2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015

Pack up Cloud: Recursive Datacenter Resource Management and Experimental Studies

Ye Xia† Shigang Chen† Mauricio Tsugawa‡ Jose A. B. Fortes‡
† Department of Computer and Information Science and Engineering, University of Florida
‡ Department of Electrical and Computer Engineering, University of Florida

***Abstract*—Today's virtualized resource management in datacenters is typically VM-centric, which has serious problems in scalability. We propose a shift from the traditional flat, fine-grained model towards a hierarchical, abstract model that facilitates decomposition of the complex resource allocation problem in a divide-and-conquer approach. In particular, this paper elaborates a pack-centric framework in datacenter resource management, provides a case study with a recursive algorithm, and presents detailed results from experimental studies on VM-PM mapping through recursive resource allocation, which demonstrates the performance and scalability of the pack-centric framework.**

I. Introduction

Today's virtualized resource management in datacenters is typically VM-centric. Each client specifies a desired number of virtual machines (VMs) as well as resource requirements for each VM [1], including CPU, memory, storage, and possibly bandwidth, defined in deterministic terms [2]–[7] or stochastic terms using mean and variance [8], [9]. A provider's cloud system has a large number of server blades mounted on racks that are connected through layers of switches to form the datacenter network [10]. The problem of *optimal resource management* is to 1) map VMs to PMs (physical machines or servers) such that certain cost, energy or profit objectives are optimized, subject to server resource constraints.

We argue below that the traditional VM-PM mapping solutions have serious limitation in scalability. A large cloud system has many datacenters with numerous servers; for instance, the number of servers in the Amazon EC2 cloud was estimated by some to be about half a million [11]. Optimal resource management, which is often formulated as non-linear, mix-integer optimization problems, carries tremendous computational complexity [12]–[14]. The resource allocation matrix that maps VMs to servers could alone have many billions of free variables: Consider a typical resource allocation, $A = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ & \cdots & \\ a_{M1} & \cdots & a_{MN} \end{bmatrix}$, where each element has the form of $a_{ij} = \left(a_{ij}^{CPU}, a_{ij}^{mem}, ...\right)$, specifying different resources allocated to VM i from server j. In the latter case, A is called the placement matrix. Here, M is the number of requested VM instances and N denotes the number of servers. Consider an example where there are 10 datacenters, each having 10,000 servers, and they together host 200,000 VMs, each requesting five types of resources. The resource allocation matrix will have 100 billion variables.

Some existing solutions try to alleviate the computation burden by restricting to a narrower set of resource types and applying aggressive heuristics, which limit their practical values. While the major providers do not disclose their resource allocation algorithms, the performance of the simple heuristics (e.g., first-fit, round-robin, etc.) that we see in open-source middleware stacks [15]–[17] is underachieving. No prior work provides a generic, scalable framework for comprehensive resource management of arbitrary scale.

What are the reasons that limit the scalability and agility of traditional approaches? We believe a key reason is their VM-centric view, which adopts a *flat, fine-grained model*, where resource management is performed directly on individual VMs and servers. Such a fine-grained model causes the problem size to be enormous. We propose a fundamental shift toward a *hierarchical, abstract model* that facilitates decomposition of the problem in a divide-and-conquer approach. More specifically, we apply the concepts of *pack* and *swad* to provide recursively defined multi-level abstractions of client demand and cloud resources. A collection of resource-sharing VMs is modeled as a pack, and multiple packs can be further abstracted into a higher-level pack, giving rise to a hierarchical organization of VMs and packs. Similarly, the resources in the cloud are also organized into a multilevel hierarchical structure of servers, swads of servers, swads of swads, and so on. Datacenter resource management is transformed from a problem of VM-server mapping to a problem of pack-swad mapping, with the problem size being progressively reduced as more levels of pack/swad abstractions are introduced.

We recently introduced the concepts of *pack* and *swad* as a work-in-progress [18], which however lacks any recursive pack-swad assignment algorithms or substantial experimental support to justify their practical effectiveness. This paper further elaborates the pack-centric framework in datacenter resource management, provides a case study for the framework with a recursive algorithm, and most importantly presents detailed results from experimental studies on VM-PM mapping through recursive resource allocation, using a software prototype. The results show that the proposed pack-swad assignment algorithm is not only scalable (particularly when the bottom-level assignment is performed in parallel) but also achieves far better performance in terms of cost optimization than heuristics, while the VM-centric optimization cannot scale to even modest-sized problems.

978-1-5090-1893-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SmartCity.2015.185

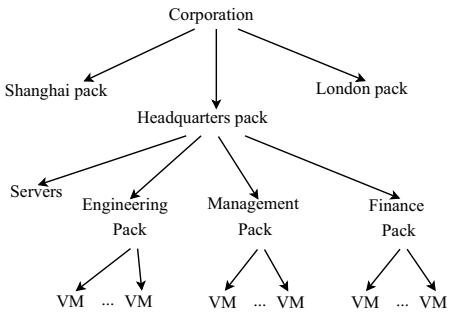

Fig. 1. Pack-based requirements

II. Pack-Centric Datacenter Resource Management

A. Overall Framework

We define a *pack* as a set of *VMs or other packs* that should be placed as a group in a datacenter for the purpose of resource sharing or performance enhancement. This recursive definition allows a client to organize its resource demand in a hierarchical structure, as illustrated by the example in Fig. 1, where a multinational corporation outsources its IT infrastructure to the cloud. The corporation has a branch in London, a branch in Shanghai, and its headquarter in San Jose, corresponding to three packs, where the headquarter pack further consists of corporate server VMs and three lower-level packs, describing the resource requirements by the management department, the finance department, and the engineering department, respectively. While each client of a cloud service provider may specify its resource requirement in packs and VMs, the provider will integrate such requirements from all its clients into a single pack hierarchy, which will be discussed shortly.

We define a *swad* as a set of servers 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 dynamically scale up its resources in real time). For example, we may use a swad to represent the servers of each rack in a datacenter. We then recursively group a certain number N of lower-level swads into higher-level swads, giving rise to a hierarchical structure (tree), with an arbitrary number of levels and with the root representing all resources owned by a cloud service provider.

Packs and swads will manifest their real value when they are used together to transform the problem of VM-server mapping into a series of much smaller problems of pack-swad mapping. Suppose a cloud service provider receives a large set of client requests in the form of packs or VMs. It will recursively group them into artificial higher-level packs to form a pack hierarchy. We give a simple algorithm that constructs a pack hierarchy with the same number of levels as the swad hierarchy. It first places the individual packs and VMs from the clients at the appropriate levels where each of them can be satisfied by a single swad at the same level. It then recursively groups the VMs and packs level by level from bottom up to create artificial packs. The resource requirement of an artificial pack is the combined requirement of its children; the grouping can be arbitrarily made as long as each artificial pack can be satisfied by one swad at its level. Note that there can be a lot more packs than swads at each level, with each pack readily fit in any swad. This paper focuses on experimental studies to justify the usefulness of the pack-centric resource management framework. We defer a detailed study on other algorithms for constructing the pack hierarchy and the swad hierarchy to a separate future work.

Starting from the top of the swad hierarchy, the pack-to-swad assignment algorithm is performed recursively. First, we map the top-level packs to the top-level swads, such that each swad has sufficient capacity to support the packs mapped to it. Then, for each swad and the packs it supports, we further map the child packs to the child swads. This process repeats recursively until the VMs are mapped to the servers. It breaks a large problem of enormous complexity into small problems that are far more manageable and can be solved in parallel. The above procedure transforms the complex problem of global resource management into a hierarchically structured one, where the resource requirements and performance constraints can be enforced iteratively from higher-level swads/packs to lower-level swads/packs with a decentralized implementation in the datacenters.

B. An Concrete Application of the Framework

We apply the above framework on a concrete problem with disk exclusivity constraints. It will be used in the experimental studies in the next section. Consider an arbitrary step in the recursive pack-to-swad assignment algorithm, with N packs to be assigned to M swads, where the packs will be actually VMs and the swads be PMs if the assignment is at the bottom level of the pack/swad hierarchies. Each pack has the following resource requirements: memory (GB), vCPU, number of local disk volumes (virtual ones) and their sizes. Similarly, each swad has a certain amount of resources in terms of memory, vCPUs, and disks. If the assignment is at the bottom level from VMs to PMs, when a VM i requests multiple local disks, there is 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 higher total disk throughput and/or better fault tolerance.

Let the sets of packs and swads 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 pack i, let α_i be the number of vCPUs required and let β_i be the memory requirement (in GiB of 2^{30} bytes). Suppose for each pack 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 swad 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 pack i to swad j, which takes the value 1 if i is assigned to j and 0 otherwise. The following

are the constraints posed by the swads' capacities.

$$\sum_{i\in\mathcal{V}} \alpha_i x_{ij} \le \beta_c C_j, \quad j \in \mathcal{P} \tag{1}$$

$$\sum_{i\in\mathcal{V}} \beta_i x_{ij} \le \beta_m M_j, \quad j \in \mathcal{P}, \tag{2}$$

$$\sum_{i\in\mathcal{V}} x_{ij}(\sum_{k\in R_i} \nu_{ik}) \le \sum_{l\in D_j} S_{jl}, \quad j \in \mathcal{P}, \tag{3}$$

$$\sum_{i\in\mathcal{V}} |R_i| x_{ij} \le \beta_d |D_j|, \quad j \in \mathcal{P}, \tag{4}$$

where $\beta_c, \beta_m, \beta_d \in (0, 1)$ are introduced to leave some slack in the swad capacity for dynamic scaling or other reasons.

If $\mathcal{V}$ is a set of VMs and $\mathcal{P}$ is a set of PMs, we need to enforce the disk exclusive constraints with binary variables y_{ikjl}, which 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:

$$y_{ikjl} \le x_{ij}, \quad i \in \mathcal{V}, j \in \mathcal{P}, k \in R_i, l \in D_j \tag{5}$$

$$\sum_{j\in\mathcal{P}} \sum_{l\in D_j} y_{ikjl} = 1, \quad i \in \mathcal{V}, k \in R_i \tag{6}$$

$$\sum_{j\in\mathcal{P}} x_{ij} = 1, \quad i \in \mathcal{V} \tag{7}$$

$$\sum_{k\in R_i} y_{ikjl} \le 1, \quad i \in \mathcal{V}, j \in \mathcal{P}, l \in D_j \tag{8}$$

$$\sum_{i\in\mathcal{V}} \sum_{k\in R_i} \nu_{ik} y_{ikjl} \le S_{jl}, \quad j \in \mathcal{P}, l \in D_j. \tag{9}$$

The condition (5) 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 (6) ensures that every requested virtual disk must be assigned to exactly one physical disk. The condition (7) ensures that every VM is assigned to exactly one PM. The condition (8) ensures that VM i cannot have more than one virtual disks assigned to the same physical disk. The condition (9) is the disk capacity constraint.

The costs and performance objective will ultimately be decided by the cloud provider. For concreteness, we assume that a certain operational cost $\hat{c}_j$ is incurred for a swad j if some pack(s) is assigned to it; otherwise, there is zero cost involved. Let z_j be a 0-1 variable indicating whether swad j is used by some packs. 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 $B = N$).

$$z_j \le \sum_{i\in\mathcal{V}} x_{i,j}, \quad j \in \mathcal{P} \tag{10}$$

$$Bz_j \ge \sum_{i\in\mathcal{V}} x_{i,j}, \quad j \in \mathcal{P}. \tag{11}$$

The optimization objective is to minimize the total operation cost, which leads to profit maximization,

$$\min_{x,y,z} \sum_{j\in\mathcal{P}} \hat{c}_j z_j. \tag{12}$$

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

III. EXPERIMENTAL STUDIES

To demonstrate the effectiveness of the pack-based framework in scalability and performance improvement, we present the results from experimental studies on VM-PM mapping based on the application of the framework in Section II-B.

A. Experimental Setup

• *VM Types:* We take a subset of the allowed VM types (classes) of Amazon's EC2 [19]. Their resource requirements are shown in Table I.

• *PM Types:* Cloud providers generally don't disclose the capabilities of all their PMs. For our experiments, we use the PM types of Amazon EC2 in Table II, whose details are largely our guess (based on the information revealed on Amazon's web site).

• *Greedy randomized heuristic algorithm for comparison:* 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 [20]–[22] but should achieve much lower costs than the

TABLE III
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

latter due to two exhaustive search steps. 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. 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. For each scanned PM, our heuristic algorithm checks whether it is possible to assign the currently considered VM to that PM. For disk assignment, the algorithm exhaustively enumerates different disk assignment possibilities and uses the first one that is feasible.

• *VM-PM Mapping:* The experiments are to assign 1000 VMs to 1000 PMs of different types. This problem size is way too big for a flat VM-centric optimization solution that directly maps VMs to PMs without pack and swad hierarchies. The experimental results of applying the optimization of Section II-B on VMs and PMs directly are shown in Table III.

• *Setting for Pack-Swad Assignment Algorithm:* We use the recursive algorithm in Section II-B with two-level hierarchies. Packs and Swards are constructed as follows. We split the VMs randomly into 25 packs, each containing different types proportional to their respective total numbers, and split the PMs randomly into 25, each containing different types proportional to their respective total numbers. 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. We set the default values of β_c, β_m, and β_d to zeros, but for the first-level pack-swad assignment, we let $0 < \beta_d \leq 1$ — we stipulate that the total number of disks requested by all the packs assigned to a swad is no more than β_d 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_d in the first-level optimization, we can spread out packs more across the swads to gain more room for maneuver.

B. Performance Evaluation and Comparison

The results of the experiments are summarized in Table IV.

1) Mix 1: The mix of VMs and PMs is described in Table V. At the first-level pack-swad assignment, $\beta_d = 0.7$. The pack-swad assignment 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 pack-swad assignment algorithm achieves about half the cost as that of the randomized heuristics.

TABLE IV
SUMMARY OF RESULTS: PACK-SWAD ASSIGNMENT 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	

TABLE V
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
		l6	0
i2.xlarge	0		
i2.2xlarge	0		
i2.4xlarge	0		
i2.8xlarge	0		

Next we 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.

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 75 seconds per swad. Note that these 17 different assignment subproblems are completely independent and could run in parallel on different computers. There is variability in the running times for different swads, due to different problem sizes. The running times are shown in Fig 2, sorted in increasing order. Overall, we see that the optimization that assigns VMs to PMs at the bottom level of hierarchies 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.

The heuristic algorithm took a fairly long time, hundreds of seconds per run. It computation time scales with the product of the total number of VMs and the total number of PMs.

We make additional comments about the experimental results. In the optimal solution for the first-level assignment,

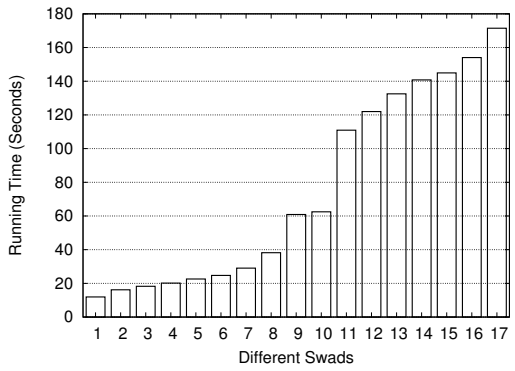

Fig. 2. Sorted running times for VM-to-PM assignment for different swads; mix-1

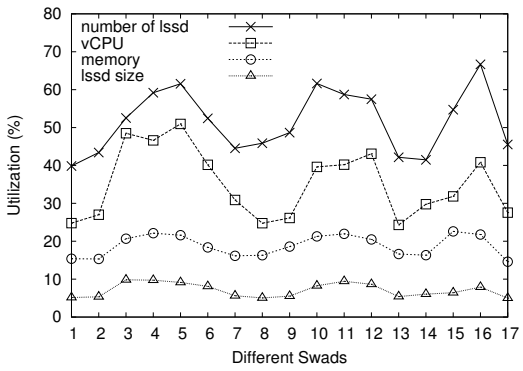

Fig. 3. Resource utilization for different swads; mix-1

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. In the setting of our experiments, the vCPUs tend to be the resource bottleneck. 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. 3. 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 the memory are seriously under-utilized, at around 10% and 20%, respectively.

2) Mix 2: The mix of VMs and PMs is described in Table VI. At the first-level pack-swad assignment, $\beta_d = 0.7$.

The pack-swad assignment 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. 4.

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 pack-swad assignment algorithm.

TABLE VI
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		

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 V. At the first-level pack-swad assignment, $\beta_d = 0.7$. 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 III-B1 (1281 and 75 seconds, respectively).

The total cost achieved by pack-swad assignment is $98,040$, still a big improvement over the randomized heuristics, which has a cost of $150,573$.

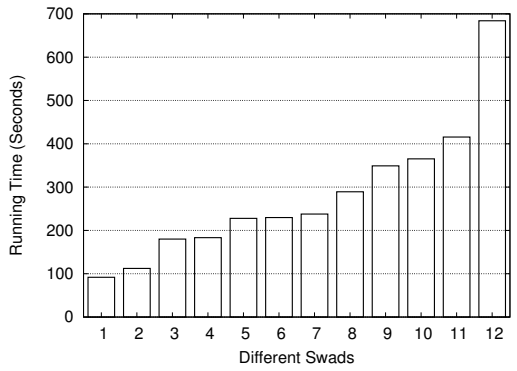

Fig. 4. Sorted running times for VM-to-PM assignment for different swads; mix-2

TABLE VII
CONTROLLING NUMBER OF SWADS USED BY β_d

β_d	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

4) 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_d. The results after the first and second-level optimization are shown in Table VII. As β_d 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_d decreases. Also, the number of PMs used after the second-level optimization increases as β_d 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_d decreases, cheaper PMs tend to be used and more expensive PMs tend to be avoided, resulting in a lower total cost. As β_d continues to decrease, the first-level optimization problem will eventually become infeasible.

IV. CONCLUSION

We present a pack-centric framework, combined with integer programming formulations and hierarchical decomposition, for datacenter resource management. We also give a case study for the framework with a recursive pack-swad assignment algorithm. Our extensive experimental results show that the proposed algorithm is not only scalable (particularly when the bottom-level assignment is performed in parallel) but also achieves far better performance than heuristics.

REFERENCES

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

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

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

[4] V. Inc., "VMWare vCenter CapacityIQ," http://www.vmware.com/products/vcenter-capacityiq/.

[5] "IBM WebSphere CloudBurst," http://www-01.ibm.com/software/webservers/cloudburst/.

[6] "Lanamark Suite," http://www.lanamark.com/.

[7] "Novell PlateSpin Recon," http://www.novell.com/products/recon/.

[8] 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.

[9] 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.

[10] "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.

[11] H. Liu, "Amazon data center size," March 2012, http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/.

[12] 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.

[13] 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.

[14] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and J. Guofei, "Power and Performance Management of Virtualized Computing Environments via Lookahead Control," *Proc.of ICAC*, 2008.

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

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

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

[18] Y. Xia, M. Tsugawa, J. Fortes, and S. Chen, "Hierarchical mixed integer programming for pack-to-swad placement in datacenters (working-in-progress)," *Proceedings of 12th IEEE International Conference on Autonomic Computing*, 2015.

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

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

[21] 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.

[22] 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.