

Shares and Utilities based Power Consolidation in Virtualized Server Environments

Michael Cardosa
University of Minnesota
cardosa@cs.umn.edu

Madhukar R. Korupolu
IBM Almaden Research Center
madhukar@us.ibm.com

Aameek Singh
IBM Almaden Research Center
Aameek.Singh@us.ibm.com

Abstract—Virtualization technologies like VMware and Xen provide features to specify the minimum and maximum amount of resources that can be allocated to a virtual machine (VM) and a *shares* based mechanism for the hypervisor to distribute spare resources among contending VMs. However much of the existing work on VM placement and power consolidation in data centers fails to take advantage of these features. One of our experiments on a real testbed shows that leveraging such features can improve the overall utility of the data center by 47% or even higher.

Motivated by these, we present a novel suite of techniques for placement and power consolidation of VMs in data centers taking advantage of the min-max and shares features inherent in virtualization technologies. Our techniques provide a smooth mechanism for power-performance tradeoffs in modern data centers running heterogeneous applications, wherein the amount of resources allocated to a VM can be adjusted based on available resources, power costs, and application utilities. We evaluate our techniques on a range of large synthetic data center setups and a small real data center testbed comprising of VMware ESX servers. Our experiments confirm the end-to-end validity of our approach and demonstrate that our final candidate algorithm, PowerExpandMinMax, consistently yields the best overall utility across a broad spectrum of inputs – varying VM sizes and utilities, varying server capacities and varying power costs – thus providing a practical solution for administrators.

I. INTRODUCTION

Modern server virtualization technologies are driving a transformation in enterprise data centers. By consolidating multiple physical bare-metal servers into fewer virtualized machines, enterprises are improving resource utilizations and reducing operational costs. There exist many tools like CiRBA [1] that assist enterprises in devising an appropriate consolidation plan for their IT environment. In the research community as well, there are many proposed techniques [2], [3], [4] that analyze utilizations of physical servers and output a consolidation plan describing which combinations of physical servers can be virtualized into Virtual Machines (VMs) and *placed* on each virtualized server.

However, all of these fail to take advantage of the important *min*, *max* and *shares* parameters. Most virtualization technologies like VMware [5], Xen [6] provide administrators with the ability to set the minimum amount of resource required for a VM (*min*, also referred to as a *reservation*), a maximum allowable resource (*max*, also called a *limit*) and its share of spare resource (*shares*, also referred as the *weight* in Xen schedulers). These are useful constructs that ensure intelligent distribution of resources between different applications.

A. *Min, Max and Shares*

Not all applications are created equal. In a typical enterprise data center, there are a variety of co-existing high priority applications like company’s e-commerce web server and low priority applications like the intranet blogging server. Under situations of high load, valuable CPU resources are best utilized when allocated to the high priority applications instead of the low priority ones. Along with different priorities, different applications in the data center have a different affinity for each resource. For example, a web server may value additional CPU cycles much more than a storage backup application. In such scenarios as well, CPU resources are best allocated to the higher utility application – the web server in this case.

Most current research tackles this challenge only through the placement of VMs - which combination of application VMs are placed on each server. However, once placed all VMs on the server contend equally for resources in spite of their different priorities and resource affinities. This lack of granularity in resource allocation is a fundamental drawback which can significantly impact the utility gained from the system.

Surprisingly, current techniques are lacking even though all major virtualization vendors provide enabling capabilities with parameters like *min*, *max* and *shares*. Setting a *min* for a VM ensures that it receives at least that amount of resources when powered on and setting a *max* for a low-priority application ensures that it does not use more resources, thus keeping them available for high-priority applications. *Shares* provide advice to the virtualization scheduler on how to distribute resources between contending VMs. A CPU shares ratio of 1:4 between low and high priority VMs informs the scheduler to give 4 CPU cycles to the high priority VM for each CPU cycle given to the low priority one.

1) *Min, Max and Shares Impact Experiment*: To illustrate the impact of *min*, *max* and *shares*, we conducted an experiment on a testbed of 3 VMware ESX servers with 12 VMs (running RedHat Enterprise Linux). The VMs were randomly classified into low priority and high priority VMs (6 each) and the utility of each VM was measured as the amount of CPU resources it received multiplied by a factor that differentiates VMs based on their priority¹. For this experiment, we used a factor of 4 for high priority VMs and 1

¹The complete experimental setup is described in Section IV-B

for low priority VMs). We created two different load situations - (i) *Low Load*, in which VMs cumulatively only desired 35% of the total available CPU capacity, and (ii) *High Load*, in which VMs desired 100% of the total capacity. Figure 1 shows the performance of (a) the BASE technique that does not utilize min, max or shares, (b) the MM technique, which only used the min and max parameters, the latter to limit the consumption of low priority applications and (c) the MMS technique that used all three parameters.

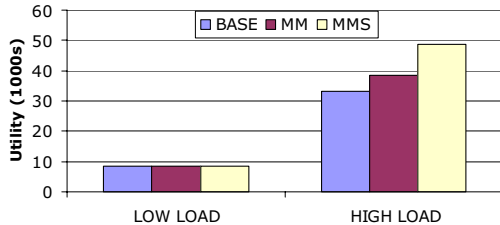


Fig. 1. Min, Max and Shares Impact: Under high load conditions, MMS technique delivers 47% more utility than BASE

As shown in Figure 1, under low load conditions all techniques perform equally well as each VM is able to get all the resources that it desires. Under high load conditions the MM technique performs better than the BASE technique by limiting low priority VMs to a fixed level of resource usage. The MMS technique performs the best outperforming BASE by nearly 47% by limiting resource usage of low-priority VMs and by further guiding the VM scheduler to discriminate between high priority and low priority VMs using *shares*. Note that this percentage gain can be even higher for application priorities ratios of greater than four.

B. Power Based Consolidation

While it can be argued that *min*, *max* and *shares* are useful only under high load conditions, it is important to note that such conditions will occur very frequently in modern virtualized data centers. According to industry analysts, enterprise data centers have doubled their energy use in the past five years [7] and IDC predicts that power costs will equal hardware acquisition costs by 2012 [8]. As a result, enterprises are under immense pressure to reduce power consumption. This increased emphasis on reducing power costs will result in a large number of physical servers being consolidated into each virtualized server.

The research community has also noticed this trend and techniques have been proposed that tradeoff performance for power savings [2], [9], [10], [11]. However, none of these approaches utilize the *min*, *max* and *shares* mechanisms for intelligent power-performance tradeoffs. As a result, these techniques are unable to differentiate between applications and once consolidated each VM receives equal resources.²

²Repeating the above experiment, by consolidating all VMs into a single server under low load conditions, the MMS technique lost only 4% of total utility while using 67% fewer servers. In contrast the BASE technique lost over 18% of total utility.

C. Our Contributions

This paper makes the following contributions:

- **Min, Max and Shares:** To the best of our knowledge, this work is the first ever to exploit the *min*, *max* and *shares* parameters for VM placement and power consolidation in data centers.
- **Resource allocation techniques:** We present a suite of techniques that address the VM placement and resource allocation problem with increasing degrees of effectiveness. We provide detailed analysis for each one and use the insights to design a final recommended one that combines the best of all.
- **Comprehensive experimental evaluation:** We provide a comprehensive experimental evaluation using a range of synthetic data center setups as well as a real data center testbed comprising of VMware ESX 3.5 servers.

II. PROBLEM FORMULATION

Consider an IT environment where we are given a set of VMs that need to be placed among a set of physical servers in the data center. Let the set of VMs in the environment be denoted by $\mathcal{V} = \{V_1, V_2, \dots, V_{|\mathcal{V}|}\}$. Each VM requires a certain amount of CPU resource on the machine where it is placed. In this paper, we focus on the CPU resource only. For a virtual machine V_i , $1 \leq i \leq |\mathcal{V}|$, we use $V_i.m$ and $V_i.M$ to denote the respective minimum and maximum amounts of resources for V_i as defined by the *min* and *max* parameters. For example, VM V_1 may have $V_1.m = 500$ Mhz and $V_1.M = 2000$ MHz.

We capture differences in application priority and resource affinity using a utility function for each application. The utility function specifies the amount of utility (e.g. dollar value) that the application VM can generate for a given amount of resource allocated. For VM V_i , we use $V_i.u$ and $V_i.U$ to denote the utility derived from the VM when it is allocated $V_i.m$ or $V_i.M$ amount of resources respectively. For points in between $V_i.m$ and $V_i.M$ we use a linear interpolation to derive the utility value. The usefulness of the utility function is best illustrated with an example. Consider two virtual machines, V_1 with $\langle V_1.m, V_1.M, V_1.u, V_1.U \rangle$ as $\langle 500 \text{ MHz}, 2000 \text{ MHz}, 1000 \text{ units}, 2000 \text{ units} \rangle$ and V_2 with $\langle V_2.m, V_2.M, V_2.u, V_2.U \rangle$ as $\langle 500 \text{ MHz}, 2500 \text{ MHz}, 500 \text{ units}, 1000 \text{ units} \rangle$. For an available CPU capacity of 2500 MHz, the best allocation is to allocate 2000 MHz to V_1 and 500 MHz to V_2 generating a total utility of 2500. We discuss how to obtain the min-max and utility inputs in Section IV.

In the rest of the section we address the question: given the above resource requirements and utilities for each VM, how can we best decide a placement of the VMs on the available physical servers and estimate the *shares* that should be given to each VM? The *shares* serve as an advice to the hypervisor scheduler on how to distribute the spare resources among the contending VMs on the server. The scheduler first gives the minimum amount to each VM (assuming the sum of the minimums of the VMs on the server is less than the capacity

of the server) and then the remaining amount is distributed among the VMs in proportion to their shares. For the example mentioned above, giving a larger proportion of shares to V_1 restricts V_2 to 500 MHz while V_1 uses 2000 MHz.

Let $\mathcal{S} = \{S_1, S_2, \dots, S_{|\mathcal{S}|}\}$ denote the set of physical servers in the data center. For server S_j , we use C_j to denote the CPU capacity of the server. We also use a parameter P_j to denote the power cost for the server. We say that power cost P_j is *incurred* if server S_j is turned *on* in the placement solution. Note that P_j can be used to represent other costs as well besides the power cost of keeping S_j on: for example, any additional management overheads or depreciation costs associated with S_j .

The allocation of VMs to the servers can be represented using an allocation function $A : \mathcal{V} \rightarrow \mathcal{S} \times \mathbb{R}^+$, where $A(V_i)$ represents a pair capturing the node S_j to which V_i has been allocated and the amount x_i of the resource on S_j that has been allocated to V_i . In this case, we denote $A(V_i)$ as $\langle S_j, x_i \rangle$. The first component is referred to as $A(V_i).Server$ and the second component as $A(V_i).Size$. If V_i is not placed under A , then $A(V_i).Server$ is considered to be *null*. If placed, note that $A(V_i).Size$ must satisfy $V_i.m \leq A(V_i).Size \leq V_i.M$. Next, we formally develop the problem objective.

We denote the set of VMs allocated to server S_j under the allocation function A by $VSet_A(S_j)$.

$$VSet_A(S_j) = \{V_i : A(V_i).Server = S_j\}$$

The subscript A is dropped when there is no ambiguity. The utility earned by a node S_j under allocation A is denoted by

$$Util_A(S_j) = \begin{cases} 0 & \text{if } VSet_A(S_j) \text{ is empty} \\ \sum_{V_i \in VSet_A(S_j)} V_i.u(A(V_i).Size) & \text{otherwise} \end{cases}$$

where the latter quantity captures the utility provided by application VM V_i when its allocation is of size $A(V_i).Size$.

The aim of the consolidation process then is to find a suitable allocation function A that maximizes the overall utility of the system minus the power costs incurred. The formulation then becomes, find $A : \mathcal{V} \rightarrow \mathcal{S} \times \mathbb{R}^+$ so as to

$$\text{maximize} \quad \left(\sum_j Util_A(S_j) - \sum_{j: VSet_A(S_j) \neq \emptyset} P_j \right)$$

subject to

$$\begin{aligned} V_i.m \leq A(V_i).Size \leq V_i.M & \text{ for each } V_i \in \mathcal{V} \\ \sum_{V_i \in VSet(S_j)} A(V_i).Size \leq C_j & \text{ for each } S_j \in \mathcal{S} \end{aligned}$$

where i and j are indexes over the set of VMs and servers respectively ($1 \leq i \leq |\mathcal{V}|$, $1 \leq j \leq |\mathcal{S}|$).

A. Problem Complexity

It is important to understand the complexity of this problem. Compared to the traditional formulations of VM placement problem as a multi-knapsack problem, the min-max and shares version can be thought of as a unique multi-knapsack problem in which items packed into the knapsack can be *elastic* in size between min and max with shares dictating the precise

Algorithm 1 BASICOVERPROVISION(VMS \mathcal{V} , SERVERS \mathcal{S})

Output: $A : \mathcal{V} \rightarrow \mathcal{S} \times \mathbb{R}^+$

- 1: Sort servers by increasing power rate, $\frac{P_j}{C_j}$
 - 2: **for** $i = 1 \dots |\mathcal{V}|$ **do**
 - 3: Set $j \leftarrow$ first node such that $\sum_{V_i \in VSet(S_j)} A(V_i).Size + V_i.M \leq \frac{9}{10} C_j$
 - 4: Set $A(V_i) = \langle S_j, V_i.M \rangle$
 - 5: **end for**
 - 6: **return** A
-

size of the item. The goal is to identify the best size for each item along with choosing an appropriate knapsack to use in order to maximize the overall utility. This is a new and challenging problem that to the best of our knowledge has not been addressed before.³

III. ALGORITHMS

In this section we present a suite of algorithmic techniques for addressing this min-max and shares aware placement problem. The insights from each technique lead into the next technique eventually culminating in our recommended *PowerExpandMinMax*.

Basic Strategy: A basic strategy, one that is often used by administrators in practice in the absence of detailed techniques and methods, is to pack all the virtual machines at their maximum requirement and leave a certain amount of room in each server to account for future growth of the VMs. We call this the *BasicOverprovision* (BO) strategy (seen in Algorithm 1) and use it as a baseline to compare with other algorithms. It packs VMs in a first-fit manner among the available servers. We use a $\frac{1}{10}$ th overprovisioning factor for the sake of concreteness, i.e., each server is filled to (at most) $\frac{9}{10}$ ths of its capacity.

BO is still power-aware, in the sense that it sorts the nodes in increasing order of their power rates. The power rate of a server S_j is defined to be the power cost per unit capacity of the server, i.e., $\frac{P_j}{C_j}$. This sorting helps the algorithm to select nodes with lower power cost per unit capacity first before considering higher ones.

A downside of BO, however, besides its conservative use of only $\frac{9}{10}$ ths of each server's capacity, is that it ignores the relative values of the different virtual machines. In cases where there is not enough server capacity to fit all the VMs, then it would still choose the first few VMs even though there may be higher utility ones later.

GreedyMax Algorithm: To account for that, we introduce the *GreedyMax* (GM) strategy which sorts the VMs by their **profitability at max**, i.e., $\frac{V_i.U}{V_i.M}$ and places VMs in that order in a first-fit fashion, allocating the maximum requested size for each VM. The profitability metric captures the utility of

³One problem that can be considered relevant is the Multiple-Choice Knapsack Problem (MCKP) [12]. However, it is a poor fit for large-scale data center environments.

a VM per unit resource that is allocated to it and is useful in identifying VMs that provide higher utility for every unit of resource that they consume. Sorting the VMs in decreasing order of their profitabilities enables GM to do much better than BO especially when the number of server nodes is small.

A limitation of GM is that it always assigns each VM at its maximum requested allocation. This discards the possibility that some VMs may be more profitable at another smaller size and allocating at that size may yield a better overall utility by leaving room for more VMs.

GreedyMinMax Algorithm: The *GreedyMinMax* (GMM) algorithm builds on GM by considering both the min and max points for each VM. It introduces two items, V_i^{min} and V_i^{max} for each virtual machine V_i : the first one an item of size $V_i.m$ and utility $V_i.u$ whereas the second one an item of size $V_i.M$ and utility $V_i.U$. Beginning with a set of $|\mathcal{V}|$ VMs, GMM now has a set of $2|\mathcal{V}|$ items which we call $\bar{\mathcal{V}}$. It sorts the items of $\bar{\mathcal{V}}$ in decreasing order of profitabilities and starts placing them one by one in a first fit manner among the available nodes, with one caveat: when it places an item, say one corresponding to V_i^{min} , then it removes the other one (i.e., V_i^{max}) from the list and vice-versa. Thus ensuring that only one of the two V_i items gets placed for each VM.

ExpandMinMax Algorithm: GMM does well at picking the right combination for each VM especially when some VMs are more profitable at their min while some are more profitable at their max. However, it still misses out on two qualitative phenomenon. One, while the VMs which are more profitable at min get placed at min, they do not get a chance to get additional resources even when the incremental value they provide beyond min is much higher than other VMs. Second, it misses out when some nodes still have room left after the first-fit packing. A more efficient approach would be to expand one or more VMs to use that space on each node.

Instead of selecting the node for each VM in a first-fit fashion, the *ExpandMinMax* (EMM) algorithm first computes an estimated utility for each node if the new VM were added to that node and selects the node that gives the best utility improvement. The utility for each node is computed by first setting all the VMs assigned there to be at min, and then *expanding* the VMs that give the most incremental utility per unit capacity until either the node's capacity is reached or no more expansion is possible.

To make it more concrete, we call a set of VMs Q *feasible* for a node S_j if the minimum requirements of the VMs in Q add up to less than the capacity of S_j . Formally, $feasible(S_j, Q)$ iff $\sum_{V_i \in Q} V_i.m \leq C_j$. Once a set of VMs Q is feasible for S_j , Algorithm 2 gives a method of estimating the utility of node S_j using the set of VMs Q .

Instead of assigning VMs in a first-fit manner, EMM assigns each VM to the node S_j that maximizes the utility gain, i.e., $NodeUtility(S_j, Set(S_j) + V_i) - NodeUtility(S_j, Set(S_j))$ where $Set(S_j)$ is the set of VMs currently assigned to S_j . If

Algorithm 2 NODEUTILITY(SERVER S_j , SET OF VMs Q)

```

1:  $remCapacity = C_j - \sum_{V_i \in Q} V_i.m$ 
2: return -1 if ( $remCapacity < 0$ )
3:  $nodeUtility = \sum_{V_i \in Q} V_i.u$ 
4: Sort VMs in  $Q$  in decreasing order of  $\frac{V_i.U - V_i.u}{V_i.M - V_i.m}$ 
5: for  $i = 1 \dots |Q|$  do
6:   Let the  $i$ th item of  $Q$  correspond to VM,  $V_i$ .
7:   if ( $remCapacity > 0$ ) then
8:      $\delta = \min(V_i.M - V_i.m, remCapacity)$ 
9:      $remCapacity - = \delta$ 
10:     $nodeUtility + = V_i.util(V_i.m + \delta) - V_i.u$ ;
11:   else
12:     return  $nodeUtility$ 
13:   end if
14: end for
15: return  $nodeUtility$ 

```

nothing has been assigned to S_j yet, $Set(S_j)$ is considered to be empty and the corresponding NodeUtility is zero. Note that EMM implicitly expands VMs on each node to realize the maximum utility possible from them.

PowerExpandMinMax Algorithm: A limitation of EMM, however, is that it tends to use all the servers that it has access to. For example, if an additional empty server were available, EMM would likely pick that server because placing the new VM there would give a better utility gain as it would not have to shrink any existing VMs on other nodes. This can be a disadvantage in terms of power costs if it opens more nodes than necessary. The ideal algorithm would automatically detect the “sweet spot” number of nodes and use only that many nodes even if additional ones are available. However detecting whether to start a new node or not – based on local knowledge only (placement of the current VM) without a look-ahead for VMs being packed later – is a tricky question.

The *PowerExpandMinMax* (PEMM) technique we propose uses the following strategy to address this aspect. It is similar to EMM, except that the comparative measure it uses is the node utility gain minus the proportional power cost incurred (if any) on the new node. The latter quantity applies only when a new node is being started, and in this case the power cost of the machine is scaled down to a size used by the VM and that is considered the proportional power cost for the VM.

More formally, if S_j were an already opened node, i.e., $Set(S_j) \neq \emptyset$, then the net gain $NetGain(S_j, V_i)$ if V_i were also added to S_j is given by

$$NodeUtility(S_j, Set(S_j) + V_i) - NodeUtility(S_j, Set(S_j))$$

and if S_j were a new node, then

$$NetGain(S_j, V_i) = NodeUtility(S_j, \{V_i\}) - x_i \times \frac{P_j}{C_j}$$

where x_i is the size allocated for V_i on S_j as dictated by the $NodeUtility(S_j, \{V_i\})$ function. Note that this may sometimes result in a negative net gain, especially when the proportional power cost of the new node exceeds the utility gain from the VM, in which case the algorithm may choose

Algorithm 3 POWEREXPANDMINMAX(VMS \mathcal{V} , SERVERS \mathcal{S})

Output: $A : \mathcal{V} \rightarrow \mathcal{S} \times \mathbb{R}^+$

```
1: Sort server nodes by increasing power rate,  $\frac{P_i}{C_j}$  and set  $\bar{\mathcal{V}} \leftarrow \emptyset$ 
2: for  $i = 1 \dots |\mathcal{V}|$  do
3:   Set item  $V_i^{min}$  to be of size  $V_i.m$  and utility  $V_i.u$ 
4:   Set item  $V_i^{max}$  to be of size  $V_i.M$  and utility  $V_i.U$ 
5:    $\bar{\mathcal{V}} \leftarrow \bar{\mathcal{V}} \cup \{V_i^{min}, V_i^{max}\}$ 
6: end for
7: Sort  $\bar{\mathcal{V}}$  in decreasing order of profitabilities.
8: while  $\bar{\mathcal{V}}$  not empty do
9:   Remove the first entry of  $\bar{\mathcal{V}}$ , say it corresponds to  $V_i$ .
10:  for (feasible  $S_j \in \mathcal{S}$ ) do
11:    Compute  $NetGain(S_j, V_i)$  as given in the formula
12:  end for
13:   $j \leftarrow \text{argmax}_{feasible S_j \in \mathcal{S}} NetGain(S_j, V_i)$ 
14:  if  $feasible(S_j, Set(S_j) + V_i)$  then
15:    Assign  $V_i$  to node  $S_j$ 
16:    for VMs  $V_k \in Set(S_j) + V_i$  do
17:      Set  $A(V_k) \leftarrow \langle S_j, x_k \rangle$  where  $x_k$  is as dictated by the
       $NodeUtility(S_j, Set(S_j) + V_i)$  function
18:    end for
19:    Remove the corresp. complementary entry of  $V_i$  from  $\bar{\mathcal{V}}$ 
20:  end if
21: end while
22: return  $A$ 
```

a different (already opened) node that provides better utility gain. However if all opened nodes yield much smaller (more negative) utility gain, then the new node will still get chosen. The latter can happen, for example, if adding new VM to an already opened node causes more valuable VMs to be compressed.

The pseudocode for PEMM is in Algorithm 3. We expect PEMM to do well across a broad range of scenarios including not starting up more nodes than necessary.

Hypothetical Upper Bound Algorithm (HUB): For completeness, we decided to compare our algorithms against a hypothetical “upper bound” algorithm. We relax the multi-bin packing constraint by “lining up the servers end-to-end” and allowing single VMs to be placed over multiple servers at once. We order VMs and servers in a greedy fashion, charging only for each VM’s fractional power consumption over its respective machine(s). This allows achieving 100% capacity usage on each machine thus providing an upper bound over any real placement.

IV. EXPERIMENTAL EVALUATION

In this section, our goal is to evaluate the techniques presented so far: in terms of (a) their *quality* in deriving most utility, (b) *robustness* in providing good results even with varying data center configurations and VM sizes, and (c) *scalability* in dealing with large data center setups. Our other aim is to validate the end-to-end effectiveness of our overall approach in real installations. For this, we use a combination of experiments conducted on synthetic data center setups as well as a real datacenter testbed with VMware ESX 3.5 servers.

A. Large Synthetic Data Center Experiments

Synthetic experiments allow us to study the impact and performance of techniques in large data center environments. Not having access to such large *live* data centers, we used a simulator written in Java to generate installations with thousands of VMs and hundreds of servers, with configurable sizes, utilities and min-max inputs.

Utility and Min-Max Inputs: A natural question that arises is how to get the utility and min-max inputs required for the algorithms. In real installations, management interfaces such as VMware Virtual Center [13] provide a mechanism for administrators to specify the min-max values for each VM they create. Administrators set these numbers based on either their expert knowledge or from analyzing templates and traces of running applications. These can also be derived from application Service Level Agreement (SLA) information [14], [3]. Similarly, application management interfaces such as IBM EWLM (Enterprise Workload Manager) [15] provide mechanisms for administrators to specify priorities for each application. We envision extending such interfaces to specify a utility value as well for each application. Our focus in this paper is not as much on how to obtain these inputs but more on how to use the inputs effectively to derive meaningful placement and consolidation plans.

In our synthetic experiments, to be able to generate a broad controllable range of these inputs, we use normal distributions with appropriate mean and standard deviation values. The $V_i.m$ values are first generated using a normal distribution followed by the $V_i.M - V_i.m$ values. This combination provides a mix of small, large and medium sized VMs. Increasing mean leads to tighter packings and increasing standard deviation leads to greater variability among the VMs. The $V_i.u$ values and the $V_i.U - V_i.u$ values are also generated in a similar fashion using configurable mean and standard deviation values, default being 15 and 10 for the mean, and 5 and 2 for the standard deviation. Node sizes are set to be a configurable multiple (default 8) of the mean $V_i.m$ size.

In a real system, power costs would be obtained from the energy ratings on the machines or from a management tool such as IBM Active Energy Manager [16]. For our experiments, to avoid excessive parameterization, we use an intuitive economics-motivated model. We consider a data center that uses the *BasicOverprovision* technique and assume that under such VM placement a percentage of total utility received (from the placed VMs) was spent by the data center towards power cost. We run our experiments by varying this percentage with the default being 90%. This enables us to capture the power cost expenditure of a data center as a percentage of the revenue generated by the overprovision strategy.

1. Quality of Techniques: In our first experiment, we evaluate the utility derived by the different techniques – BO, GMM, EMM, PEMM, along with the hypothetical upper bound technique HUB – with 1600 VMs and varying number of available servers. We vary the number of servers from 10 to 450 and measure

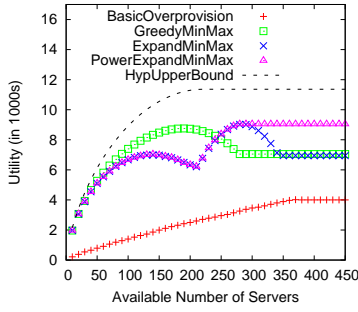


Fig. 2. Utility with increasing number of available servers: PEMS delivers the best utility among valid placements.

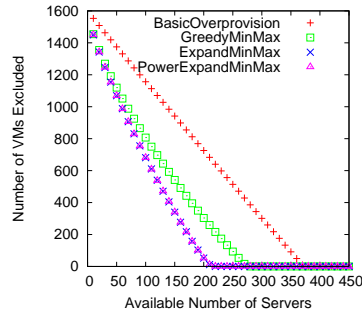


Fig. 3. Number of VMs excluded: EMM and PEMS fit all VMs the earliest.

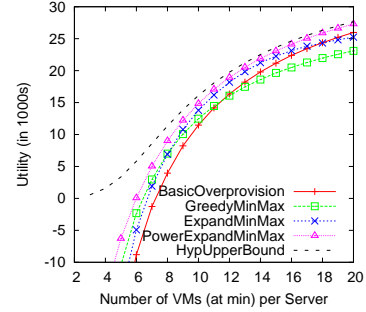


Fig. 4. Increasing Server to VM size ratio: PEMS delivers the best valid placement decisions

how well each technique performs in terms of the total utility obtained. As shown in Figures 2 and 3, this experiment illustrates many interesting aspects of our algorithms.

When the number of servers is few, all algorithms exclude some VMs as shown in Figure 3. BO excludes a large number of VMs and also derives a low utility, primarily because it packs VMs at max that too in a first-fit manner without prioritization. The *MinMax* algorithms (GMM, EMM, PEMS) derive better utility because of their ability to pack VMs at other sizes and also because of their prioritization of VMs. However, as seen in Figure 3, in this stage all algorithms leave many VMs out up to 150 servers.

As we move closer to 150-200 servers, the *MinMax* algorithms attempt to place more VMs, possibly by shrinking resources given to high-profitable VMs in favor of fitting more, even low-profitable VMs. This leads to their lower utility gain or even a drop in some cases. Of the three *MinMax* algorithms, GMM continues to exclude more VMs thus dropping in utility much later. As we aim to place as many VMs as possible, PEMS and EMM are the best performers and are able to fit all VMs the earliest.

As the number of servers increases further (e.g. 200 to 300) the total utility for PEMS increases sharply as there is now more room to expand high-utility VMs. Finally, as the number of servers approaches 300-350, all algorithms except BO place every VM. This is the important “*steady*” state that will occur in enterprise data centers once they have enough available resources. In this state, both GMM and EMM plateau much lower than PEMS as they do not adequately tradeoff power costs. In contrast, PEMS plateaus at its highest point indicating its ability to do a smooth performance-power tradeoff.

The hypothetical upper bound, is not a practical valid placement as it relaxes some of the problem constraints. In general, of all the valid techniques, PEMS provides the most utility over all server number combinations. Most importantly, it has the maximum utility achieved in the steady state. This illustrates the ability of PEMS to choose the “*sweet-spot*” number of servers needed for consolidating VMs. In contrast, EMM continues to expand as more servers become available even though it loses overall utility.

In the rest of the experiments, *we will provide a steady state*

analysis only, essentially capturing the plateau in Figure 2 and changing other parameters to show the robustness of the algorithms.

2. Robustness of Techniques: In the next set of experiments, we vary various parameters of our experiments to test the robustness of these techniques under varying configurations.

a. Varying Server to VM Size Ratio: In Figure 4, we vary the sizes of the servers with respect to the size of the average VM. A higher ratio means that each server is able to accommodate more VMs. At lower numbers, more servers are needed and hence the power cost rises leading to lower, even negative, overall utility. As the ratio increases, fewer servers are needed and the utility goes up. Throughout the range, PEMS yields the highest utility placements tracking closely with HUB.

b. Increasing Heterogeneity: So far, all servers in our experiments were homogeneous with a certain capacity. In Figure 5, we vary the capacities of the servers. As the standard deviation of server capacities increases, certain servers will have high capacity, hence lower power to capacity rate and will get filled first. The overall utility goes up for all algorithms, and PEMS continues to do better than all the others. This demonstrates the robustness of our algorithm.

3. Scalability: In Figure 6, we plot the running times of the algorithms with increasing number of VMs to be placed (the number of servers is chosen automatically by the algorithms to maximize utility). The experiments were run on a Pentium 4 3.4 GHz machine with 1.5 GB of RAM. Each algorithm was run 10 times and results were averaged. As shown in the graph, PEMS takes less than 4 seconds per invocation under a system environment of 1600 VMs. Also, it takes under 40 seconds for 4800 VMs. This indicates that PEMS can easily scale to large systems. Note that EMM has a greater execution time as it attempts to use many more servers than PEMS. Other algorithms run more quickly due to their first-fit nature but yield lower utility inferior solutions.

These experiments demonstrate the superior ability of PEMS to consistently deliver high quality solutions with varying

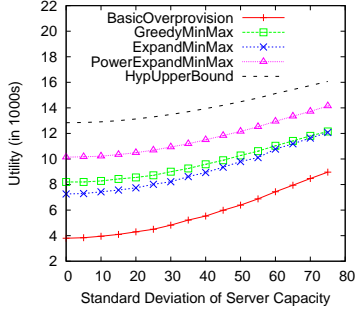


Fig. 5. Increasing server heterogeneity: P_{EMM} delivers the best valid placement decisions

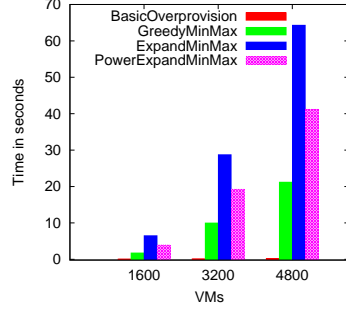


Fig. 6. Algorithm running time

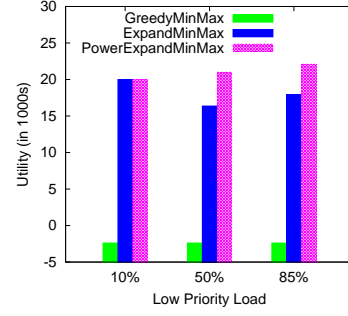


Fig. 7. Real Testbed Experiments: Varying load generated by low priority VMs. P_{EMM} outperforms other techniques.

number of available servers, data center configurations and VM size requirements.

B. Real Testbed Experiments

To further validate our findings in real installations, we conducted a set of experiments on a real data center testbed. The testbed consists of three VMware ESX 3.5 servers with 3 GHz, 4.8 (2x2.4) GHz and 6 (2x3.0) GHz total capacity. The servers are connected using an enterprise storage area network (SAN) and are configured as a cluster that allows live migration of VMs from one server to another (to test various allocation algorithms). We constructed 12 VMs running RedHat Enterprise Linux (RHEL) 4. The VMs were randomly classified into high priority and low priority(6 each). The priority was used to determine the utility generated from each VM using a simple product of amount of CPU received multiplied by a factor which was set as 4 for high priority VMs and 1 for low priority VMs. The $V_{i,m}$ for each VM was set to be 150 MHz and $V_{i,M}$ was set to be 3 GHz.

The workloads in each VM were generated using the HPC Challenge benchmark (HPCC) [17] solving a small sized linear algebra problem instance. The number of problem instances solved by the high and low priority VMs can be used as an estimate of the real-world impact of our decisions. The experiments compare GMM, EMM and P_{EMM}. The other techniques HUB and BO were not included, as HUB does not generate valid feasible placements and BO was not able to place all 12 VMs with these settings. We placed all VMs on the servers as per the allocation decisions and shares were set to provide $A(V_i).Size$ amount of resources to each VM.

1. Increasing Low-Priority VM Load: As a first experiment, we evaluated the allocation decisions of the techniques while varying the amount of load cumulatively generated by low-priority VMs. We used 75% of the total utility of BO as the power cost in this experiment. Figure 7 demonstrates that under low load of 10%, EMM and P_{EMM} perform similarly. In fact, the allocations decided by both algorithms are exactly the same, as they both use 2 servers to place all 12 VMs. Running the workload for 4 minutes solved 18 HPCC instances of high-priority VMs. GMM places all VMs in a single server allocating

only the min amount of resources resulting in poor utility generated and solving only 3 high-priority HPCC instances⁴. As the load generated by low-priority VMs increases to 50%, EMM aggressively expands VMs to 3 servers, whereas P_{EMM} is able to identify that the increased load is low-priority that does not justify using new servers. As a result, it continues to use 2 servers generating better utility while continuing to solve 18 high-priority HPCC problem instances. When the load reaches 85%, P_{EMM} continues to use 2 servers, but is able to use more capacity that results in further improvement in utility. Bigger capacity usage also helps EMM to improve utility over the previous case of 50% low-priority load.

2. Increasing Power Costs: In the second experiment, we varied the power cost for all three servers from 25% of total utility to 90% of total utility. Figure 8 shows the results. As the power cost increases, it becomes increasingly expensive to use extra servers, causing a drop in total net utility. However, P_{EMM} does the best in adjusting to these increased costs ensuring that it does not use additional servers. In contrast, EMM expands to all three servers.

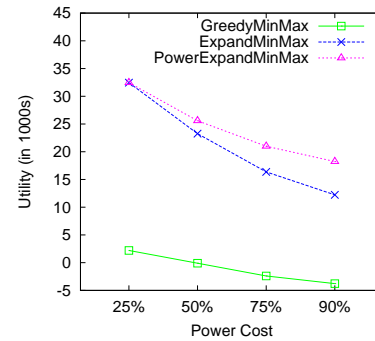


Fig. 8. Read Testbed Experiments: Varying power cost calculation. P_{EMM} improves superiority with increasing power cost.

These experiments validate our techniques on a real testbed and demonstrate that our techniques will be practically useful

⁴Since unit profitability of VMs is same at min and max in these settings, *GreedyMinMax* places VMs at their min as allocating max does not yield a feasible allocation.

in real enterprise installations.

V. RELATED WORK

What distinguishes our work from other research is our unique approach at a previously unaddressed problem of taking `min`, `max` and `shares` into account during VM allocation. Our approach allows fine grained resource allocation to heterogeneous applications in a manner that increases overall system utility and provides an intelligent power-performance tradeoff capability. In the rest of the section, we overview some related techniques, some of which are complimentary to our work.

[2], [18] describe architectures using collaborative managers and arbitrators to handle multiple objectives like power and performance during VM placement. We believe that by intelligently using `min-max` and `shares`, our work provides a key capability to perform such power-performance tradeoffs. [3], [14] describe techniques to control the rate of SLA violations in virtualized data centers. They also address mapping of SLAs to fine grained resource requirements. Our work assumes these SLA mappings already exist and uses them as inputs to our algorithms.

[19] describes techniques for developing and using server power models for consolidation and multi-objective VM placement. These server power models can be used using simple enhancements to our techniques by modifying the power-cost computation in `PEMM`. [20], [21] used utility functions for application scheduling. However, we aim to achieve best system-wide utility and exploit `min-max` and `shares` characteristics in doing so. An application placement controller supporting completion-time goals for heterogeneous workloads is presented in [22]. However, this work also does not use `min-max` and `shares` and is best suited for web server-like workloads.

There is also relevant work on VM placement that optimizes existing virtualized environments using live migrations of VMs [23], [24], [4]. Our algorithms are general enough to be extended to similar continuous optimization tasks. Our techniques will also bring important power-cost tradeoffs into the solution along with more fine-grained resource allocation.

VI. CONCLUSIONS AND FUTURE WORK

We presented a novel suite of techniques for placement and power consolidation of VMs in data centers. Unlike existing research, our techniques take advantage of the `min`, `max` and `shares` features inherent in virtualization technologies and provide a smooth mechanism for power-performance tradeoffs in modern data centers. Guided by application utilities, we are able to perform more fine-grained resource allocation that exploits application heterogeneity, ensuring that high utility applications get the most resources.

Our synthetic and real datacenter testbed experiments confirm the end-to-end validity of our approach and demonstrate that our final candidate algorithm *PowerExpandMinMax* consistently yields the best overall utility across a broad spectrum of inputs. As part of future work, we intend to expand our techniques to handle more complex resource constraints as well as using these techniques for continuous optimization of data centers using live VM migrations.

REFERENCES

- [1] CiRBA, <http://www.cirba.com>.
- [2] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *ACM 9th International Middleware Conference*, 2008.
- [3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *Proceedings of the 10th IEEE Symposium on Integrated Management (IM)*, 2007.
- [4] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Proceedings of 10th IEEE/IFIP Network Ops and Management Symp. (NOMS 2006)*, 2006.
- [5] VMware, <http://www.vmware.com/>.
- [6] P. Bohrer, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of Symposium on Operating Systems Principles*, 2003.
- [7] J. G. Koomey, "Estimating regional power consumption by servers: A technical note," *AMD Technical Study*, 2007.
- [8] IDC, 2007.
- [9] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *IEEE Computer*, vol. 37, p. 2004, 2004.
- [10] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. Mcdowell, and R. Rajamony, "The case for power management in web servers," pp. 261–289, 2002.
- [11] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proceedings of the 2005 ACM SIGMETRICS Conference*. New York, NY, USA: ACM, 2005, pp. 303–314.
- [12] D. Pisinger, "A minimal algorithm for the Multiple-choice Knapsack Problem," *European Journal of Operational Research*, vol. 83, p. 394410, 1995.
- [13] VMWare Virtual Center, <http://www.vmware.com/products/vi/vc/>.
- [14] V. Kumar, K. Schwan, S. Iyer, Y. Chen, and A. Sahai, "A State-Space Approach to SLA based Management," in *Proceedings of 11th IEEE/IFIP Network Ops and Management Symp. (NOMS 2008)*, 2008.
- [15] IBM Enterprise Workload Manager, <http://www.ibm.com/developerworks/autonomic/ewlm/>.
- [16] IBM Active Energy Manager, <http://www.ibm.com/systems/management/director/extensions/actengmgr.html>.
- [17] HPC Challenge Benchmark, <http://icl.cs.utk.edu/hpcc/>.
- [18] I. Whalley, A. N. Tantawi, M. Steinder, M. Spreitzer, G. Pacifici, R. Das, and D. M. Chess, "Experience with collaborating managers: node group manager and provisioning manager," *Cluster Computing*, vol. 9, no. 4, pp. 401–416, 2006.
- [19] A. Verma, P. Ahuja, and A. Neogi, "Power-aware Dynamic Placement of HPC Applications," in *Proceedings of the 22nd Annual International Conference on Supercomputing (ICS 2008)*, 2008.
- [20] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility-based Placement of Dynamic Web Applications with Fairness Goals," in *Proceedings of 11th IEEE/IFIP Network Ops and Management Symp. (NOMS 2008)*, 2008.
- [21] A. AuYoung, L. Rit, S. Wiener, and J. Wilkes, "Service contracts and aggregate utility functions," in *HPDC*, 2006, pp. 119–131.
- [22] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. M. Chess, "Server virtualization in autonomic management of heterogeneous workloads," in *Integrated Network Management*, 2007, pp. 139–148.
- [23] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," in *Proceedings of Symposium on Networked Systems Design and Implementation*, 2007.
- [24] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proceedings of IEEE/ACM Supercomputing*, 2008.