PACMan: Performance Aware Virtual Machine Consolidation

Alan Roytman
University of California, Los Angeles

Aman Kansal
Microsoft Research

Sriram Govindan
Microsoft Corporation

Jie Liu
Microsoft Research

Suman Nath
Microsoft Research

Abstract

Consolidation of multiple workloads, encapsulated in virtual machines (VMs), can significantly improve efficiency in cloud infrastructures. But consolidation also introduces contention in shared resources such as the memory hierarchy, leading to degraded VM performance. To avoid such degradation, the current practice is to not pack VMs tightly and leave a large fraction of server resource unused. This is wasteful. We present a system that consolidates VMs such that performance degradation is within a tunable bound while minimizing unused resources. The problem of selecting the most suitable VM combinations is **NP**-Complete and our system employs a practical method that performs provably close to the optimal. In some scenarios resource efficiency may trump performance and for this case our system implements a technique that maximizes performance while not leaving any resource unused. Experimental results show that the proposed system realizes over 30% savings in energy costs and up to 52% reduction in performance degradation compared to consolidation algorithms that do not consider degradation.

1 Introduction

Average server utilization in many data centers is low, estimated between 5% and 15% [10]. This is wasteful because an idle server often consumes more than 50% of its peak power [11], implying that servers at low utilization consume significantly more energy than fewer servers at high utilization. Additionally, low utilization implies a greater number of servers being used, resulting in wasted capital. One solution to prevent such wastage is to *consolidate* applications on fewer servers.

Consolidation inevitably introduces resource contention resulting in performance degradation. To mitigate this contention, data centers virtualize resources and split them across applications consolidated on shared hardware. However, virtualization does not prevent all forms of contention and hence does not completely eliminate performance degradation. In particular, contention in shared caches and memory bandwidth degrades performance significantly, as measured for a variety of workloads [3–5, 16, 17, 19, 21, 32, 35]. Execution times increase by several tens of percent.

To reduce degradation, prior works have measured the degradations for possible VM combinations and then co-locate those VMs that lead to the least degradation [17, 18, 29]. But this approach does not respect a target performance bound. Performance is often paramount for Internet services. Measurements on Amazon, Microsoft and Google services show that a fraction of a second increase in latency can result in revenue losses as high as 1% to 20% [13, 20, 26]. A knee-jerk reaction then is to forgo all or part of the savings from consolidation. In Google data centers for instance, consolidated workloads use only 50% of the processor cores [21]. Every other processor core is left unused simply to ensure that performance does not degrade.

We wish to preserve the performance of consolidated VMs, but *not waste excessive resources in doing so.* The challenges are to (1) determine how much each VM will degrade when placed with different sets of VMs to be consolidated, and (2) identify *which* and *how many* VMs can be placed on a server such that required performance is maintained. The problem of identifying suitable VMs turns out to be **NP**-Complete, and we design a computationally efficient algorithm that we prove performs close to the theoretical optimal. As a result, the excess resources left unused in our approach are significantly lower than in current practice.

An additional mechanism to preserve performance after consolidation is to improve the isolation of resources in hardware [3, 5, 16, 28, 35], or software [1, 4, 6, 27, 32]. Further, excess resources may be allocated at run time [23] to overcome degradation. These approaches are complementary because they do not determine the

best VMs to be placed together in the first place. Our method can make that determination, and then these techniques can be applied with a lower overhead.

Specifically, we make the following contributions:

First, we present a *p*erformance *a*ware *c*onsolidation *man*ager, PACMan, that minimizes resource cost, such as energy consumption or number of servers used. PACMan consolidates VMs such that performance degradation stays within a specified bound. Since this problem is **NP**-complete, PACMan uses an approximate but computationally efficient algorithm that we prove performs logarithmically close to the optimal.

Second, while customer-facing applications prioritize performance, batch processes, such as Map-Reduce [8], may prioritize resource efficiency. For such situations PACMan provides an "Eco" mode, that fills up all server cores, and minimizes worst case degradation. We specifically consider worst case, as opposed to average considered in [17], since in Map-Reduce, reduce cannot start until all map tasks have completed and hence, only the degradation of the worst hit map task matters. We show that it is difficult to design provably near-optimal methods for this scenario and present a suitable heuristic.

Finally, we evaluate PACMan using degradations measured on SPEC CPU 2006 applications. For minimizing wasted resource while preserving performance, PACMan operates within about 10% of the optimal, saves over 30% energy compared to consolidation schemes that do not account for interference, and improves total cost of operations by 22% compared to current practice. For the Eco mode, PACMan yields up to 52% reduction in degradation compared to naïve methods.

2 PACMan Design

This section describes the performance repercussion of consolidation and how our design addresses it.

2.1 Problem Description

Consolidation typically relies on virtual machines (VMs) for resource and fault isolation. Each VM is allocated a fixed share of the server's resources, such as a certain number of cores on a multi-core server, a certain fraction of the available memory, storage space, and so on. In theory, each VM should behave as if it is a separate server: software crashes or resource bottlenecks in one VM should not affect other VMs on the same server. In practice however, VM resource isolation is not perfect. Indeed, CPU cores or time slices, memory space, and disk space can be isolated well using existing virtualization products, and methods have emerged for other resources such as network and storage bandwidth [22, 34]. However, there remain resources, such as *shared caches and memory bandwidth*, that are hard to isolate. Hence, consolidated applications, even when encapsulated in VMs, may suffer resource contention or *interference*, and this leads to performance degradation.

Example: Consider a toy data center with 4 VMs, A, B, C, and D (Figure 1). On the left, the 4 VMs are placed on a single server each. Suppose the task inside each VM takes 1 hour to finish. The shaded portion of the vertical bars represents the energy used over an hour; the darker rectangle represents the energy used due to the server being powered on (idle power consumption) and the rectangles labeled with the VM name represent the additional energy consumed in VM execution (increase in server energy due to processor resource use). On the right, these VMs are consolidated on two servers (the other two are in sleep mode).

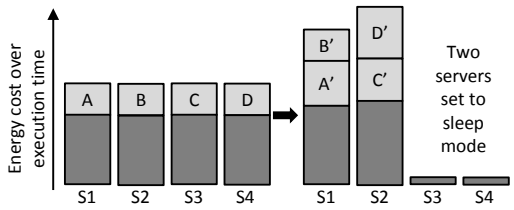

Figure 1: Energy cost change due to consolidation.

The setup on the right is more efficient. However, due to resource contention, the execution time goes up for most of the jobs. Both the server idle energy and the additional energy used by each job increase due to the longer run time. The increase in energy consumption due to contention may wipe out some or all of the energy savings obtained by turning off two servers. Also, longer running time may violate quality of service (QoS) requirements.

One may minimize performance degradation by placing each VM in a separate server, but that obviously reduces efficiency. On the other hand, one may maximize efficiency by packing the VMs into the minimum number of servers required to satisfy the number of processor cores, memory and disk space requirements of each VM, but such packing hurts performance.

2.2 System Overview

Our goal is to select the right set of VMs to co-locate on each server such that performance constraints are satisfied and wasted resource is minimized.

2.2.1 Assumptions

We make three assumptions about the system:

Degradation: We assume that the performance degradation suffered by each VM, when consolidated with

any set of other VMs, is known from existing methods [12, 19, 21]. These methods do not require explicit performance measurement for each possible set of VMs. Rather, a VM is profiled individually to generate an *interference profile*. Profiling takes a few hundred milliseconds depending on the cache architecture. These profiles can be used to compute the expected degradation for any set of VMs placed together. Small errors in prediction can be addressed by including an error margin in the performance bound, and consolidating to within that conservative bound. Explicit measurement may also be used for a small number of VMs, as in [17]. We focus on the consolidation method given the degradations. Since our algorithms work given any interference data, the techniques we use can be applied to cross-socket interference or any other type of interference as well, so long as it can be quantified and measured.

Temporal Demand Variations. We assume that as demand for an application varies, the number of VM instances hosting that app are increased or decreased to match the demand. Overloading a small number of VMs would degrade performance while leaving VMs underutilized would incur excess cost to host them. Hence, commercial tools are available to dynamically change the number of VMs [24, 33]. This implies that the degradation data or interference profile needs to be collected only for the desired demand level, rather than at multiple demand levels that a VM may serve. If demand is lower than that served by a single VM instance for the application, we conservatively use the profile at its optimal demand level.

VM to Processor Core Mapping: We assume that each VM is assigned one core, following the model in [3, 12, 16, 17, 19, 32]. Considering VMs that span multiple cores does not change the problem fundamentally. However, if multiple VMs share a single core, the nature of resource contention may change, and existing degradation estimation methods [12, 19, 21] will not suffice. If alternate degradation modeling methods are available or explicit measurements of degradations are provided, our consolidation algorithm would extend to that case.

2.2.2 Architecture

The PACMan system architecture is shown in Figure 2. The system consists of the following three components:

Conservatively Packed Servers: Customers submit VMs through appropriate cloud APIs. Ideally, a VM placement solution should determine the optimal placement for each VM as soon as it arrives, such that the entire set of VMs currently running in the cloud is optimally placed. However, since such an optimal online solution is not available, we focus on a *batched* operating scenario. The cloud initially hosts the incoming VMs on conservatively packed servers, for a batching period (say 30 to 60 minutes). These servers may comprise a small fraction (say 1%-5%) of the data center. Conservative placement implies that a significant amount of resources are left unused to avoid interference, such as by leaving alternate processor cores empty [21]. Since the VM is active, it does not matter to the customer that it is placed on a conservatively packed server.

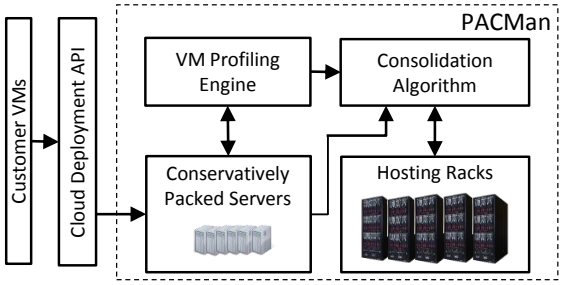

Figure 2: PACMan block diagram.

VM Profiling Engine: While a VM is running on the conservatively packed servers, profiling methods from [12, 21] are applied to the VMs[1]. These methods characterize a VM while it is running normally, and generate a set of parameters that allow estimating the performance degradation that will be *suffered* and *caused* by the VM when consolidated with other VMs. Their prediction accuracy is high (5-10% of actual performance), as measured on real data center and benchmark applications. Given n VMs and k core servers, only $O(n)$ measurements are needed, even though the number of possible consolidated sets is $O(n^k)$.

Consolidation Algorithm: At the end of each batching period, PACMan uses the VM consolidation algorithm proposed in this paper to place the VMs on *hosting racks* that comprise the bulk of the cloud's infrastructure. Most of the data center thus operates efficiently using the near-optimal placement. The inputs to the algorithm are the VM interference characteristics obtained by the profiling engine. The output is a placement of VMs that respects performance constraints and minimizes unused resources. Typically, other algorithms (including bin packing methods such as best fit or first fit) do not take interference into account, and hence cannot consolidate VMs efficiently. The design of PACMan algorithms is presented in the next two sections.

[1] We use [12] in our prototype. In this method, each VM is mapped to a *clone* application, which closely mimics the application's interference signature. A discrete set of clones covers the entire spectrum of memory-subsystem interference behaviors. Thus, a potentially unbounded number of applications are mapped to a finite number of clones. A one-time profiling step maps a new VM to a known clone. The clones are then used as a proxy for predicting performance for different consolidation sets.

3 Performance Mode

The first mode of PACMan operation, denoted the performance mode (P-mode), determines the best sets and their sizes such that performance constraints are not violated. It may leave some processor cores unused, unlike prior methods that use up every core [17, 18] but may violate performance constraints.

Servers and VMs: Suppose m chip-multiprocessors (CMPs) are available, each with k cores. We are primarily interested in the inter-core interference within a CMP. The VMs placed on the same CMP suffer from this degradation. If a server happens to have multiple processor sockets, we assume there is no interference among those. As a result, multiple CMPs within a server may be treated independently of each other. We loosely refer to each CMP as a separate server as shown in Figure 3. We are given n VMs to be placed on the above servers, such that each VM is assigned one core.

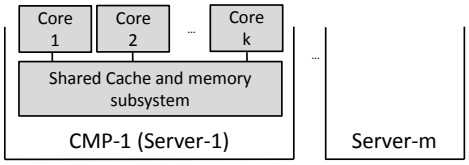

Figure 3: CMPs (referred to as servers) with k cores. Contention in the shared cache and memory hierarchy degrades the performance of VMs in the same server.

Degradation: Suppose that the set of VMs placed together on a server are denoted by S. For singleton sets, i.e., a VM j running alone, there is no degradation and we denote this using a degradation $d_j = 1$. For larger sets, the degradation for VM $j \in S$ is denoted by $d_j^S \geq 1$. For example, for two co-located VMs, $S = \{A, B\}$, suppose A's running time increases by 50% when it runs with B, relative to when it runs alone, while B is unaffected by A. Then, $d_A^S = 1.5$ and $d_B^S = 1$.

We assume that adding more VMs to a set may only increase (or leave unchanged) the degradation of previously added VMs.

3.1 Consolidation Goal

The consolidation objective may be stated as follows.
P-Mode: (Minimize resource cost subject to a performance constraint)
Given
n VMs,
Servers with k cores,
Degradations for all sets of VMs up to size k,
Cost $w(S)$ for a set of VMs S placed on a server, and
Maximum tolerable degradation $D \geq 1$ for any VM[2].
Find a placement of the n VMs using some number, b, of servers, to minimize

$$\sum_{i=1}^{b} w(S_i)$$

where S_i represents the set of VMs placed on the i^{th} server.

Cost Metric: The resource cost, $w(S)$, to be minimized may represent the most relevant cost to the system. For instance, if we wish to minimize the number of servers used, then we could use $w(S) = 1$ for any set S regardless of how many VMs S contains. To minimize energy, $w(S)$ could be defined as the sum of a fixed cost c_f and a dynamic cost c_d. The fixed cost c_f models the idle energy used for keeping a server active, and may also include capital expense. The dynamic cost, c_d, models the increase in energy due to VMs assigned to the server. For concreteness, we consider the cost function $w(S)$ to be the energy cost. The specific values used for c_f and c_d are described in Section 5 along with the evaluations. Our solution is applicable to any cost function that monotonically increases with the number of VMs.

Batched Operation: The problem above assumed all VMs are given upfront. In practice, following the setup from Figure 2, only the VMs that arrived in the most recent batching period will be consolidated. Each batch will hence be placed optimally using P-mode consolidation, but the overall placement across multiple batches may be sub-optimal. Hence, once a day, such as during times of low demand, the placement solution can be jointly applied to all previously placed VMs, and the placement migrated to the jointly optimal placement. The joint placement satisfies the same performance constraints but may reduce resource cost even further.

3.1.1 Problem Complexity

The complexity is different depending on whether the servers have only $k = 2$ cores or more than 2 cores.

Dual-Core servers: For $k = 2$ cores, there is a polynomial time algorithm that can compute the optimal solution. The main idea is to construct a weighted, undirected graph on $2n$ nodes. The first n nodes represent the VMs, and the others are "dummy" nodes (one for each VM). For VM pairs whose degradation is below the bound D, we place an edge connecting them and assign an edge weight equal to the cost of placing those two VMs together. We place an edge between each VM node and its dummy node with a weight that corresponds to the cost

[2]We assume that the performance constraint is the same for all VMs though multiple quality of service classes, each with their own degradation limit, could be considered as well and do not fundamentally change the problem.

of running that VM alone. Finally, we place edges of weight 0 between all pairs of dummy nodes. Finding the best pairs of VMs for a consolidated placement is equivalent to computing a minimum cost perfect matching on this graph. Graph algorithms are available to compute a minimum cost perfect matching in polynomial time. We omit details for this case since most data center servers have more than 2 cores.

NP-Completeness: For servers with more than two cores ($k \geq 3$), the problem is **NP**-Complete. This is because it can be thought of as a variant of the k-Set Cover problem. In the k-Set Cover problem, we have a universe U of elements to cover (each element could represent a VM), along with a collection C of subsets each of size at most k (the subsets could represent sets of VMs with degradation below D). Placing VMs on servers corresponds to finding the minimum number of disjoint VM subsets that cover all VMs. Assuming $w(S) = 1$ for all sets S, the k-Set Cover problem becomes a special case of the P-mode problem, i.e., solving the P-mode problem enables solving the k-Set Cover problem. The k-Set Cover problem is **NP**-Complete [9]. Hence, the P-mode problem is **NP**-Complete.

3.2 Consolidation Algorithm

Since the problem is **NP**-Complete for $k \geq 3$ cores, we propose a computationally efficient algorithm that finds a near-optimal placement.

Using the profiling method described in Section 2.2, it is easy to filter out VM sets that violate the degradation constraint. Suppose the collection of remaining sets (VM combinations that can be used) is denoted by $\mathscr{F}$.

First, for each set $S \in \mathscr{F}$, the algorithm assigns a value $V(S) = w(S)/|S|$. Intuitively, this metric characterizes the cost of a set S of VMs. Sets with more VMs (larger set size, $|S|$) and low resource use ($w(S)$) yield low $V(S)$.

Second, the algorithm sorts these sets in ascending order by $V(S)$. Sets that appear earlier in the ascending order have lower cost.

The final step is to make a single pass through this sorted list, and include a set S as a placement in the consolidation output if and only if it is disjoint from all sets that have been chosen earlier. The algorithm stops after it has made a single pass through the list. The algorithm can stop earlier if all the VMs are included in the chosen sets. The first set in the sorted list will always be taken to be in the solution since nothing has been chosen before it and it is hence disjoint. If the second set is disjoint from the first set, then the algorithm takes it in the solution. If the second set has at least one VM in common with the first, the algorithm moves onto the third set, and so on. The precise specification is given in Algorithm 1.

Example: Consider a toy example with three VMs, A, B, and C and $k = 2$ cores. Suppose the characterization from the VM profiling engine results in the degradation numbers shown in Table 1. Suppose the performance constraint given is that no VM should degrade more than 10% ($D = 1.1$) and the cost metric $w(S)$ is just the number of servers for simplicity ($w(S) = 1$ for any set). A set with two VMs ($|S| = 2$) will have $V(S) = 1/2$ while a set with one VM will have $V(S) = 1$. Then filtering out the sets that cause any of the VMs to have a degradation greater than D, and computing the $V(S)$ metric for each set, we get the sorted list as: BC, AB, A, B, C. The algorithm first selects set BC and allocates it to a server (VMs B and C thus share a single server). The next set AB is not disjoint from BC and the algorithm moves to the subsequent set A. This is disjoint and is allocated to another server. All VMs are now allocated and the algorithm stops.

Algorithm 1 CONSOLIDATE($\mathscr{F}$, n, k, D)

```
Compute V(S) ← w(S)/|S|, for all S ∈ F
L ← Sorted sets in F such that V(S_i) ≤ V(S_j) if i ≤ j
𝕃 ← φ
for i = 1 to |L| do
    if S_i is disjoint from every set in 𝕃 then
        𝕃 ← 𝕃 ∪ {S}
Return 𝕃
```

VM Set	AB	AC	BC	A	B	C
d_{VM}^{Set}	$d_A = 1.1$ $d_B = 1.1$	$d_A = 1.0$ $d_C = 1.5$	$d_B = 1.0$ $d_C = 1.1$	1	1	1

Table 1: Degradations for VMs in the example.

Complexity: The algorithm operates in polynomial time since sorting is a polynomial time operation, $O(|\mathscr{F}| \cdot log(|\mathscr{F}|))$. The subsequent step requiring a single pass through the list has linear time complexity. At every step in the linear pass the algorithm needs to check if each VM in the set being selected has been assigned already and this can be achieved in constant time as follows. Maintain a boolean bit-vector for every VM indicating if it has been assigned yet. For the set being checked, just look up this array, which takes at most $O(k)$ time per set since the set cannot have more than k VMs. Also, after selecting a set we update the boolean array, which again takes constant time.

While the computation time is polynomial in the size of the input, the size of the input can be large. The list of degradation values for all possible VM sets has size $|\mathscr{F}| = O(n^k)$ elements, which can be large for a cloud infrastructure hosting thousands of VMs. However, when the degradation estimation technique from [12] is used,

all VMs are mapped to a finite set of clones and the number of clones does not grow with the number of VMs. We can treat all VMs that map to a common clone as one type of VM. The number of clones used to map all VMs then represents the distinct types of VMs in the input. For instance, for the characterization technique in [12], for quad-core servers, at most 128 types of clones are required, and not all of them may be used for a particular set of input VMs.

Suppose the n VMs can be classified into $\tau \leq 128$ types. Then, the algorithm only needs to consider all sets S from τ VM types with possibly repeated set elements. The number of these sets is $O(\tau^k)$, which is manageable in practice since τ does not grow very large, even when n is large.

The algorithm changes slightly to accommodate multiple VMs of each type. The assignment of value $V(S)$ and the sorting step proceed as before. However, when doing the single pass over the sorted list, when a disjoint set S is chosen, it is repeatedly allocated to servers as long as there is at least one unallocated instance of each VM type required for S. The resultant modification to Algorithm 1 is that $\mathscr{F}_\tau$ is provided as input instead of $\mathscr{F}$ where $\mathscr{F}_\tau$ denotes the collection of all feasible sets of VM types with repeated elements, and at step 5, instead of checking if the VMs are not previously allocated, one repeats this step while additional unallocated VMs of each type in the set remain.

Correctness: The algorithm always assigns every VM to a server since all singleton sets are allowed and do appear in the sorted list (typically after the sets with large cardinality). Also, it never assigns a VM to more than one server since it only picks disjoint sets, or sets with unallocated VM instances when VM-types are used, while making the pass through the sorted list. Hence, the algorithm always obtains a correct solution.

3.3 Solution Optimality

A salient feature of this algorithm is that the consolidation solution it generates is guaranteed to be near-optimal, in terms of the resources used.

Let ALG denote the allocated sets output by the proposed algorithm, and let OPT be the sets output by the optimal solution. Define the resource cost of the proposed algorithm's solution to be $E(ALG)$, and that of the optimal algorithm as $E(OPT)$. We will show that for *every* possible collection of VMs to be consolidated,

$$E(ALG) \leq H_k \cdot E(OPT)$$

where H_k is the k^{th}-Harmonic number. $H_k = \sum_{i=1}^{k} \frac{1}{i} \approx \ln(k)$.

In other words, the resource cost of the solution generated by the proposed algorithm is within $\ln(k)$ of the resource cost of the optimal solution. Given that k is constant for a data center and does not increase with the number of VMs, this is a very desirable accuracy guarantee. The proof is inspired by the approximation quality proof for the weighted k-Set Cover problem [7, 15]. However, we cannot pick overlapping sets (since choosing sets in our setting corresponds to choosing a placement of VMs onto servers), and the input sets are closed under subsets.

Theorem 1. *For all inputs, the proposed algorithm outputs a solution that is within a factor $H_k \approx \ln(k)$ of the optimal solution.*

Proof. By definition, we have

$$E(ALG) = \sum_{S \in ALG} w(S).$$

Assign a cost to each VM $c(j)$ as follows: whenever the proposed algorithm chooses a set S to be part of its solution, set the cost of each VM $j \in S$ to be $c(j) = w(S)/|S|$ (these costs are only for analysis purposes, the actual algorithm never uses $c(j)$). Hence,

$$E(ALG) = \sum_{S \in ALG} |S| \frac{w(S)}{|S|} = \sum_{S \in ALG} \sum_{j \in S} c(j) = \sum_{j=1}^{n} c(j),$$

where the last equality holds because the set of VMs in the solution is the same as all VMs given in the input. Then, since the optimal solution also assigns all VMs to servers:

$$E(ALG) = \sum_{j=1}^{n} c(j) = \sum_{S^* \in OPT} \sum_{j \in S^*} c(j),$$

where $S^* \in OPT$ is a set chosen by the optimal solution. Suppose, for the moment, we could prove that the last summation term above satisfies $\sum_{j \in S^*} c(j) \leq H_k w(S^*)$. Then we would have

$$E(ALG) \leq \sum_{S^* \in OPT} H_k w(S^*) = H_k \cdot E(OPT).$$

All we have left to prove is that, for any $S^* \in OPT$, we indeed have $\sum_{j \in S^*} c(j) \leq H_k w(S^*)$. Consider any set S^* in the optimal solution and order the VMs in the set according to the order in which the *proposed* algorithm covers the VMs, so that $S^* = \{j_s, j_{s-1}, \ldots, j_1\}$. Here, j_s is the first VM from S^* to be covered by the proposed algorithm, while j_1 is the last VM to be covered by the proposed algorithm in potentially different sets. In case the proposed algorithm chooses a set which covers several VMs from S^*, we just order these VMs arbitrarily.

Now, consider VM $j_i \in S^*$ immediately before the proposed algorithm covers it with a set T. At this time, there are at least i VMs which are not covered, namely

$j_i, j_{i-1}, \ldots, j_1$. There could be more uncovered VMs in S^*, for instance, if the proposed algorithm chose set T such that T covers VMs $j_s, \ldots, j_i$, then all VMs in S^* would be considered uncovered immediately before set T is chosen. Moreover, since the optimal solution chose S^*, and since sets are closed under subsets, it must be the case that the proposed algorithm could have chosen the set $S = \{j_i, \ldots, j_1\}$ (since it is a feasible set and it is disjoint). At each step, since the proposed algorithm chooses the disjoint set T that minimizes $w(T)/|T|$, it must be the case that $w(T)/|T| \leq w(S)/|S|$. By our assumption that energy costs can only increase if VMs are added, we have $w(S) \leq w(S^*)$, and hence VM j_i is assigned a cost of $w(T)/|T| \leq w(S)/|S| \leq w(S^*)/|S| = w(S^*)/i$. Summing over all costs of VMs in S^*, we have

$$\sum_{j \in S^*} c(j) \leq \sum_{j_i \in S^*} w(S^*)/i = H_s \cdot w(S^*) \leq H_k \cdot w(S^*)$$

(since $|S^*| = s \leq k$). Hence, $\sum_{j \in S^*} c(j) \leq H_k \cdot w(S^*)$ indeed holds and this completes the proof. □

To summarize, we provide a polynomial time algorithm that is guaranteed to provide a solution within a logarithmic factor of the optimal. Note that this is a worst-case guarantee, and in practice we can expect the solution quality to be better (e.g., our experimental results in Section 5). In fact, our approximation guarantee is asymptotically the best approximation factor one could hope for, due to the hardness of approximation lower bound known for the k-Set Cover problem [30] (hence, there are worst-case instances in which any algorithm must perform poorly, but these instances typically do not occur in practice and the algorithms perform much better).

4 Eco-Mode

In some cases, such as batch based data processing, resource efficiency may take precedence over performance. For such scenarios, PACMan provides a resource efficient mode of operation, referred to as the Eco-mode. Here, the number of servers used is fixed and the VMs are tightly packed. The goal is to minimize the degradation. Prior works have minimized average degradation [17, 18, 29] and their heuristics can be used in Eco-mode. We additionally consider worst case degradation. The worst case degradation is especially important for parallel computing scenarios where the end result is obtained only after all parallelized tasks complete and hence performance is bottle-necked by the worst hit VM.

Eco-Mode: (Minimize maximum degradation)
Given
n VMs,
m servers with k cores ($n \leq mk$), and
Degradations for all sets of VMs up to size k,
Find an allocation of the n VMs to m servers which minimizes the objective

$$\max_{1 \leq i \leq m} \max_{j \in S_i} d_j^{S_i}$$

where S_i represents the set of VMs placed on the i^{th} server ($|S_i| \leq k$ for each i).

As in the previous case, while the 2-core case can be solved in polynomial time, the Eco-mode problem becomes **NP**-Complete for $k \geq 3$.

Efficient Near-Optimal Algorithms: Given that the problem is **NP**-Complete, a polynomial time algorithm to compute the optimal solution is unlikely to be found, unless $\mathbf{P} = \mathbf{NP}$. The next best thing would be an efficient algorithm that computes a provably near-optimal solution.

Surprisingly, for $k \geq 3$ a computationally efficient algorithm that guarantees the solution to be within any polynomial factor of the optimal cannot be found. For instance, a computationally efficient algorithm that can guarantee its solution to be within a factor n^{100} of the optimal cannot be found.

Theorem 2. *For the Eco-mode consolidation problem, it is* **NP***-Hard to approximate the optimal solution to within any factor that is polynomial in the number of VMs and servers.*

The proof is relegated to a tech report [25] for brevity.

Algorithm: The implication of the above theorem is that any computationally efficient Eco-mode algorithm will have to rely on heuristics.

The heuristic we propose greedily improves a given placement using VM swaps. A swap refers to exchanging the placement of one VM with another. Start out with any initial placement of VMs. Consider all possible placements that are reachable from the existing placement in a single swap. In each such placement, for each server, compute the degradation of the worst hit VM on that server, using the degradation characterization from the VM profiling engine. Take the sum of these worst case degradations on all servers as the cost of that VM placement.

Among all possible placements reachable within a swap, greedily select the one with the lowest cost and actually perform the swap required to reach that placement. Repeat the above process as long as additional swaps are allowed or until a swap no longer yields an improvement.

The work of [31] studies the cost of swapping, giving insight into the trade-off between improving resource efficiency and swapping VMs. To this end, we limit the number of swaps allowed to terminate the search at $G = (k-1)(m-1)$. Starting with an arbitrary placement, it is possible to reach *any* other placement (e.g., the optimal

placement) by performing at most $G = (k-1)(m-1)$ swaps. This holds because for each server, we can imagine one of the VMs to be in the correct position on that server, and hence there can be at most $k-1$ VMs on that server that are out of place. By swapping two VMs, we can assign each VM which is on the wrong server to the right server. Hence, each server can be fixed in at most $k-1$ swaps. Once $m-1$ servers have been fixed, the last server must already be correct. However, determining the appropriate number of swaps is not easy and our heuristic is not guaranteed to find the optimal placement in $G = (k-1)(m-1)$ swaps, or any number of swaps. Hence, the number of allowed swaps may be set based on other constraints such as limits on tolerable swap overheads or a threshold on minimum improvement expected from a swap.

While not provably near-optimal, our heuristic is beneficial to the extent that it improves performance compared to naïve methods.

5 Experimental Results

In this section, we quantify the resource savings and performance advantages of using PACMan consolidation for realistic scenarios. Ideally, we wish to compare the practical algorithm used in PACMan with the theoretical optimal, but the optimal is not feasible to compute (these problems are **NP**-Complete) except for very small input sizes. Hence, we illustrate the performance of the proposed methods with respect to the optimal for a few small input instances ($n = 16$ VMs, $m \geq \lceil n/k \rceil$). For more realistic inputs, relevant to real data centers (10^3 VMs), we compare the performance to naïve methods that are unaware of the performance degradation and with one current practice that leaves alternate processor cores unused [21]. For these cases, we also compute the degradation overhead compared to a hypothetical case where resource contention does not cause any degradation. This comparison shows an upper bound on how much further improvement one could hope to make over the PACMan methods.

5.1 Experimental Setup

Degradation Data: We use measured degradation data for SPEC CPU 2006 benchmark applications. These degradations are in the same range as measured for Google's data center workloads in [21], which includes batch and interactive workloads. Since the degradation data is representative of both interactive workloads and batch workloads, it is relevant for both P-mode and Eco-mode.

In particular, we select 4 of the SPEC CPU benchmark applications for which we have detailed interference data for all possible combinations, namely: `lbm`, `soplex`, `povray`, and `sjeng` (some combinations shown in Table 2). These span a range of interference values from low to high. When experimenting with n VMs, we generate an equal number, $n/4$, of each. We do not vary VM degradations over time during VM execution.

Application VMs (S_i)	Degradations (%)
`lbm, soplex`	2, 19.7
`soplex, soplex`	10, 10
`lbm, soplex, sjeng`	2, 10, 4.1
`lbm, povray, lbm`	19.6, 5.32, 19.6
`lbm, soplex` `soplex, sjeng`	14.56, 36.9, 36.9, 5.83
`lbm, lbm, lbm, lbm`	104.6 (each)

Table 2: Sample degradation data for the application VMs used in experiments. Degradations are measured on a quad-core processor. For combinations with only 2 or 3 VMs, the remaining cores are unused. Degradations over 100% imply that the execution time of the workload increases by more than twice.

Cloud Configuration: We assume that each server has $k = 4$ cores since quad-core servers are commonly in use. While a server may have many cores across multiple processor sockets, the relevant value of k is the number of cores sharing the same cache hierarchy, since that is where most of the interference occurs. Using real world degradation data, we simulate our proposed algorithm for the cloud configuration described above.

5.2 Performance Mode

The P-mode problem optimizes resource cost given a degradation constraint. The evaluation metric of interest is thus the resource cost. We choose energy as our resource metric. Each server has a fixed and dynamic energy component (Section 3), resulting in an energy cost $w(S) = c_f + \sum_{j \in S} d_j^S$. Here, the additional cost of each VM is being modeled as d_j^S. Considering that running 4 VMs each with an incremental cost of 1 or more would add an additional 4 units of dynamic resource cost, we set the fixed cost $c_f = 4$ to reflect about 50% of the total server energy as the fixed idle cost, which is representative of current server technology. Newer generation servers are trending towards better energy proportionality and idle power costs as low as 30% are expected in the near future. A lower idle power cost will only exaggerate the fraction of overhead due to interference and lead to even greater savings in PACMan.

Comparison with Optimal: To facilitate computation of the optimal, we use a small number, 16, of VMs, with equal proportion of VMs from each of the four

benchmarks. We vary the degradation constraint from 10% ($D = 1.1$), to as high as 50%[3]. Aside from the optimal, we also compare against a naïve method that does not quantitatively manage degradation but conservatively leaves every other core unused [21].

Figure 4 shows the energy overhead of the consolidation determined by PACMan, and by the naïve method, over and above the energy used by the optimal method. The proposed approach is within 10% of the optimal, and is significantly better than the naïve approach currently in use.

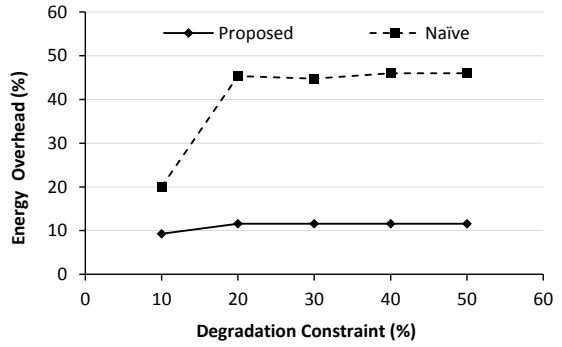

Figure 4: (P-Mode) Excess energy used compared to that used by the optimal solution (computable for a small number of VMs).

Figure 5 shows the server utilizations achieved by the three methods at equivalent performance. The proposed method achieves over 80% utilization in most cases yielding good resource use. Of course, when the degradation allowed is small, servers must be left under-utilized to avoid interference, and even the optimal method cannot use all cores.

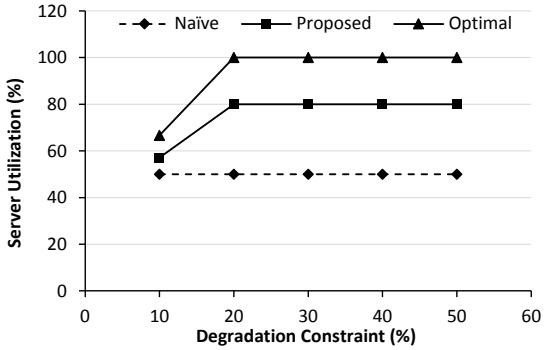

Figure 5: (P-Mode) Server utilizations achieved by the theoretical optimal, proposed, and naïve algorithms.

[3]We start at 10% instead of 0%, since if no degradation is allowed, most VMs would require a dedicated machine, with no benefits from consolidation.

Large number of VMs: The second set of experiments uses more realistic input sizes, up to $n = 1000$ VMs, again taking an equal proportion of VMs from each of the four SPEC CPU applications listed in Section 5.1. Since it is not feasible to compute the optimal solution for a large number of VMs, we compare against a lower bound: resources used when interference has no effect. In reality, interference will lead to a non-zero overhead and the optimal should be expected to be somewhere between 0% and the overhead seen for the proposed method. Figure 6 shows the results, with a performance constraint of 50% ($D = 1.5$), for varying n. We see that the proposed method performs significantly better than the naïve one.

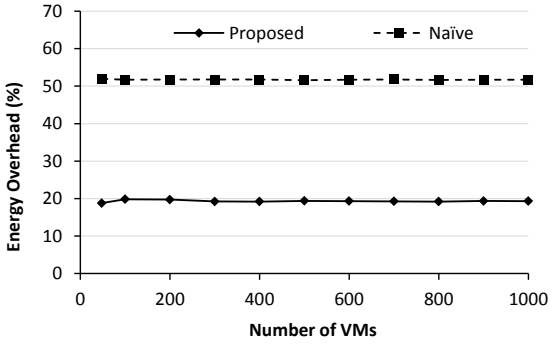

Figure 6: (P-Mode, Large number of VMs) Resource overhead comparison, normalized with respect to hypothetical resource use when there is no interference, which is a lower bound on the optimal.

5.3 Eco-Mode

For the Eco-mode problem, we again compute the optimal solution for a small set $n = 16$ VMs with $m = 4$ servers, with the VMs taken from the SPEC CPU benchmarks. The initial allocation of VMs to servers is arbitrary and we repeat the experiment 10 times, starting with a random initial allocation each time. Since *any* allocation can be reached in at most $(k-1)(m-1) = 9$ swaps, we vary the number of allowed swaps G from 2 to 9. As an additional point of comparison we use a naïve approach that does not consider interference and places the VMs randomly. The performance of the randomized approach is averaged across 10 trials.

Figure 7 shows the excess degradation suffered by the VMs compared to that in the optimal allocation. The practical heuristic used in PACMan performs very closely to the optimal and has up to 30% lower degradation than the naïve method.

Next we vary the number of VMs up to $n = 1000$, packed tightly on $m = n/4$ quad core servers. The ap-

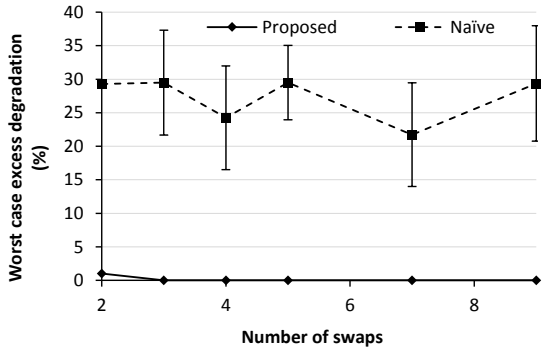

Figure 7: (Eco-mode) Comparison of proposed heuristic and a naïve random algorithm with the theoretical optimal (computable for small input instances). Excess worst case degradation compared to that in the optimal solution is shown. The error bars show the standard deviation across 10 random runs for the naïve approach.

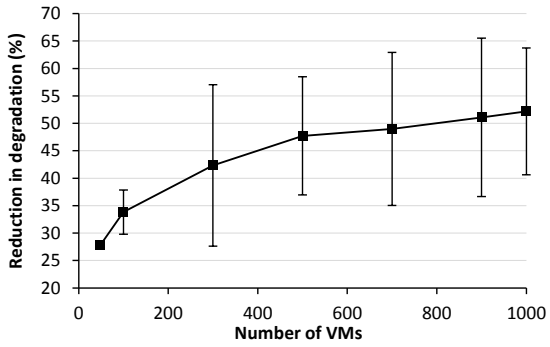

Figure 8: (Eco-mode, Large number of VMs) Reduction in degradation compared to a naïve approach. The error bars show the standard deviation across 10 random placements for the naïve approach.

plications are taken from the SPEC CPU benchmarks as before, in equal proportion. The naïve approach used for comparison is a random placement that does not account for interference (10 random trials are performed for each point).

Since it is not feasible to compute the optimal solution, we use the naïve approach as the base case and show the reduction in degradation achieved by PACMan (Figure 8). The worst case degradation is reduced by 27% to 52% over the range of the number of VMs. While the number of servers is a fixed constraint, reduction in performance degradation results in a corresponding increase in throughput or reduction in runtime, yielding a proportional saving in energy per unit work performed.

In summary, we see that PACMan performs well on realistic degradation data.

5.4 TCO Analysis

The total cost of ownership (TCO) of a data center includes both the operating expenses such as energy bills paid based on usage, and capital expenses, paid upfront. Consolidation affects multiple components of TCO. The resultant savings in TCO are described below.

To compare capital costs and operating expenses using a common metric, James Hamilton provided an amortized cost calculation of an entire data center on a monthly basis [14]. In this calculation, the fixed costs are amortized over the life of the component purchased. For instance, building costs are amortized over fifteen years while server costs are amortized over three years. This converts the capital costs into a monthly expense, similar to the operating expense.

Figure 9 shows the savings resulting in various data center cost components. The baseline we use is the current practice of leaving alternate cores unused [21], and we compare this with our proposed performance preserving consolidation method. In all, a 22% reduction in TCO is achieved, which for a 10MW data center implies that the monthly operating expense is reduced from USD 2.8 million to USD 2.2 million.

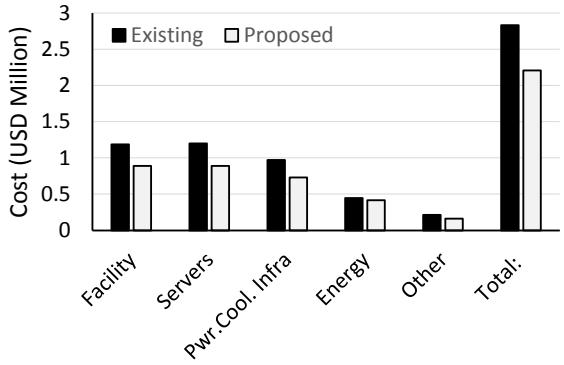

Figure 9: (P-Mode) TCO reduction using the proposed performance preserving consolidation method. Pwr. Cool. Infra. refers to the power and cooling infrastructure cost, as defined in [14].

6 Related Work

Performance isolation from memory subsystem interference has been studied at different levels of the system stack: the hardware level [3, 5, 16, 28, 35], the OS/software level [1, 6, 23, 27], and the VM scheduler level [2, 4, 32]. Our method is complementary in that we facilitate determining the placements with lower interference to which above isolation techniques can then be applied, and are likely to be more effective.

Performance estimates due to interference [12, 19, 21] have also been developed to aid VM placement. We build upon the above works and use the interference characterization provided by them to determine the placements with lower interference.

Consolidation methods taking interference into account have been studied in [17], along with variants for unequal job lengths [29]. Another method to model VM interference through cache co-locality, and a heuristic for run-time scheduling to minimize degradation, was presented in [18]. We allow optimizing for a different objective. While heuristics were proposed in prior works, we provide an algorithm with provable guarantees on the solution quality. We also provide a new inapproximability result.

7 Conclusions

VM consolidation is one of the key mechanisms to improve data center efficiency, but can lead to performance degradation. We presented consolidation mechanisms that can preserve performance.

The extent of performance degradation depends on both how many VMs are consolidated together on a server and which VMs are placed together. Hence, it is important to intelligently choose the best combinations. For many cases, performance is paramount and consolidation will be performed only to the extent that it does not degrade performance beyond the QoS guarantees required for the hosted applications. We presented a system that consolidated VMs within performance constraints. While the problem of determining the best suited VM combinations is **NP**-Complete, we proposed a polynomial time algorithm which yields a solution provably close to the optimal. In fact, the solution was shown to be within $\ln(k)$ of the optimal where k is the number of cores in each server, and is independent of the number of VMs, n. This is a very tight bound for practical purposes. We also considered the dual scenario where resource efficiency is prioritized over performance. For this case, we showed that even near-optimal algorithms with polynomial time complexity are unlikely to be found. Experimental evaluations showed that the proposed system performed well on realistic VM performance degradations, yielding over 30% savings in energy and up to 52% reduction in degradation.

We believe that the understanding of performance aware consolidation developed above will enable better workload consolidation. Additional open problems remain to be addressed in this space and further work is required to develop consolidation methods that operate in an online manner and place VMs near-optimally as and when they arrive for deployment.

References

[1] AWASTHI, M., SUDAN, K., BALASUBRAMONIAN, R., AND CARTER, J. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In *Proceedings of the International Symposium on High-Performance Computer Architecture* (2009).

[2] BAE, C. S., XIA, L., DINDA, P., AND LANGE, J. Dynamic adaptive virtual core mapping to improve power, energy, and performance in multi-socket multicores. In *Proceedings of the twenty-first International Symposium on High-Performance Parallel and Distributed Computing* (2012), ACM, pp. 247–258.

[3] BITIRGEN, R., IPEK, E., AND MARTINEZ, J. F. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the forty-first annual IEEE/ACM International Symposium on Microarchitecture* (2008).

[4] BOBROFF, N., KOCHUT, A., AND BEATY, K. Dynamic placement of virtual machines for managing sla violations. In *Proceedings of the International Symposium on Integrated Network Management* (2007).

[5] CHANDRA, D., GUO, F., KIM, S., AND SOLIHIN, Y. Predicting inter-thread cache contention on a chip multiprocessor architecture. In *Proceedings of the International Symposium on High-Performance Computer Architecture* (2005).

[6] CHO, S., AND JIN, L. Managing distributed, shared L2 caches through os-level page allocation. In *Proceedings of the thirty-ninth annual IEEE/ACM International Symposium on Microarchitecture* (2006).

[7] CHVATAL, V. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research 4* (1979), 233–235.

[8] DEAN, J., AND GHEMAWAT, S. MapReduce: A flexible data processing tool. *Communications of the ACM 53*, 1 (2010), 72–77.

[9] DUH, R., AND FÜRER, M. Approximation of k-set cover by semi-local optimization. In *Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing* (1997).

[10] ENVIRONMENTAL PROTECTION AGENCY, U. S. Report to congress on server and data center energy efficiency public law 109-431. Tech. rep., EPA ENERGY STAR Program, 2007.

[11] GANDHI, A., HARCHOL-BALTER, M., DAS, R., AND LEFURGY, C. Optimal power allocation in server farms. In *Proceedings of ACM SIGMETRICS* (2009).

[12] GOVINDAN, S., LIU, J., KANSAL, A., AND SIVASUBRAMANIAM, A. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the ACM Symposium on Cloud Computing* (2011).

[13] GREENBERG, A., HAMILTON, J., MALTZ, D. A., AND PATEL, P. The cost of a cloud: research problems in data center networks. *In ACM SIGCOMM Comput. Commun. Rev. 39*, 1 (January 2009), 68–73.

[14] HAMILTON, J. Cost of power in large-scale data centers. Blog entry dated 11/28/2008 at `http://perspectives.mvdirona.com`, 2009. Also in Keynote, at ACM SIGMETRICS 2009.

[15] HASSIN, R., AND LEVIN, A. A better-than-greedy approximation algorithm for the minimum set cover problem. *SIAM Journal on Computing 35*, 1 (2005), 189–200.

[16] IYER, R., ILLIKKAL, R., TICKOO, O., ZHAO, L., APPARAO, P., AND NEWELL, D. Vm3: Measuring, modeling and managing vm shared resources. *Computer Networks 53*, 17 (2009), 2873–2887.

[17] JIANG, Y., SHEN, X., CHEN, J., AND TRIPATHI, R. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In *Proceedings of the seventeenth International Conference on Parallel Architectures and Compilation Techniques* (2008).

[18] JIANG, Y., TIAN, K., AND SHEN, X. Combining locality analysis with online proactive job co-scheduling in chip multiprocessors. In *Proceedings of the International Conference on High-Performance Embedded Architectures and Compilers* (2010), pp. 6–8.

[19] KOH, Y., KNAUERHASE, R., BRETT, P., BOWMAN, M., WEN, Z., AND PU, C. An analysis of performance interference effects in virtual environments. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software* (2007).

[20] KOHAVI, R., HENNE, R. M., AND SOMMERFIELD, D. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *Proceedings of the thirteenth annual ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2007), pp. 959–967.

[21] MARS, J., TANG, L., HUNDT, R., SKADRON, K., AND SOFFA, M. L. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the forty-fourth annual IEEE/ACM International Symposium on Microarchitecture* (2011).

[22] MENON, A., SANTOS, J. R., TURNER, Y., JANAKIRAMAN, G., AND ZWAENEPOEL, W. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the USENIX International Conference on Virtual Execution Environments* (2005).

[23] NATHUJI, R., KANSAL, A., AND GHAFFARKHAH, A. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the fourth ACM European Conference on Computer Systems* (2010).

[24] RIGHTSCALE. RightScale scalable websites. `http://www.rightscale.com/solutions/cloud-computing-uses/scalable-website.php`.

[25] ROYTMAN, A., KANSAL, A., GOVINDAN, S., LIU, J., AND NATH, S. Algorithm design for performance aware VM consolidation. Tech. rep., Microsoft Research, 2013. Number MSR-TR-2013-28.

[26] SCHURMAN, E., AND BRUTLAG, J. Performance related changes and their user impact. In *O'REILLY: Web Performance and Operations Conference (Velocity)* (2009).

[27] SOARES, L., TAM, D., AND STUMM, M. Reducing the harmful effects of last-level cache polluters with an os-level, software-only pollute buffer. In *Proceedings of the forty-first annual IEEE/ACM International Symposium on Microarchitecture* (2008).

[28] SRIKANTAIAH, S., KANSAL, A., AND ZHAO, F. Energy aware consolidation for cloud computing. In *Proceedings of the workshop on Power Aware Computing and Systems* (2008), HotPower.

[29] TIAN, K., JIANG, Y., AND SHEN, X. A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In *Proceedings of the ACM International Conference on Computing Frontiers* (2009).

[30] TREVISAN, L. Non-approximability results for optimization problems on bounded degree instances. In *Proceedings of the thirty-third annual ACM Symposium on Theory of Computing* (2001).

[31] VERMA, A., AHUJA, P., AND NEOGI, A. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware* (2008).

[32] VERMA, A., AHUJA, P., AND NEOGI, A. Power-aware dynamic placement of hpc applications. In *Proceedings of the International Conference on Supercomputing* (2008).

[33] VMWARE. Vmware distributed power management concepts and use. `http://www.vmware.com/files/pdf/DPM.pdf`.

[34] ZHANG, J., SIVASUBRAMANIAM, A., WANG, Q., RISKA, A., AND RIEDEL, E. Storage performance virtualization via throughput and latency control. *Trans. Storage 2*, 3 (2006), 283–308.

[35] ZHAO, L., IYER, R., ILLIKKAL, R., MOSES, J., MAKINENI, S., AND NEWELL, D. Cachescouts: Fine-grain monitoring of shared caches in cmp platforms. In *Proceedings of the sixteenth International Conference on Parallel Architectures and Compilation Techniques* (2007).