PMs selected in the previous step.
- A pack hierarchy and a swad hierarchy are constructed, and the assignment subproblems in hierarchical decomposition are solved by integer programming algorithms. The packs/VMs are placed according to the solution, which may involve VM migration.

In between re-optimization events: Customer requests for new or additional resources are handled immediately upon arrival. If a request is simple (e.g., for a set of non-interacting VMs), it is handled by any online heuristic algorithm such as first-fit or the randomized algorithm in [4]. If a request is complex (e.g., the IBM Connections example, or more generally, a pack with

its own hierarchy and dependencies among its components), a small-scale integer optimization problem is formulated to allocate the requested resources, involving only the current request(s) and a small set of physical resources.

There are several reasons to perform resource allocation periodically: (1) Since on-demand packs, VMs or other resources are placed immediately through sub-optimal online heuristics or small-scale optimization, resource inefficiency can accumulate and eventually become significant; periodic re-optimization can restore resource efficiency. (2) To support elastic services such as auto-scaling, the resources allocated to a customer's packs are allowed to dynamically change based on the customer's time-varying workload and measured performance. (3) A customer may explicitly modify its resource requirement for a variety of reasons.

After a re-optimization event, redistribution of the resources may incur pack/VM migration and migration costs should have been factored into the resource allocation formulation. This is to ensure that re-optimization does not cause performance degradation or costly disruption to the users. For instance, for those packs or VMs that have very large migration costs, constraints can be added to prevent their migration. On the other hand, inactive packs and VMs have very low migration costs, especially when the network traffic is light (e.g., during the night time); they can be selected to participate in re-optimization without any migration constraints.

The frequency of re-optimization is determined by the overhead-benefit tradeoff, which may be estimated or measured. If re-optimization is done too frequently, the computation cost and various other overheads may be large while the benefit may be small because only small changes may have occurred during the short period of time between two consecutive optimization events. If it is done too sparsely, the opposite will be true.

IV. A Concrete Datacenter Resource Management Problem

In the following, we will describe an important datacenter resource management problem. It serves to illustrate why we advocate the integer programming formulation, which is that it is capable of describing complex constraints. In this case, it easily captures a subtle anti-colocation constraint of disks, i.e., the disk exclusivity requirement. The example also demonstrates the enormous scale and complexity of datacenter management problems in general. In fact, the problem to be described corresponds to a relatively straightforward scenario. But, the integer programming formulation is exceedingly large and difficult to solve using conventional algorithms. One can imagine that more complicated datacenter management scenarios will be much more difficult solve. We will later show how hierarchical decomposition provides a scalable solution framework. Finally, being a novel variant of the VM-to-PM assignment problem, the problem is interesting in its own right.

A. Problem Formulation

We consider a problem of assigning N VMs to M PMs with disk exclusivity constraints, which will be explained. Here, N and M can both be fairly large, e.g., thousands or more. Each VM has the following resource requirements: memory (GB), vCPU, number of local disk volumes (virtual ones) and their sizes. For the numerical experiments later, we will use the VM types of Amazon EC2 and the local disks are SSDs (denoted as lssd). The model can clearly be extended to include SAN (storage area network) SSDs, local and SAN magnetic disks, and local and network bandwidth constraints.

Each of the M PMs has certain memory, a number of vCPUs, a number of local disks (SSDs) and their sizes. These local disks may be in the PM or directly attached (either by local disk interface or by fast, dedicated network interface such as fiber channels).

1) Constraints: We first give an overview of the constraints for this problem. With respect to each resource (e.g., vCPUs or memory), the constraints are that the total amount of resource required by all the VMs assigned to each PM j cannot exceed the resource capacity of PM j. When a VM i requests multiple local disks, there is often an *exclusivity requirement*: no physical disk of the PM (to which VM i is assigned) can contain more than one of VM i's requested virtual disks. For instance, separate physical disks allow the end-user of the VM to enjoy higher total disk throughput and/or more fault tolerance. Our problem formulation contains several constraints to capture the exclusivity requirement. We should point out that the disk exclusivity requirement is a form of anti-colocation constraints and it makes the problem much harder to solve. The problem is also much harder than typical VM-to-PM placement problems with VM colocation and anti-colocation constraints. A final set of constraints is that the capacity of each physical disk must be greater than or equal to the aggregate size of all virtual disks assigned to it.

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)[3]. Suppose for each VM i, a set of virtual disks is requested and the set is denoted by 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 available vCPUs, M_j be the amount of memory (in GiB), and 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 from VM i to PM j, which takes the value 1 if i is assigned to 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

[3] 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.

required:

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

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

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

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

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

The condition (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. The condition (2) ensures that every requested virtual disk must be assigned to exactly one physical disk. The condition (3) ensures that every VM is assigned to exactly one PM. The condition (4) ensures that VM i cannot have more than one virtual disks assigned to the same physical disk. The condition (5) is the disk capacity constraint.

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. One of the key points is that integer programming formulations can capture subtle or complex requirements quite easily, such as the disk exclusivity requirement, pack/VM/storage colocation and anti-colocation constraints, resource grouping and hierarchy, other topological constraints, network constraints, and workflow dependencies. The various requirements in the IBM Connections example in Section III-D can all be expressed.

The following are the constraints posed by the number of vCPUs and the total memory size of each PM j.

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

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

2) Costs and Optimization Objective: The costs and performance 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 involved. The operation cost may include the average energy cost when a machine is running and typical maintenance 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 $x_{i,j} = 1$ for some $i \in \mathcal{V}$, we add the following two constraints, where B is a large enough constant (it is enough to take $B = N$).

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

$$B z_j \ge \sum_{i \in \mathcal{V}} x_{i,j}, \quad j \in \mathcal{P}. \tag{9}$$

We assume the assignment is feasible so that no VMs will be rejected. Then, 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,

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

Remark. The revenue/cost structure and optimization objective can be substantially refined. 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 cost may incorporate load-dependent costs (e.g., the energy cost is higher for higher load or running a CPU a higher clock speed) 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 [9] for a related treatment). Variants and examples of refinements are the subject of ongoing work.

3) Problem Complexity: As typical for integer programming problems, we will measure the complexity of the problem by the number of variables and the number of constraints, which can be counted easily. To make the matter more concrete, suppose 1000 VMs are to be assigned to 1000 PMs, i.e., $N = 1000$ and $M = 1000$. Suppose each VM requests 2 virtual disks and each PM provides 4 physical disks. Then, the number of y variables is $1000 \times 1000 \times 2 \times 4 = 8,000,000$ 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 an extremely large integer optimization problem, far exceeding what any integer optimization software can handle. A large datacenter may have $1,000,000$ PMs servicing more than $1,000,000$ VMs. Even if at each re-optimization period, only 10% of them participate in re-optimization, 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. The examples show that the disk exclusivity requirement is the main source of difficulty.

V. Solution and Simulation Results

This section presents the hierarchical decomposition solution to the problem in IV-A. To demonstrate its effectiveness in scalability and performance improvement, the solution is compared with an aggressive heuristic algorithm through simulation/numerical experiments.

TABLE I
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

TABLE II
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 × 1000	900
m4	16	256	8 × 1000	1500
m5	16	256	16 × 512	1800
l1	32	256	4 × 1000	2500
l2	48	512	8 × 1000	3500
l3	64	1024	4 × 1000	5000
l4	80	2048	16 × 1600	7000
l5	120	4096	4 × 1000	9000
l6	120	4096	24 × 1600	12000

A. Setup

1) VM Types: We take a subset of the allowed VM types (classes) of Amazon's EC2 [38]. Their resource requirements are shown in Table I. Below are the use cases of the different VM types, according to Amazon's web site.

- m3: m3 VM instances provide a balance of computing, memory, and network resources, and they are good choices for many applications.
- c3: c3 instances are the latest generation of compute-optimized instances, providing customers with the highest performing processors and the lowest price/compute performance available in EC2 currently.
- r3: r3 instances are optimized for memory-intensive applications and have the lowest cost per GiB of RAM among Amazon EC2 instance types.
- i2: i2 instances are equipped with large SSD storage optimized for very high random I/O performance, providing high IOPS at a low cost.

2) PM Types: 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 1 512-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 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 (virtual CPU) corresponds to a hyperthread of a physical core [39]. 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.

For our experiments, we use the PM types of Amazon EC2, which are listed in the first column of Table II. Cloud providers generally don't disclose the capabilities of all their PMs. The amount of resources that each type of PM is equipped with 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 II should be quite sensible. The operation costs (in the 5th column) are not exactly known *circa* 2014, but are based on our estimate[4]. Since the experiments results are comparative between two alternative algorithms, only normalized costs are needed.

B. Target of Comparison: Greedy Randomized Heuristic Algorithm

Since we are not aware of prior studies on exactly our problem instance, as a target for performance comparison, we had to develop our own heuristic algorithm. The heuristic algorithm is motivated by the general ideas of online randomized algorithms [4], [40], [41] but should achieve much lower costs than the latter due to two exhaustive search steps, which we will describe.

Imagine that VM requests arrive dynamically. An online randomized algorithm will assign a requested VM to some

[4]The large cost increase when the number of disks exceeds 4 reflects the cost of running separate DAS devices.

random PM one at a time in the arrival order of the VM requests. Note that, in our experiments, all the VMs to be placed are given together in a batch. Our greedy randomized 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 greedy aspect is that, for assignment, the list of *used* PMs, which are those already with some assigned VMs, is checked first; if the VM cannot be assigned to any PM in the used list, then the list of unused PMs is checked. The greediness tends to lead to more VM consolidation. In scanning either PM list, the order of scanning is uniformly random to emulate random selection; the first PM in the list that can accommodate the VM is selected (first-fit).[5]

For each scanned PM, our heuristic algorithm checks whether it is possible to assign the currently considered VM to that PM. For vCPU or memory, all that is needed is 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[6]. If the disk assignment (for the currently considered VM and PM) cannot be done by the algorithm, it is because the assignment is infeasible.

C. Results: Comparison between Flat Optimization and Greedy Randomized Heuristics

In this part of the experiments, we will show sample instances that can be solved by directly using the integer programming software Gurobi [42] *without* hierarchical decomposition, an approach that we call *flat optimization*. We will compare flat optimization with our heuristic algorithm. The intention is to demonstrate that (1) the optimization approach achieves much lower costs (i.e., the total operation costs) than even sophisticated heuristics; (2) however, the size of solvable instances is rather limited. Later in Section V-D, we will show how hierarchical decomposition can help to solve large instances while maintaing the advantage over the heuristic algorithm in achievable costs.

The results for experiments 1 and 2 are summarized in Table III.[7]

[5]For a large datacenter, a scalable online algorithms cannot afford to search through all the used PMs or unused PMs for each VM request. A typical strategy is to randomly 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 this case, exhaustive search is needed and scalability is limited. Our heuristic algorithm does not maintain an ordered list because there is no obvious criterion for the order due to the difficult disk exclusivity requirement.

[6]Checking the feasibility of disk assignment can be done by some standard assignment algorithm, which may be faster than enumeration but still takes some time. Either way, our heuristic algorithm has limited scalability, since in the worst case, there is one disk assignment problem for every PM for every VM request. But, it should achieve a lower cost than more scalable online randomized algorithms that do not check all the PMs for all possible disk assignment possibilities.

[7]Since the heuristic algorithm is not sufficiently scalable, there is no need to collect the computation time. See later comment in Section V-D2.

TABLE III
SUMMARY OF RESULTS: FLAT OPTIMIZATION V.S. HEURISTICS

Experiments		Flat Optimization	Heuristics
1	Cost	4540	9913
	Run Time (s)	106	
2	Cost	45300	65105
	Run Time (s)	3756	

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.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, our formulation of this problem involves 17950 binary variables and 26120 constraints, which make it non-trivial for any optimization software. The instance is solved by Gurobi in 106 seconds, yielding the optimal cost 4540.

The randomized heuristic algorithm, which serves as an target of comparison, can solve the problem more quickly. Since the heuristic algorithm involves randomization, we collected the results of 50 runs, which all together took several seconds. The average cost of the 50 runs is 9913, more than twice the optimal cost. The minimum, maximum and standard deviation of the costs are 6980, 12000, and 1074, respectively.

2) Experiment 2 – 77 VMs and 70 PMs: In this experiment, 77 VMs will be assigned to 70 PMs (see Table IV). Here, we have a fuller mix of almost all types of VMs and PMs. The instance has 55380 binary variables and 80825 constraints, quite a bit larger than the previous instance. 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. The problem took Gurobi about 3756 seconds (about 63 minutes) to solve, which is much longer than for the previous instance. The optimal assignment has a cost of 45300.

We ran the heuristic algorithm 50 times. The resulting average cost is 65105 and the standard deviation is 3683. The minimum and maximum costs are 58400 and 76200, respectively. The heuristic algorithm results in 44% 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.

3) Additional Results and Comments: Table V summarizes the computation time for flat optimization on several 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, 100 VMs and 100 PMs represent an upper limit of instances that can be solved under an hour

TABLE IV
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		

TABLE V
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	3756
90	75	4885

using standard integer programming algorithms on ordinary computers. Improvement of that limit may come from more clever problem formulations, customized algorithms, and more powerful computers.

D. Results: Two-Level Hierarchical Decomposition

The major experiments are to assign 1000 VMs to 1000 PMs of different types using our hierarchical decomposition method[8].

1) Two-Level Decomposition Algorithm: We split the VMs into 25 packs and the PMs into 25 swads randomly. In this case, the pack and swad hierarchies each have two levels. In the pack hierarchy, the root pack has 25 child packs, each of which has 40 VMs as children. Similarly, in the swad hierarchy, the root swad has 25 child swads, each of which has 40 PMs as children[9].

In the first level of assignment, we assign the 25 packs to the 25 swads by solving an optimization 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 PMs in the swad; we also record the maximum of the 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. Then, we solve an optimization problem that minimizes the number of swads used, subject to the constraints that, at each swad, the resource usage are no greater than the corresponding capacity of the same resource. For instance, the sum of the memory requested by all the packs allocated to a swad is no more than the total amount of memory provided by the swad. For the requested number of disks, we define a safety margin, $0 < \beta \leq 1$, and we stipulate that the total number of disks requested by all the packs assigned to a swad is no more than β times the total number of disks provided by the swad. The reason for doing so is that disk exclusivity is often a difficult constraint to satisfy in the second-level optimization problems. By reducing β in the first-level optimization (if needed), we can spread out packs more across the swads to gain more room for maneuver. We also have a constraint that the maximum amount of memory requested by any pack (which is the maximum amount requested by any VM in the pack) allocated to the swad is no more than the maximum amount of memory provided by any PM in the swad. We do the same for the maximum number of vCPUs.

A second-level assignment is performed for each of the swads that has some packs assigned to it. 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 perform optimal allocation of the VMs to the PMs using the formulation provided in Section IV-A, with the objective of minimizing the total operation cost of the PMs. For each of the swads, the minimum cost is given by the optimization solution (the cost is equal to zero if no packs are assigned to the swad). The overall cost is the sum of all the minimum costs for all the swads.[10]

The results of the experiments are summarized in Table VI.

2) Mix 1: The mix of VMs and PMs is described in Table VII. The safety margin is $\beta = 0.7$. The two-level decompo-

TABLE VI
SUMMARY OF RESULTS: TWO-LEVEL DECOMPOSITION V.S. HEURISTICS

Experiments		Two-Level Decomp.	Heuristics
Mix 1	Cost	82540	150573
	Run Time (s)	1281;75 per swad	
Mix 2	Cost	487840	601914
	Run Time (s)	3366;280.5 per swad	
Mix 1; Smaller Pack/Swad Sizes	Cost	98040	150573
	Run Time (s)	202;7.8 per swad	

[8]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.

[9]Although the pack hierarchy is stated as an artificial construction by us, the same hierarchy may arise as a combined effort of the customers and the cloud provider. For instances, some of the packs may be specified directly by customers, and other are created by the provider after aggregating some individually-requested VMs. Also, although each pack has 40 VMs and each swad has 40 PMs, multiple packs may be assigned to a swad in the first-level assignment, and hence, the total number of VMs may be significantly more than the number of PMs in the swad.

[10]There is still a possibility that the second-level optimization is 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 optimization at the pack-swad level. One remedy is to put safety margins on all the stringent resources (like the use of β) and find suitable values by binary search. When the constraints are violated by small amounts, one can simply move some of the packs or VMs to unused swads. For the experiments that we are presenting, all the second-level assignments turn out to be feasible.

TABLE VII
1000 VMs AND 1000 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		

sition algorithm achieves a total cost $82,540$. This number should be compared with the randomized heuristics, which has an average (over 50 runs) total cost $150,573$, a standard deviation 4951, 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 optimistic and actual online algorithms will 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 allocated 1 or 2 packs. The computation for the first-level assignment took very little time, on the order of a few seconds, due to the small problem size at this level. In general, the optimization at the upper levels for assigning packs to swads does not pose scalability challenges.

For the second-level VM-to-PM assignments, the total running time is 1281 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 these 17 different assignment subproblems 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 optimization algorithm. The running times are shown in Fig 5, sorted in increasing order. Overall, we see that the optimization that assigns VMs to PMs at the bottom level of hierarchical decomposition is where the computation complexity lies. To get a solution within a prescribed time budget, the size of each such optimization subproblem needs to be limited, which can be achieved by sufficient decomposition.

The heuristic algorithm took a fairly long time, hundreds of seconds per run. At the minimum, the computation time scales as the product of the total number of VMs and the total number of PMs. The disk exclusivity requirement poses greater scalability challenge as the number of disks offered by some

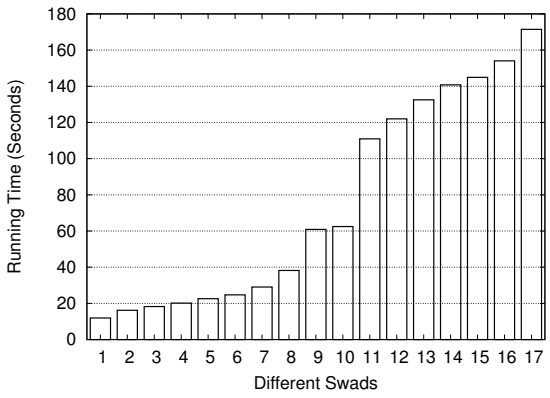

Fig. 5. Sorted running times for VM-to-PM assignment for different swads; mix-1

PM and/or the number requested by some VM increase[11]. The precise situation depends on the implementation of the heuristic algorithm. Overall, the heuristic algorithm can handle this instance. But, it is not a sufficiently scalable algorithm.

We next make additional comments about the experimental results. In the optimal solution for the first-level assignment, the utilization of various resources is generally low. This is in part due to the chosen granularities of the packs and swads, measured in the numbers of VMs and PMs, respectively. For instance, while the resource utilization at a swad may be low, bringing in another pack to the swad involves a big jump in the total resource requirements, likely exceeding the resources provisioned by the swad. The other part of the reason is the inherent imbalance in the supply and demand of various resources. The vCPUs tend to be the resource bottleneck whereas the memory and disk space tend to be abundant. The total number of disks also tends to be a stringent resource when considered jointly with the vCPUs. Recall that the disks requested by a VM can only be assigned to the PM to which the VM itself is assigned. Since each PM typically can accommodate a small number of VMs due to the vCPU constraint, it follows that a PM can accommodate a small number of disks even if the PM's total disk capacity is abundant.

The resource utilization is shown in Fig. 6. 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% to close to 70%. Given that the safety margin is set at $\beta = 0.7$, 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%. The total lssd size and

[11]Enumeration of the disk assignment possibilities works fine when the numbers of disks are small. But, the number of assignment possibilities rapidly increases with the disk numbers. For example, when a PM has 8 disks and a VM requests 4 disks, there are total $8!/4! = 1680$ assignment possibilities, which is a small number to enumerate; when the PM has 24 disks and the VM requests 8 disks, there are total $24!/8! = 29,654,190,720$ possibilities. The enumeration strategy becomes impractical for the second case. When that happens, one can use one of the standard assignment algorithms. The disk numbers in our experiments are chosen such that cases with a large number of assignment possibilities are avoided.

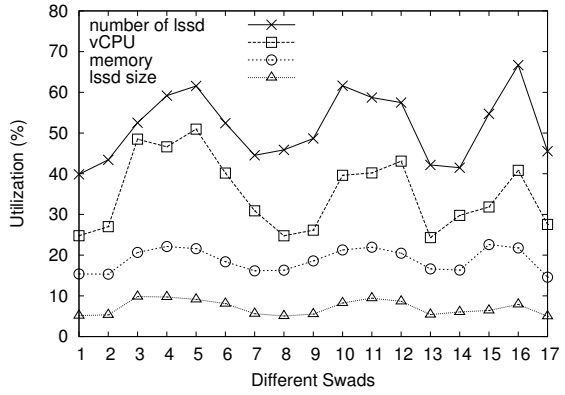

Fig. 6. Resource utilization for different swads; mix-1

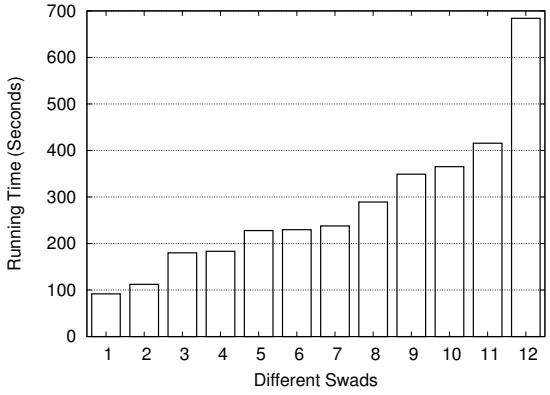

Fig. 7. Sorted running times for VM-to-PM assignment for different swads; mix-2

TABLE VIII
1000 VMs AND 1000 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		

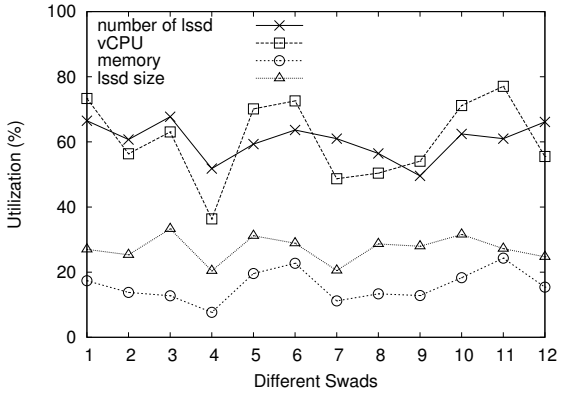

Fig. 8. Resource utilization for different swads; mix-2

the memory are seriously under-utilized, at around 10% and 20%, respectively.

3) Mix 2: The mix of VMs and PMs is described in Table VIII. The safety margin is $\beta = 0.7$.

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 3366 seconds, or 280.5 seconds per swad. The running time for VM-to-PM assignment (at the bottom level) for each of the used swad is shown in Fig. 7.

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 5079; the minimum and the maximum costs are $589,900$ and $613,520$, respectively. The heuristic algorithm is about 23% more costly than the decomposition algorithm.

For the decomposition algorithm, the resource utilization results are shown in Fig. 8.

4) 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 VII. The safety margin is $\beta = 0.7$. The 1000 VMs are divided into 50 packs and the 1000 PMs are divided into 50 swads. Thus, each pack has 20 random VMs and each swad has 20 random PMs.

The first-level optimization 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 subproblem attempts to assign 20 or more VMs (on average, $1000/26 \approx 38$) to 20 PMs. The total running time is 202 seconds, which is the aggregate for solving 26 subproblems corresponding to the 26 used swads. The average running time is therefore 7.8 seconds per swad. Both the total and the per-swad running times are much smaller than the case in Section V-D2 (1281 and 75 seconds, respectively).

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 $150,573$.

5) Effects of the First-Level Optimization: There is a complex relationship between the optimization problems at the two levels. The resulting cost depends crucially on how the optimization 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 optimization problem should aim at reducing the number of swads used. We can control that number by varying the parameter β. The results after the first and second-level optimization are shown in Table IX. As β decreases, the constraint about

the number of disks becomes more stringent in the first-level optimization and consequently, each swad is assigned fewer packs on average and more swads are used. However, after the second-level optimization, the total cost in fact decreases as β decreases. Also, the number of PMs used after the second-level optimization increases as β decreases. One explanation is that, as more swads are used, there are more second-level optimization instances (one for each used swad), and hence, there is more opportunity to improve the total cost. Although more swads and more PMs are used as β decreases, cheaper PMs tend to be used and more expensive PMs tend to be avoided, resulting in a lower total cost. As β continues to decrease, the first-level optimization 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 IX, the total cost would have increased as β decreases.

How to formulate the first-level (or, in general, higher-level) optimization appropriately 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 possible formulations and look at the total cost achieved, under the given set of cost structure, constraints and objective encountered in practice. This approach is possible because the hierarchical decomposition method is scalable and the result for each experiment can be computed quickly.

6) Scalability: The hierarchical decomposition method is scalable with the help of parallelism. Suppose the basic building blocks of hierarchical decomposition are 100×100 VM-to-PM assignment subproblems and suppose each takes 2 minutes to solve. A system with 1 million VMs and 1 million PMs has 10000 100×100 assignment subproblems, which takes $2 \times 10000 = 20000$ minutes computation time. If 200 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-optimization. The 200 management servers represent an overhead of $200/1000000 = 0.02\%$, which is low.

It is unlikely that every VM needs re-allocation of resources every 100 minutes. The numbers of VMs and PMs that need to be considered at each re-optimization period are likely to be drastically smaller than 1 million each, may be in the order of thousands to tens of thousands. Even a reduction by a factor of 10, i.e., 100,000 VMs and 100,000 PMs, can bring the total computation time down to 2000 minutes or 10 minutes per management server. In practical systems, the variability of the problem sizes at the bottom level can be exploited. Some VM-to-PM assignment subproblems at the bottom level may have smaller problem size, e.g., 20×20, which can be solved in seconds. On the other hand, larger subproblems (e.g., of the size 100×100) can be computed less frequently, such as once every few hours.

With smarter algorithms, it is hopeful that the computation time of each 100×100 VM-to-PM assignment subproblem 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 bottom-level subproblems smaller to speed up the overall computation. But, the achievable cost will be higher.

TABLE IX
CONTROLLING NUMBER OF SWADS USED BY β

β	No. of Swads Used	No. of PMs Used	Total Cost
Mix 1			
0.5	24	346	71720
0.6	19	336	78980
0.7	17	326	82540
0.8	12	306	95740
Mix 2			
0.5	22	438	443260
0.6	13	361	474440
0.7	12	346	487840

VI. CONCLUSIONS AND DISCUSSION

In this paper, we propose a pack-centric approach, combined with integer programming formulations and hierarchical decomposition, for datacenter resource management. The new approach enables complex and system-oriented cloud services, enhances customer agility, and at the same time, improves datacenter resource efficiency or costs. With simulation and numerical experiments, the approach has been shown to be effective and scalable. Next, we briefly discuss some additional issues.

Customer workload often exhibits time non-stationarity. The requested resources may be occasionally insufficient to handle large, temporary workload increase, or they may be overly abundant in other times. Our proposal handles non-stationary in several ways. (1) It allows customers to specify time-dependent resource requirements, e.g., for different time of the day, week or month. Batch optimization at each period uses the customers' requirements for that period as input. (2) We add real-time monitoring of the actual workload, performance and resource usage. Based on the measurement results, the problem formulations can be modified and solved again for improved solutions. (3) Re-optimization of different kinds may happen at various frequencies in a nested fashion. For instance, small-scale re-optimization that occurs once every five minutes is accompanied by larger-scale re-optimization that occurs once every several hours. (4) To accommodate fast autoscaling of resources and on-demand handling of newly arrived requests, the framework incorporates incremental resource adjustment in between two adjacent re-optimization events. The last three mechanisms are also used to cope with uncertainty, incomplete information, failure and other dynamics.

We have seen that, in hierarchical decomposition, the higher-level optimization (e.g., the first-level assignment in the experiments) does not take much computation time. The computational challenge comes from the bottom-level VM-to-PM assignment subproblems. Ultimately, the sizes of these bottom-level subproblems need to be limited. Any future research that can improve that limit will be worthwhile. Improvement may come from more efficient problem formulations, more clever

customization of the integer optimization algorithms, or new algorithms.

REFERENCES

[1] C.-S. Li, B. L. Brech, S. Crowder, D. M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software defined environments: An introduction," *IBM Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.

[2] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software defined infrastructures," *IBM Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.

[3] 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 Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.

[4] 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 Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.

[5] "The Software-Defined Data Center," http://www.vmware.com/software-defined-datacenter/.

[6] S. Anthony, "Microsoft now has one million servers," *Extremetech*, July 19, 2013, http://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-amazon-says-ballmer.

[7] "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2/.

[8] VMware Inc., "VMware Capacity Planner," http://www.vmware.com/products/capacity-planner/.

[9] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," *Proceedings of IEEE Online Green Communications Conference (GreenCom)*, 2010.

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

[11] H. Jin, D. Pan, J. Xu, and N. Pissinou, "Efficient VM placement with multiple deterministic and stochastic resources in data centers," *IEEE Global Communications Conference (GLOBECOM)*, 2012.

[12] "Cisco virtualized multi-tenant data center, version 2.2 design guide," http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND.pdf.

[13] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[14] M. Chen, H. Zhang, Y. Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," *Proc. of IFIP/IEEE Integrated Network Management (IM)*, 2011.

[15] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," *Proc. of Computer Measurement Group Conference (CMG)*, 2007.

[16] 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," *Computer Networks*, vol. 57, no. 1, pp. 179–196, 2013.

[17] "Apache CloudStack Project," http://cloudstack.org/.

[18] "OpenStack Project," http://www.openstack.org/.

[19] "Eucalyptus Systems," http://www.eucalyptus.com/.

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

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

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

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

[24] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," *Proceedings of IEEE INFOCOM*, pp. 963–971, 2012.

[25] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," *Proc. of the 2008 Real-Time Systems Symposium*, 2008.

[26] P. Padala, K. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," *Proc. of ACM EuroSys*, 2009.

[27] W. Xu, X. Zhu, S. Singhal, and Z. Wang, "Predictive control for dynamic resource allocation in enterprise data centers," *Proc. 10th IEEE/IFIP Network Operation Management Symp*, April 2006.

[28] Z. Wang, "Appraise: Application-level performance management in virtualized server environments," *IEEE Trans. Network & Service Mgmt*, December 2009.

[29] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," *Proc. of IFIP/IEEE IM*, 2007.

[30] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," *Proc. of ACM/IFIP/USENIX international Conference on Middleware*, 2008.

[31] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," *Proc. of ICDCS*, June 2010.

[32] J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," *Proc. of ICAC*, 2011.

[33] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," *Proc. of IEEE HPCA*, February 2008.

[34] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," *Proc. of SIGMETRICS*, 2009.

[35] M. Wang, N. Kandasamy, A. Guez, and M. Kam, "Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters," *Proc. of ICAC*, 2006.

[36] J. Xu and J. Fortes, "Optimization in autonomic data center resource and performance management," *Technical Report, Department of Electrical and Computer Engineering, University of Florida*, 2012.

[37] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Proc. of ACM SIGCOMM*, 2008.

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

[39] M. Fielding, "Virtual CPUs with Amazon web services," June 2014, http://www.pythian.com/blog/virtual-cpus-with-amazon-web-services/.

[40] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise datacenters," in *Proceedings of WWW*, 2007.

[41] 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," *Mathematical and Computer Modelling*, vol. 58, no. 5-6, pp. 1222–1235, 2013.

[42] Gurobi, "Gurobi Web Site," http://www.gurobi.com/.