Validating Heuristics for Virtual Machines Consolidation

Sangmin Lee* Rina Panigrahy Vijayan Prabhakaran Venugopalan Ramasubramanian
Kunal Talwar Lincoln Uyeda† and Udi Wieder
Microsoft Research Silicon Valley, Mountain View, CA 94043

Abstract

This paper examines two fundamental issues pertaining to virtual machines (VM) consolidation. Current virtualization management tools, both commercial and academic, enable multiple virtual machines to be consolidated into few servers so that other servers can be turned off, saving power. These tools determine effective strategies for VM placement with the help of clever optimization algorithms, relying on two inputs: a model of resource utilization vs performance tradeoff when multiple VMs are hosted together and estimates of resource requirements for each VM in terms of CPU, network and storage. This paper investigates the following key questions: What factors govern the performance model that drives VM placement, and how do competing resource demands in multiple dimensions affect VM consolidation? It establishes a few basic insights about these questions through a combination of experiments and empirical analysis. This experimental study points out potential pitfalls in the use of current VM management tools and identifies promising opportunities for more effective performance consolidation algorithms. In addition to providing valuable guidance to practitioners, we believe this paper will serve as a starting point for research into next-generation virtualization platforms and tools.

1 Introduction

Virtualization has become an important trend in data centers [11] and cloud computing [15, 16]. One of its attractive features is the ability to utilize compute power more efficiently. Specifically, virtualization provides an opportunity to consolidate multiple virtual machine (VM) instances running on underutilized computers into fewer hosts, enabling many of the computers to be turned-off, and thereby resulting in substantial energy savings. In fact, commercial products such as the VMware vSphere Distributed Resource Scheduler [19] (DRS), Microsoft System Center Virtual Machine Manager [18] (VMM), and Citirix XenServer [20] offer VM consolidation as their chief functionality.

VM consolidation tools typically have optimization algorithms to decide which virtual machine should be placed on which physical host. Research on VM consolidation has generated several clever heuristics for effective VM consolidation [12, 13, 14]. These heuristics take as input estimated resource requirements of the virtual machines and the capacities of the physical hosts, and decide which VM instance should be placed on which host, while trying to minimize the total number of physical hosts utilized.

In this paper, we take a step away from designing the most appropriate heuristic. Instead, we take a look at deeper, fundamental aspects that lie at the heart of VM consolidation. Specifically, we ask two questions about VM consolidation heuristics. First, what assumptions do these heuristics typically make about how virtual machines operate when hosted together, and to what extent do these assumptions hold in practice? Second, how sophisticated should the heuristics be in dealing with resource requirements that span different dimensions such as CPU, memory, and I/O bandwidth; in other words, to what extent can sophisticated heuristics improve the benefits of VM consolidation?

These questions and their answers have important implications for understanding how VM consolidation will work in practice. For instance, most heuristics assume that the virtualization platform provides perfect performance isolation; that is, resource utilization of a VM, and correspondingly its performance, will remain the same irrespective of whether it is run in isolation or along with other VMs. However, when this assumption is violated—the resource utilization drops because of adverse interference from other VMs, the performance of the applications hosted on that VM might be severely degraded. Even though a VM consolidation tool might be able

*Sangmin Lee is a graduate student at The University of Texas, Austin and was an intern at Microsoft Research Silicon Valley.

†Lincoln Uyeda is a member of Microsoft's VMM product group.

to tolerate a small degree of performance degradation by holding small amounts of resources as a reserve [9], severe degradation in performance can be counter productive to the goals of consolidation.

Similarly, VM consolidation heuristics vary in increasing sophistication depending on how they treat demands for multiple resource types. At one extreme, they might choose a single primary resource dimension to optimize for and ignore others; for example, a simple heuristic might only consider CPU requirements and ignore memory and disk usage. At the other extreme, heuristics might be influenced by resource demands across every resource dimension such as CPU, memory, and disk. The latter, more sophisticated heuristics have higher overhead, both for obtaining accurate estimates of the information they need and the time they take to compute the results. It is essential to understand what workload characteristics will lead to benefits commensurate with the additional overhead.

We answer these questions in this paper through a combination of experiments and empirical analysis. To answer the first question, we run controlled experiments on machines running Microsoft's Hyper-V hypervisor, using micro-benchmarks to examine how each of CPU, cache, network, and storage resources aggregate. The second question is answered by studying the effectiveness of multi-dimensional heuristics. We start with resource utilization data collected from a real compute cluster running Dryad jobs [8] and analyze the performance of five VM consolidation heuristics drawn from existing proposals.

This experimental study—even though specific to the selected workloads, heuristics, and virtualization environment—highlights many interesting properties of a virtualized system. We find that the performance of a virtualized environment depends heavily on many factors such as the resource type, number of virtual machines, and the quality of the workloads. For example, for a random write workload, disk utilization decreases as more VMs are co-hosted, whereas utilization improves in a solid state drive for the same workload. We also find that sophisticated, multi-dimensional heuristics provide significant returns for specific scenarios where the VM workloads have a mixture of complementing resource demands, but simpler heuristics suffice for other scenarios.

This paper is organized as follows. It starts with a brief background on a few representative VM consolidation heuristics in the next section. The two fundamental questions are then taken up in Sections 3 and 4 respectively. Finally, Section 5 summarizes the implications of our findings and concludes.

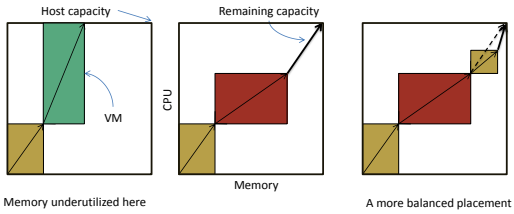

Figure 1: **Intuition for 2D-bin packing. The outer rectangle is the capacity of a host. A tight packing will get the VMs (smaller rectangles) as close to the top right corner as possible.**

2 VM Consolidation Heuristics

The problem of VM consolidation maps to the classical optimization problem called *vector bin packing*, where the hosts are conceived as bins and the VMs as objects that need to be packed into the bins. Each host is characterized by a d-dimensional vector called the host's *vector of capacities* $H = (h_1, h_2, ..., h_d)$. Each dimension represents the host's capacity corresponding to a different resource such as CPU utilization, memory utilization, or disk bandwidth. Similarly, each VM is represented by its *vector of demands* $V = (v_1, v_2, .., v_d)$. The goal is to place all the VMs in as few hosts as possible, ensuring that, across any dimension, the total demand of VMs placed in a host does not exceed the capacity of the host.

Vector bin packing is an NP hard problem. However, good heuristics with measurable performance are well known and appear in surveys and textbooks [5, 7]. We next outline a set of the common heuristics for the problem, describing them in increased order of sophistication. The heuristics we describe, or their variants, are used by research prototypes as well as real VM management tools (summarized in Section 2.3). This description helps us understand the inherent assumptions that underlie VM consolidation heuristics.

First Fit Decreasing: A natural heuristic for one-dimensional bin packing is FFD (First Fit Decreasing). This heuristic orders the bins and the objects in decreasing order of size. Starting with the first bin, it iterates over the objects, placing any object it can into the first bin till no more objects can be placed into it. It then considers the first bin to be filled and proceeds to the second bin, repeating the same procedure. It is known that this algorithm gives a 11/9 worst case approximation [5] if all bins are of equal size.

2.1 FFD-based heuristics

If the placement problem is effectively constrained by a single resource, say all VM's are CPU bound, then the problem is in essence one dimensional and FFD is the algorithm of choice. If the problem instance is constrained by more than one dimension then we need some generalization of FFD for multiple dimensions. The typical approach is to map the vector of capacities and demands into a single scalar called the *Volume* of the vector, and then perform a one dimensional FFD based on the calculated volumes. We call these types of algorithm *FFD-based*.

There are many ways one can compute the volume. One option is to set the volume as the product of the values; i.e., $Volume(V) = \Pi_i v_i$. We call the FFD algorithm based on this function the *FFDProd* algorithm. An attractive feature of this algorithm is that the order in which the VM's are sorted is independent of the units used for measuring each dimension (as long as the same unit is used across all VMs). A second approach, which we call *FFDSum*, is to set the volume to be some weighted sum of the values; i.e., for each dimension i, $Volume(V) = \sum_i w_i v_i$, where w_i is the assigned weight to that resource. The weights reflect the scarcity of the resource. A suitable definition for the weight could be the ratio between the total demand for resource i and the capacity of the host, expressed as $\sum_{VM} v_i/h_i$.

The main drawback of FFD-based algorithms is that they do not take into account correlations across dimensions. Consider a case where there are two kinds of VMs, one is memory intensive and one is CPU intensive. An FFD-based algorithm would order the VMs such that all VMs of a certain type are first, followed by all the VMs of the second type. As a result it would try and pack the maximum number of CPU intensive VMs in each host, possibly preventing the host from accepting any other VMs. A better solution would have been to pack memory intensive and CPU intensive VMs on the same host, keeping resource allocation across dimensions as balanced as possible.

2.2 Dimension-aware heuristics

The next set of heuristics, which we call *dimension-aware*, take advantage of such complimentary requirements for different resources. Their key intuition can be understood from the two dimensional example illustrated in Figure 1. The capacity of a host is represented using a rectangle, its sides representing the capacity across the two dimensions. Each VM is also a vector and as a VM is added to a host, their corresponding demand vectors are added with the constraint that the point corresponding to the total load given by the sum of these vectors must lie within the rectangle. To best utilize the capacity of the host along both dimensions (resources) it is desirable that the final sum of the demand vectors end as close as possible to the top right corner of the rectangle; it is undesirable to end in a situation where the combined vector hits the upper edge or the side edge while wasting space in the other dimension. This implies that at any point it is desirable to choose the next VM in such a way that the point representing the current load moves towards the top right corner.

We present two heuristics that achieve this, one picks the next vector by comparing its direction to that of the direction to the top right corner. The second heuristic picks the next vector by comparing its distance to the top right corner.

Dot-Product ($\mathcal{D}otProduct$) When choosing the next VM to place into an open host, this heuristic takes into account not only the VMs total demand, but also how well its demands align with the remaining capacities. It does this by looking at the angle (dot product) between the vector of remaining capacities along the dimensions and the vector of demands for a VM. At time t let $H(t)$ denote the vector of *remaining* or *residual* capacities of the current open host, i.e. subtract from the host's capacity the total demand of all VMs currently assigned to it. It places the VM that maximizes the dot product $\sum_i w_i v_i h(t)_i$ with the vector of remaining capacities without violating the capacity constraint. The weights w_i are calculated in the same manner as described in the FFDSum algorithm.

Norm-based Greedy: ($\mathcal{L}2$) This heuristic looks at the difference between the vectors v and h under a certain distance metric (norm), instead of the dot product. For example, for the ℓ_2 norm distance metric, from all unassigned VMs, it places the VM v that minimizes the quantity $\sum_i w_i(v_i - h_i(t))^2$ and the assignment does not violate the capacity constraints.

2.3 Related heuristics from literature

There are several VM consolidation heuristics currently used in research prototypes and real VM management tools. We briefly outline how they are related to the heuristics described in this section. There are also other heuristics for server consolidation in non-virtualized environments [2, 4, 3, 10], which we omit from this study.

Sandpiper [14], a research system that enables live migration of VMs around overloaded hosts, uses a heuristic corresponding to *FFDProd*, taking the prod-

uct of CPU, memory, and network loads. Another work from IBM research [13], an application placement controller for data centers, performs application assignment/consolidation handling resource demands for CPU and memory. This work combines the two dimensions into a scalar by taking the ratio of the CPU demand to the memory demand. Even though this heuristic does not fit into the *FFDSum* or *FFDProd*, it shares the similar characteristic of ignoring complementary distributions of resource demands in different dimensions.

Microsoft's Virtual Machine Manager [18] internally uses the dimension-aware, Dot-Product and Norm-based Greedy, heuristics described above. A recent research work [12] proposes to use Euclidean distance between resource demands and residual capacity as a metric for consolidation, a heuristic analogous to Norm-based Greedy.

3 Validating Assumptions

The heuristics we just described make critical assumptions about how resource utilization and performance of virtual machines aggregate when they are hosted together. As mentioned earlier, a basic assumption they make is perfect performance isolation: Each VM's resource consumption even when hosted with others would be the same as estimated from an individual execution; or, in terms of performance, the throughput realized by each VM when hosted together is the same as the throughput it achieves when hosted individually. If this assumption holds, then the residual capacity in a host for a given resource can indeed be computed by subtracting the sum of the estimated resource requirement of each of the VMs assigned to that host from the host's total capacity.

In practice, however, the amount of resources a VM utilizes (and the performance it realizes) might not be preserved when co-hosted with other VMs. Performance degradation occurs due to multiple reasons. 1. *Virtualization overhead:* There might be overhead or bottlenecks introduced by the hypervisor that consumes some of the host's capacity. For instance, the scheduler and other background processes in the hypervisor and the host OS might consume CPU cycles and I/O bandwidth. 2. *Cross-interference:* Multiple VMs running together could interfere with each other, degrading their performance or consuming more of the resources than they would if hosted individually. For instance, contention for a shared L2 cache or cache flushes that occur as a result of context-switching decreases the effectiveness of caching and adversely impacts VM performance. 3. *Resource over-subscription:* Even in the absence of overhead and interference, a VM's resource utilization during actual execution might be different from the estimate due to errors in the estimation process or unpredictable changes in the VM's workload.

Since the above causes are often unavoidable, VM consolidation tools expect and even tolerate some amount of performance degradation for co-hosted workloads. The key concern then is the extent of performance degradation. The most common method to tolerate performance degradation, employed in Microsoft System Center VMM for example, is to provide for some slack while allocating resources. For example, if the residual capacity of a host goes below a certain threshold, say 20% of the total capacity, then VMs are not allocated to that host anymore. This slack accommodates for certain amount of deviation in how resources are consumed and tolerates performance degradation to a limited extent. However, if the performance degradation due to cross-interference or resource oversubscription is too steep, the system could start thrashing, and the VMs would be unable to complete their tasks in time. A pathological scenario occurs, for example, when the co-hosted VMs take even longer time to finish than the time taken to run the VMs sequentially in isolation—entirely defeating the goal of saving power through consolidation.

Finally, it is also essential to understand whether resource allocation, especially when performance degradation kicks in, is fair across multiple VMs. For example, assume that a VM consumes excessive amount of computation resources, say 50%, than its estimated value of say 20%. When computation resources are scarce, the hypervisor would meet this VMs requirement by stealing resources allocated to other VMs. If the overall resource allocation is not fair, an unfortunate VM could be starved off its resource cycles. In general, accommodating a non-uniform degree of performance degradation between VMs is hard as simple solutions, such as reserving a slack, do not produce desired results[1].

In summary, the above assumptions, critical for current VM consolidation heuristics to work well, translate to the following three questions: 1. Do resource utilization and performance aggregate additively, according to their estimates, when multiple

[1] A sophisticated hypervisor might handle this by reserving resource to each VM according to its estimated needs (perhaps with a slack) and ensuring that each VM gets its promised quantity of each resource. In that case, oversubscription would only affect the VM that has exceeded its resource requirement. Unfortunately, current hypervisors only support resource reservation for some resource types.

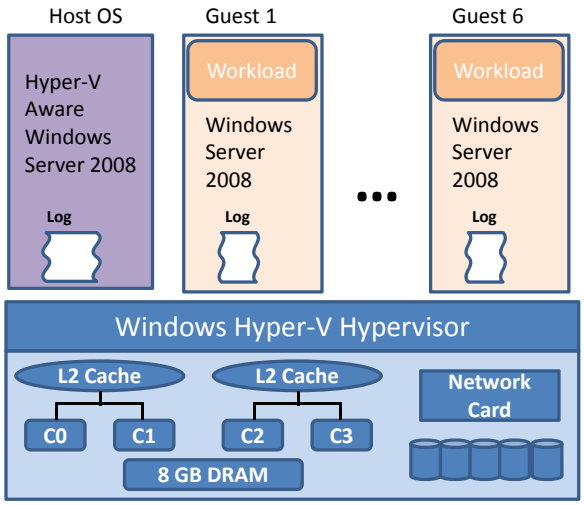

Figure 2: **Experimental setup.**

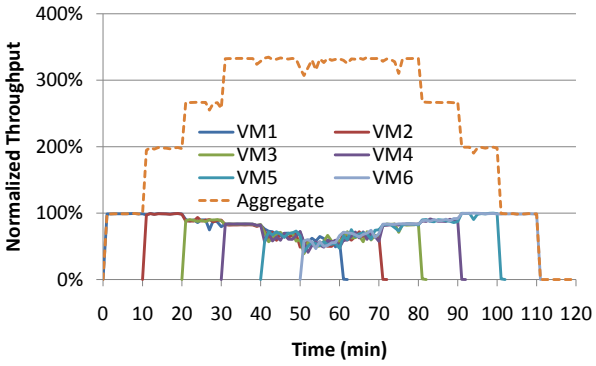

Figure 3: **Normalized CPU throughput of each VM with time. A new VM workload was started after every 10 minutes for up to 6 VMs.**

VMs are hosted together? 2. When does performance degradation kick in, and is the degradation gradual? 3. Under high utilization, are resources shared fairly so that any performance degradation would affect all VMs equally? In the rest of this section, we examine these questions separately for four types of resources—namely CPU, cache, network, and storage.

3.1 Experiments setup

We performed our experiments on a computer with the following hardware specification: a 2.83 GHz Intel Core 2 Quad Processor, 8 GB RAM, a Broadcom NetXtreme Gigabit Ethernet card, and 4 spare SATA ports. The processor has four cores, of which, each pair shares an L2 cache of 6 MB. We added additional storage hardware to this computer, consisting of either four Seagate Momentus 7200 RPM SATA drives or three Intel X-25M MLC SSDs. We also repeated some experiments on an alternative setup, a computer with four 1.9 GHz AMD Opteron 6168 processors with 12 cores each (total 48 cores) and 128 GB total RAM. We used the second setup, to verify the observations of our experiments in a multi-processor machine with much a larger number of cores.

We setup the computers with a 64-bit Windows 2008 Server host operating system, which supports virtualization through Microsoft's Hyper-V platform. Hyper-V uses the native virtualization support (Intel VT and AMD-V) provided by the processor. The guest virtual machines were also configured with the 64-bit Windows 2008 Server OS. We statically allocated 1 GB RAM to each guest virtual machine because Hyper-V does not yet support dynamic memory allocation. To avoid interference, we used a separate hard disk to store the root volume of the host and guest virtual machines.

We used the Windows Logman to collect and manage performance counter values. We collected a number of performance counters including the percentage of total processor runtime, bytes sent and received over the network interface, and average disk usage as bytes per second both from the host and the virtual machines. Figure 2 illustrates the architecture of the machine and the experimental setup.

3.2 CPU

Workload. We first present results from the CPU experiments. To study CPU performance, we used a matrix multiplication program written in C as a micro-benchmark. This program repeatedly performs a matrix multiplication of the form $A^T A$ on a matrix A of R rows and C columns. We chose this workload because it allows us to precisely control the tradeoff between memory/cache utilization and computation cycles. Choosing a small matrix, and executing many iterations of the multiplication, enables us to minimize the effects of cache contention. On the other hand, increasing the matrix size allows us to study the effects of cache contention in a controlled manner.

Procedure. We examined how computation resources aggregates by hosting up to six simultaneous VM instances on the four-core machine. Each instance ran the matrix multiplication workload on a 128x128 matrix for 60 minutes. However, we staggered the start time of the workload by 10 minutes for each VM. That is, at the start of the experiment, only one VM starts running the workload; the second VM starts 10 minutes later, and so on. The staggering allows us to examine resource aggregation for different numbers of active VMs.

Results. We measure and plot the throughput realized by each VM workload in terms of the number of matrix multiplications completed per second. The throughput of different VMs is an indication of how resources are shared between the different VMs. If resources combine additively then the throughput of the existing VMs remains the same as new VMs are added to the system, and the new VMs also realize the same throughput. However, if there are performance bottlenecks then the throughput will decrease.

Figure 3 plots the throughput versus time as each of the six VMs run their workload. We normalize the throughput with respect to the maximum throughput seen by a single VM. Since the machine is quad-core we expect to achieve, in the best case, four times the maximum throughput achieved by a single VM. In fact, the figure shows this additive increase in throughput for the first two VMs. They both achieve close to 100% normalized throughput each with minimal interference from other overheads.

However, as more VMs are added to the system, the throughput decreases for all the VMs. Throughput degradation is expected to occur when there are more than four VMs because the system needs to share its four cores with 5 or 6 VMs. However, the throughput degradation for the 3- and 4-VM cases is somewhat surprising. Figure 3 shows that sharing 4 VMs only increases the aggregate throughput 3.2 fold compared to 1 VM. It appears that there are bottlenecks that decrease the effective sharing of 4 cores with 4 VMs. Subsequent experiments showed that this bottleneck is due to cache contention created by processes running on the host OS. When there are three VMs running on one core each, host OS processes run on the fourth core and contend for the L2 cache that the fourth core shares with another core allocated to one of the VMs.

Figure 4 plots the average normalized throughput seen by each VM. Each cluster of bars in the figure corresponds to a certain number of active VMs. The bars plot the normalized throughput realized by each VM in that cluster; the exact identification of each VM is not relevant. Figure 4 summarizes the trends captured in Figure 3. It also shows that where performance degradation occurs, it affects all VMs almost equally—that is, CPU allocation is fair. Note that Hyper-V does not pin VMs to specific cores. Consequently, the performance degradation affected all VMs equally over time even though at specific moments different VMs saw different effects.

Multi-processors experiments. We also performed CPU experiments on the 4-CPU, 48 cores machine using the same procedure. In this case, we ran up to 20 simultaneous VM instances. Moreover, each VM instance internally ran four parallel instances of the matrix multiplication workload, resulting in 80 parallel workloads on the 48 core machine. We started the VM instances in the same staggered manner as earlier, but in groups of 4.

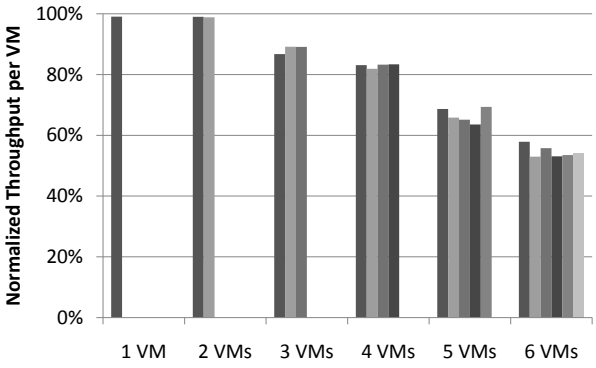

Figure 4: **Normalized CPU throughput realized by each VM as the number of co-hosted VMs vary. Each clustered set of bars shows the variation in throughput across VMs; the exact identification of the VMs—hence the color of the bars—is not relevant.**

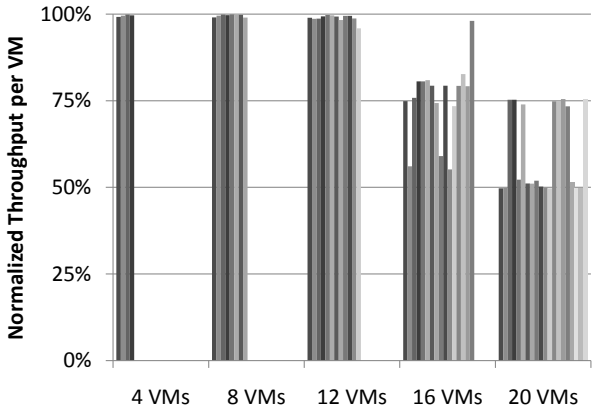

Figure 5: **Normalized CPU throughput realized by each VM as the number of co-hosted VMs vary in the multi-processors experiments. As in Figure 4, the color of the bars is not relevant.**

Figure 5 plots the average normalized throughput seem by each VM in the multi-core experiment, each cluster of bars in the figure corresponding to a certain number of active VMs. As before, the exact identification of each VM (bar in the graph) is not relevant. Figure 5 predominantly confirms the trends captured in the previous experiments. There is no real performance degradation for up to 12 VMs, which has 48 parallel workloads, equal to the number of cores in the machine. A gradual but proportional degradation in throughput is incurred when more VMs are added, about 75% on average for 16 VMs case and 60% average for 20 VMs case.

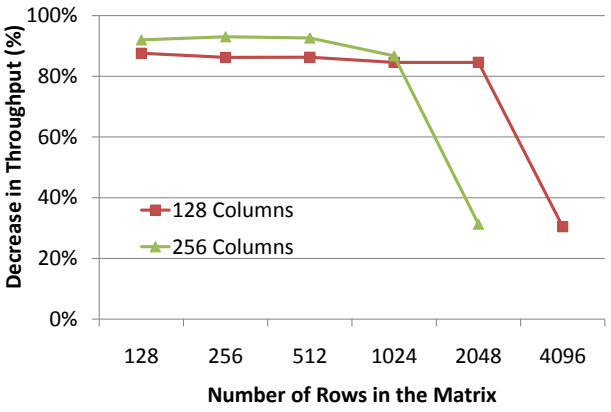

Figure 6: **Throughput degradation experienced by the shared workload compared to the corresponding isolated workload for different matrix sizes.**

However, fairness in resource utilization and the corresponding uniformity in performance degradation, seen in the small-cores experiments, is noticeably absent. Figure 5 shows that there is a discrete granularity at which resources are utilized, causing about 12 VMs (in the 20-VMs case) to aggregate around the 50% throughput degradation point while the other 8 VMs aggregate around the 75% point, as if the first 12 VMs share the cores on two processors while the remaining 8 VMs shared the other two processors, evenly in both cases. This indicates Hyper-V's allocation of cores maintains fairness for each processor, but perhaps not as effectively across multiple processors.

3.3 Cache

Workload and procedure. We next examine the effects of cache contention on shared VM workloads. We performed this study using the same matrix multiplication workload as in the CPU experiments, but with varied matrix sizes. In each experiment, we ran up to four simultaneous VM instances, staggering the start time of the workload by 10 minutes as before. Since each pair of cores in our test machine has a shared 6 MB L2 cache, VM instances hosted on cores with shared cache experience cache contention. We varied the matrix size, changing the number of rows as well as columns, to simulate different severity of cache contention.

Results. We measure and plot the degree of throughput degradation caused by cache contention. We define this degree as a ratio of the average throughput of the shared, four-VMs workload to the average throughput of running the workload on a single isolated VM.

Figure 6 plots the percentage of throughput degradation for different sizes of the matrix. It shows that a small degree of performance degradation (less than 20%) occurs even for small matrixes; these matrixes fit into the L2 cache and should not ideally incur the penalty of cache misses. But as we already observed this performance degradation in the CPU experiments, it stems from sharing the CPU and cache with the root VM. However, a remarkably larger percentage of performance degradation, where the throughput drops to about a third of the throughput in the isolated case, can be noticed as the matrix size increases beyond a certain threshold. This experiment evidently shows that cache contention could degrade performance of shared workloads precipitously.

A closer look at a specific point in Figure 6 helps to understand this cache-related behavior better. Figure 7 plots the normalized throughput produced by each VM running the matrix multiplication workload for a 2048 x 256 matrix, when different number of VMs are co-hosted. Corresponding to the respective point in the previous figure, the normalized throughput of each VM when four-VMs are co-hosted can be seen to be remarkably low, about 30% of the peak thoroughput. Essentially this brings out a pathological scenario—the shared, 4-VMs workload takes a longer time to complete compared to running two instances of 2-VMs workload sequentially.

Figure 7 also provides additional insights by showing the behavior when two or three VMs are co-hosted. In the two-VM case, one of the VM experiences significant throughput degradation (due to cache contention from the root VM running in the other core that shares its cache) while other does not. Similarly, the three-VM case also shows non-uniform severity in throughput degradation. Here, the VM instance that shares the cache with the root VM experiences less throughput degradation compared to the other two VMs that contend for the shared cache with each other.

In summary, these experiments confirm that the hypervisor does not provide performance isolation from cache contention, and neither does it allocate cache resources fairly to ensure proportional resource utilization.

3.4 Network

Workload. To measure the performance of the network components, we modified the same matrix multiplication benchmark by interspersing it with network packet transfers. Specifically, the network-benchmark transmits one UDP packet to a second

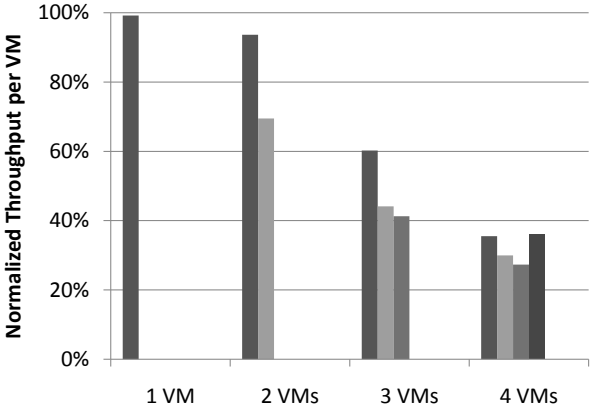

Figure 7: **Normalized throughput of each VM for the 2048 x 256 matrix multiplication workload.**

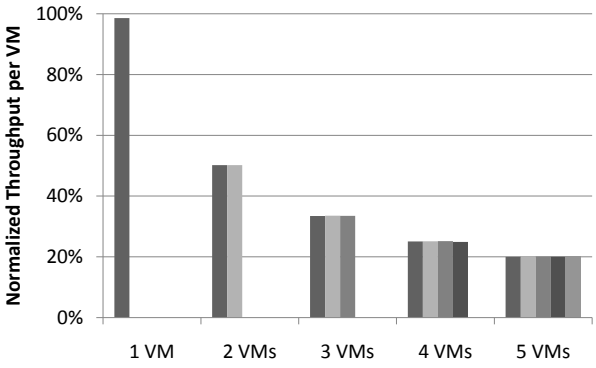

Figure 8: **Normalized throughput realized by each VM as the number of VMs vary for the high utilization network workload, which saturates the network card.**

computer after each iteration of the inner-most loop of the matrix multiplication. We created two versions of this network workload. A *light workload* which sends a packet of size 4096 bytes in a single thread. This workload is not sufficient to saturate the network. Therefore, we created a multi-threaded *heavy workload*, which sends packets of size 65000 bytes (largest allowed) in 5 parallel threads.

Procedure. We examined the aggregation of the network resource by hosting 5 simultaneous VM instances. The workload duration for each instance was 25 minutes, and the start-time of the workload on different VMs was staggered by 5 minutes, similar to the CPU and cache experiments.

Results. We again measure how resource allocation occurs as a function of the throughput realized by each VM. Figure 8 plots the network throughput seen by each VM during different phases of the experiment, where different number of VMs were hosted. The throughput plotted in Figure 8 is normalized with respect to the maximum throughput seen by a single VM, which was about 935 Mbps measured at the application level—quite close to the maximum capacity of the gigabit network card.

Figure 8 shows that throughput degrades as more VMs are added. This behavior is expected because VMs are running the high workload, which saturates the network card. The throughput degradation is also almost optimal; for 5 VMs, each VM achieves about 20%, a fair share of the total network throughput.

Figure 9, on the other hand, plots the throughput (not normalized) for the low workload. In this workload, each VM sends less than 120 Mbps of network traffic. Therefore, the network card has sufficient capacity to support the combined network requirements of all 5 VMs. Yet, there is a clear throughput degradation visible even when two VMs are active; the throughput drops from about 120 Mbps per VM to about 90 Mbps per VM. The throughput degradation continues as more VMs are added, albeit more gradually, settling at about 75 Mbps for the 4- and 5-VMs cases.

The above result for network throughput is surprising especially because the previous high-workload experiment indicated fair sharing with little throughput loss. Closer examination of the anomalous throughput degradation for the low workload indicates the following reason: Once a network *send* is issued by a process, there is a certain delay before the send returns control back to the process[2]. This delay increases sharply as another VM is added to the system. Subsequent additions of VMs increase this delay further, albeit to a lesser extent. Overall, the delay varies from about 35 μs to 50 μs per packet. The precise cause of this behavior requires further examination. For the purposes of this study, this experiment indicates that network throughput does not aggregate perfectly but incurs a mild performance degradation.

3.5 Storage

Workload. Finally, we measure the utilization of storage devices under multiple VMs using a simple sequential and random write workload. The sequential write benchmark writes 10 MB chunks sequentially to a 20 GB file from start to end. The random write benchmark uses two threads to issue 4 KB writes to random locations on a 20 GB file. In order to ensure that the file system cache is cold, all the benchmarks

[2]We observed this from the fact that we were unable to saturate the network using a single thread issuing network packets, even sending packets continuously with no intermittent computation and at the maximum packet size.

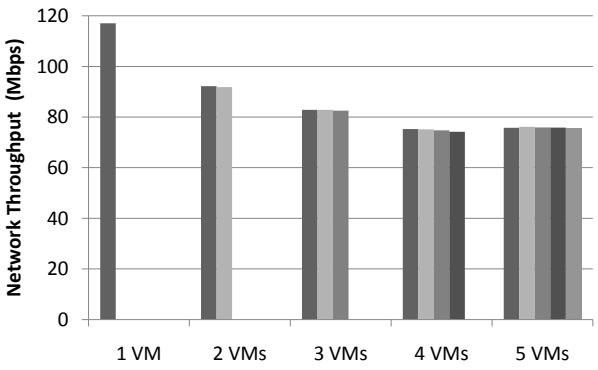

Figure 9: **Normalized throughput realized by each VM as the number of co-hosted VMs vary for the low utilization network workload.**

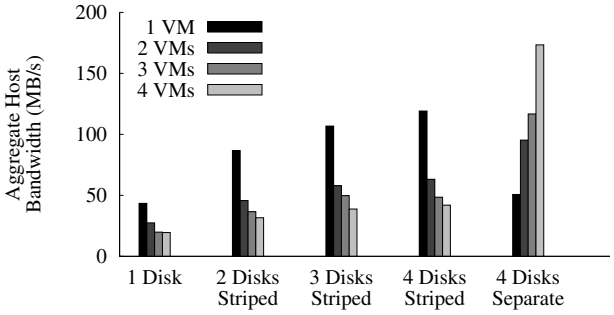

Figure 10: **Sequential write disk bandwidth.**

are run after a clean mount of the volume on which the accessed file is stored. We run these workloads on disks and Solid State Drives (SSD).

Procedure. In all our experiments, virtual disks are exported as storage devices to the VMs. Virtual disks are simply files that are pre-allocated on the physical disk or SSD on the host. We evaluated the storage performance by running between 1 to 4 VMs under different configurations: (a) all the virtual disks are stored on a simple volume on a single disk or SSD, (b) all the virtual disks are stored on a striped volume over a set of 2 to 4 disks or 2 to 3 SSDs, and (c) each virtual disk is stored on a separate physical disk. We ran each experiment 5 times and averaged the results.

Disk results. Figure 10 presents the results for disks, where the Y-axis plots the average disk aggregate bandwidth observed by the host. X-axis presents 5 clusters of bars one for each of our configuration, where each cluster has 4 bars; each bar in a cluster represents the aggregate host disk bandwidth when a specific number of virtual machines were running the same workload.

We make two observations from this figure. First, on configurations where the disks are shared, the disk bandwidth decreases with more VMs. For example, 2 striped disks give about 86 MB/s with a single VM whereas it drops to as low as 45 MB/s with 2 VMs. This happens because of the competing sequential stream of writes from two VMs, which result in a random write workload to the underlying disk storage. We see a similar effect on all the configurations where the disks are shared. However, in our last configuration, where each disk is allocated separately per VM, no I/O interference occurs and therefore, the aggregate disk bandwidth increases.

Second, we notice that the disk-bandwidth drop from 1 VM to 2 VMs is much greater than the drop from 2 to 3 VMs or 3 to 4 VMs. Specifically, for the striped disks configurations, the average bandwidth reduction from 1 to 2 VMs is 49%, whereas the drop from 2 to 3 VMs is 10-20%.

Figure 11 presents results for the random write workload. Surprisingly, even for random workload overall disk bandwidth drops with more VMs. We analyzed the disk offsets to which random access requests are issued and noticed that on an average, the seek distance between two random writes increased with more virtual machines, which could result in the decreased disk bandwidth. Overall for the random write workload, we observe trends similar to that of sequential write workload.

SSD results. Figures 12 and 13 present the sequential and random write results from similar experiments on SSDs. From Figure 12, we observe that the aggregate sequential bandwidth drops with multiple VMs even when running on SSDs, even though SSDs have better random access IOPS than disks; however, the percentage of reduction in bandwidth from 1 VM to 2 VMs is smaller than that of the disks. Specifically, for striped SSDs, the average reduction in bandwidth from 1 VM to 2 VMs is about 20% whereas it is 49% on disks. From Figure 13, we notice that the aggregate random write bandwidth increases with multiple VMs (as opposed to the decreasing trend on disks). This is because of the better random access performance of SSDs.

In summary, since hard disk drives are sensitive to the sequentiality of the workload, sharing disks among multiple VMs will result in bandwidth reduction. Surprisingly, this applies for random workloads and even for SSDs, to a certain degree. In terms of fairness, I/O bandwidth measured at each VM guests for each of the above configuration shows that the bandwidth allocation is proportional.

3.6 Implications

The above experiments analyzed resource aggregation, performance degradation, and fairness of re-

Resource type	Is resource consumption additive at low utilization?	Is performance degradation gradual at peak utilization?	Is resource allocation fair at peak utilization?
CPU	Yes	Yes	Yes
Cache	Yes	No	No
Network	No	Yes	Yes
Storage	No	No	Yes

Figure 14: **Summary of characteristics of resource-performance tradeoffs in a virtualized system.**

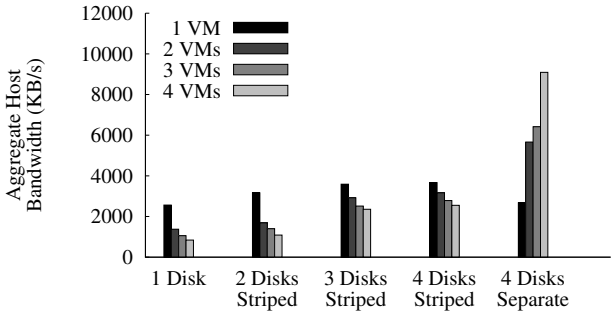

Figure 11: **Random write disk bandwidth.**

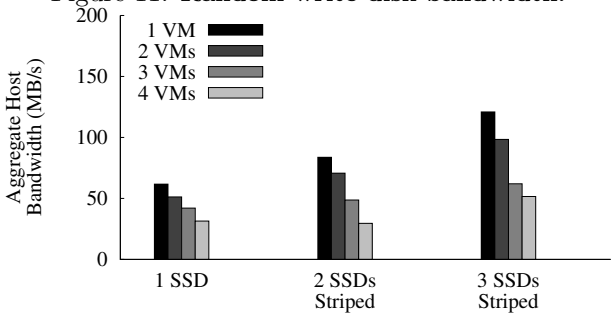

Figure 12: **Sequential write SSD bandwidth.**

source allocation for four important resource types. The key conclusion we can draw from this study is that the specific behavior is dependent on the type of resource and the quality of the workload. For computation, resources are additive at low utilization levels. But performance degradation kicks in gradually when the number of co-hosted VMs or their CPU demand exceed. However, when the memory foot-print of the workload is large and the pressure on cache is higher, it appears that, cache contention can lead to precipitous drops in performance. For network, a mild performance degradation occurs due to additional delay in network packet processing. However, VMs are able to utilize the entire available capacity even under high levels of resource contention. Finally, for storage, performance degradation is quite steep, especially for workloads with sequential I/O. The degradation is milder for random workload or when solid-state drives are used instead of the conventional mechanical disks. Overall, VMs are unable to share storage and cache resources as aggressively as network and computation resources. Finally, re-

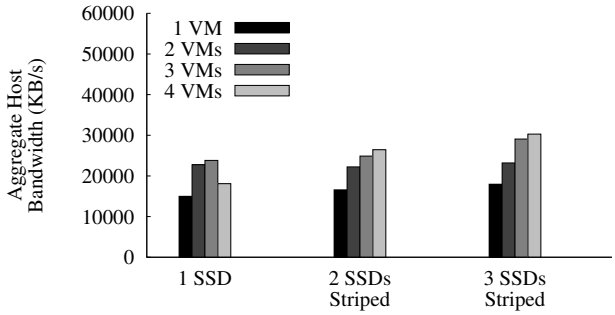

Figure 13: **Random write SSD bandwidth.**

source allocation for most resources and workloads we studied was fair, except for cache and perhaps for multi-processor systems with large number of cores, indicating that starvation is mostly a non-issue.

These observations have the following implications for VM consolidation. On the positive side, simple techniques that can account for mild performance degradation, such as reserving an unallocated *slack* capacity as proposed in [9], might be sufficient for hosting VMs with computation and network intensive workloads. Even if the actual resource usage differs from the estimate by a big quantity, the resulting performance degradation will be commensurate with the difference and be spread more or less equally over all VMs. On the negative side, consolidating VMs which rely heavily on the storage component or sensitive on the size of cache might be counter productive.

One approach to handle resource types that show steep performance degradation when shared is to enforce strict reservations on the quantity of resources allocated to each VM. In current virtualization platforms, reservation can be done at a coarse granularity by allocating one or more cores with shared caches to each VM exclusively or by allocating a separate disk to each VM. This approach also comes with a limitation on the maximum number of VMs a computer can host, determined by the number of cores or number of disks the hardware supports. Resource reservation at a finer granularity, such as a fraction of a device, is supported only for CPU and memory.

The insights drawn from this study calls for better I/O and cache performance isolation in virtualization platforms. A recent work called mClock [6] addresses the variability in I/O throughput in hy-

pervisors and proposes new mechanisms to enforce proportional fair-sharing for I/O. While this work is a step in the right direction, it needs to account for locality in I/O requests in order to avoid performance interference when I/O workloads are combined.

Cache performance isolation is an even more difficult problem to address. Current VM consolidation tools do not take into account the cache architecture of the servers and cache-sensitivity of their workloads. While a few hardware-level solutions have been proposed for cache performance isolation, dynamic page coloring for example, they are difficult to be incorporated into hypervisors. Even so, the processor and cache architecture are rapidly evolving: newly planned systems have a high degree of asymmetry in cache layout, more tiers of shared caches, and irregular features, for instance no cache coherency [17]. Effective VM consolidation in future data centers and cloud computing platforms will depend even more on addressing this issue.

4 Evaluating Heuristics

In this section, we empirically compare the various heuristics presented in Section 2. We seek to address the following questions, to guide the choice of heuristic in any given setting. 1. How well do the simpler, FFD-based heuristics perform? 2. Are the dimension-aware heuristics better, and if so, by how much? 3. What properties of the inputs affect the answers to these questions?

Towards this end, we evaluate the heuristics FFDProd, FFDSum, DotProduct, and L2 on some real, and some synthetic workloads.

4.1 Real workloads

We use data from the Dryad computing cluster [8] for this study. Dryad represents a typical map-reduce type of computational framework, a common source for virtualized workload in could computing clusters such as Amazon EC2. We recorded average usage in 5 resource dimensions namely CPU, Memory, Disk, Inbound Network traffic, and Outbound Network traffic, for four different sets of processes running on the Dryad cluster. The four sets of processes come from four large Dryad jobs implementing the pagerank algorithm (PgRank), a wordcount application (WordCount), a seive algorithm for finding primes (Primes), and a machine learning based click-bot detection algorithm (Clkbot). For each process in each job, the resource usage is averaged over measurements taken every 10 seconds. We consider a VM for each process.

Table 15 summarizes the number of VMs and the average normalized demands for each of the four sets of VMs. The capacities of all hosts are set to 400% CPU (4 cores), 4 GB of memory, 50 MBps Disk bandwidth, and 128 MBps network bandwidth. The demands in Table 15 are shown as a fraction of these capacities. Thus we can see, for example that PgRank is memory-intensive while Primes is CPU-intensive.

In order to analyze the performance of the heuristics, we need a base line. An immediate problem in this approach is that vector bin packing problem is NP-hard and thus it is difficult to find the optimum solution (OPT). Instead we use a known lower bound. For any dimension, the ratio of the total demand in the dimension, to the bin capacity in that dimension can be shown to be a lower bound on OPT. We let LB denote the maximum over all dimensions, of this ratio. We then chose the metric *percentage overhead* defined as $100 * (ALG - LB)/LB$, where ALG is the number of bins used by a specific algorithm, to compare the effectiveness of the heuristics. Since the experiments we do will have widely varying optimal number of hosts, we normalize our results relative to the lower bound, and compare how much worse the heuristics perform relative to the best possible.

We first compare the heuristics on each set of VMs corresponding to each Dryad job. Figure 16 reports the results. It shows that for the pagerank set, the FFDSum does much better than FFDProduct, and is pretty close to the lower bound. The results for the other sets are similar. The dimension-aware heuristics do not offer further improvement.

To study the efficacy of the heuristics for more heterogenous inputs, we construct inputs by mixing these sets of VMs together. To account for the variation in the number of VMs per set, we replicate smaller sets so that each set contributes approximately the same number of VMs to the input. The merged input consists of 1,2,10 and 7 copies of PgRank, WordCount, Primes and Clkbot respectively, leading to a total of 12021 VMs. We also consider Clkbot+Primes consisting of 1 copy each of Clkbot and Primes, giving us 692 jobs. Since the Clkbot is disk-intensive and Primes is CPU intensive, these give us negatively correlated dimensions. Similarly, we consider Clkbot+PgRank, consisting of 7 copies of PgRank and 1 copy of Primes, leading to 6034 VMs.

Figure 16 shows that for the mixed sets of processes, dimension-aware heuristics, DotProduct and L2, indeed make a difference and show less performance overhead compared to the dimension-less heuristics, FDProd and FDSum. Although, there is no clear winner between DotProduct and L2. The

Process	#VMs	Avg. CPU	Avg. Memory	Avg.Disk	Avg. Network_tx	Avg.Network_rx
PgRank	3137	0.2845	0.3804	0.0253	0.0533	0.0572
WordCount	1598	0.3000	0.2702	0.0138	0.0007	0.0003
Primes	279	0.3364	0.0144	0.0085	0.0078	0.0004
Clkbot	414	0.1469	0.0308	0.1713	0.0574	0.0753

Figure 15: **Characteristics of set of Inputs. Average values above are fractions of the capacities chosen, which are 400% of CPU, 4GB of memory, 50MBps disk bandwidth, and 128MBps network bandwidth.**

difference indeed looks substantial in relative terms. The values of the lower bound LB, 596, 3217, 104, and 1319 respectively in the four cases shown in the graph adds more insight in to the absolute performance. In the case of Clkbot+PgRank, the 10% difference between FFDSum and L2 translates to 132 fewer active hosts for L2. On the other hand, in the case of Clkbot+Prime the absolute benefit of using dimension-aware heuristics over dimension-less heuristics is only about 7 hosts.

4.2 Synthetic workloads

To further understand the relation between the performance of the heuristics and the correlations between the demands in the various dimensions, we evaluate the heuristics on some synthetic workloads as well, where we can control the correlations carefully. We consider four classes of synthetic inputs. We fix the number of dimensions to two. The first three classes are randomly generated with independent dimensions, positively correlated dimensions, and negatively correlated dimensions respectively.

Caprara and Toth [1] proposed ten classes of inputs to benchmark the effectiveness of two-dimensional bin-packing algorithms. Our three randomly generated classes are chosen from those. The first input class, denoted Random has the size of each bin set to $h = 150$. The demand for each VM, in each of the two dimensions is drawn uniformly and independently at random from $[20, 100]$. The next two input class denoted Positive and Negative, have correlated dimensions. In both cases, $h = 150$ as in Random, and the demand in the first dimension v_1 is drawn randomly and independently from $[20, 100]$. Demand in the second dimension v_2 is sampled from $[v_1 - 10, v_1 + 10]$ for class Positive and from $[110 - v_1, 130 - v_1]$ for class Negative. We sample 200 VMs in each case which defines our input. Finally, the fourth input class NegativeSynth is an artificially created example to show the extent of the worst case difference between the heuristics. Here capacity is set to a 100 in each of the two dimensions. There are two sets of jobs: a set of 1000 jobs with sizes $(20, 1)$ and another set of 1000 jobs with sizes $(3, 20)$. The results are resistant to small amounts of noise being added to these sizes. It turns out that the optimal solution (OPT) can indeed be computed for these classes of input. Therefore, we use OPT as the lower bound LB in the definition of percentage overhead. Figure 17 summarizes the results.

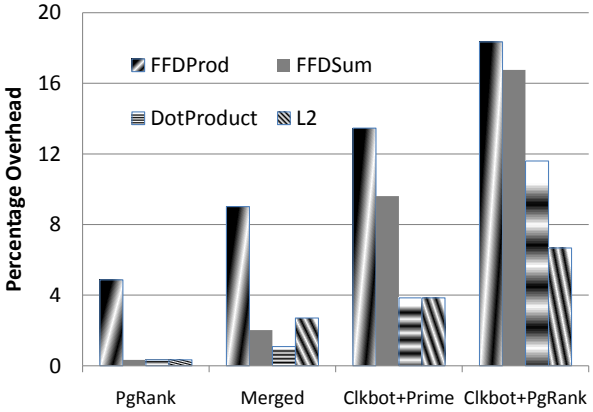

Figure 16: **Empirical results on inputs generated from Dryad data.**

4.3 Implications

We see that in all cases, FFDProd is outperformed by other heuristics. This is due to the fact that dimensions for which demand is much smaller than capacity contribute significantly to the ordering when we take the product. While we would like to ignore such dimensions and sort by decreasing sizes in the relevant dimensions, FFDProd ends up with a very noisy version of this ordering, leading to higher overhead. FFDSum gives higher weight to to the important dimensions and hence is less affected by the irrelevant dimensions[3].

In the case of homogenous VMs, the multi-

[3]In our experiments we tested three different ways to calculate weights. In the first, $w_i = D_i/(nh_i)$, where D_i is the total demand across all VMs for resource i and h_i is the capacity of the host for dimension i. Thus the first weight function is simply the average demand as a fraction of capacity. The second and third approaches aim to make the weight increase more aggressively as the resource becomes scarce. To this end, we set $w_i = \exp(1/D_i)$. In the third experiment we set $w_i = \exp(10/D_i)$

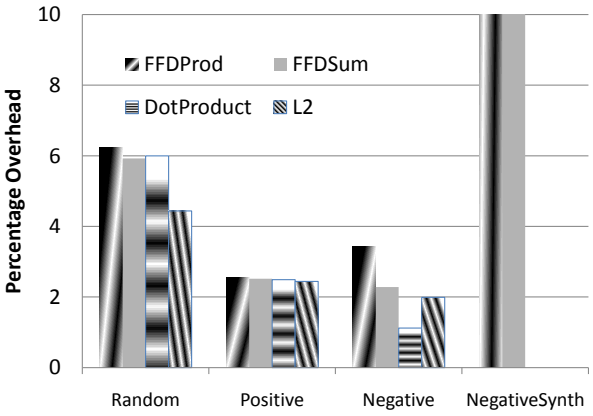

Figure 17: **Empirical results on two dimensional synthetic inputs. The overhead for NegativeSynth for the two FFD-based algorithms is 60%.**

dimensional heuristics do not necessarily improve on FFDSum. In the first case, this is due to the fact that the various VMs are very similar to each other, and have demand in one dimension much larger than the others. Hence the input is effectively one-dimensional. When we have a mix of jobs with different dominant dimensions, the dimension-aware heuristics outperform FFDSum. While we do not know OPT in these cases, it is noteworthy that the better heuristics come very close to the trivial lower bound; their performance relative to OPT is likely to be even better.

The results for the correlated input cases (Positive and Negative) are interesting. In the positive case, all algorithms do equally well and have a less than 3% overhead. This is not surprising since this class is effectively one-dimensional, which makes algorithms the same. In the negatively correlated case, DotProduct and L2 all do better than FFDSum. While we have presented results for three random 2-dimensional input classes, the results are similar for other random input classes studied in literature, and for higher dimensional inputs. The fourth input class NegativeSynth above is designed to show the starkest contrast between the dimension aware and the single-dimensional algorithms and shows a 60 percent difference. With higher dimensional data, this worst-case difference can be close to d.

In summary, the experiments suggest that FFDProd is dominated by FFDSum, which performs reasonably well on some classes of inputs. The dimension-aware heuristics can give up to 10% improvement over FFDSum on realistic workloads. The experiments suggest that when there is more than one dominant dimension, and there is mix of VMs with different dominant resources, the dimension-aware heuristics show the largest gain.

5 Discussions and Conclusions

This paper addressed two inter-related questions that are critical to the design and use of VM consolidation heuristics. The first question pertains to how resource utilization and performance aggregate when VMs are co-hosted, trying to identify bottlenecks that might have adverse impacts on consolidation. The second question pertains to how resource demands and scarcities that span across different dimensions—such as computation, network, and storage—should be treated with reference to VM consolidation.

Through a combination of experiments and empirical analysis, we draw the following insights on these two questions. First, performance (and resource) aggregation depends on the type of resource and the quality of the workload hosted on the VMs. For computation and network resources, performance and resource utilization stay close to their estimates while performance degradation in the presence of high resource contention is gradual and fair. For cache and storage, however, consolidation of cache-sensitive and storage-intensive VMs is likely to lead to severely degraded performance. Second, it appears that heuristics that have simplistic methods of treating multi-dimensional resource requirements are reasonably effective in many common practical situations. Yet, more sophisticated, dimension-aware heuristics provide additional benefits, especially for a mixed workloads with negatively correlated dependence on resources.

These broad conclusions are evident even from the limited experiments and empirical analysis presented in this paper. Further experimentation would add more details to these insights. 1. The experiments could be repeated on different types of hardware and hypervisor platforms with more intermediate points of resource utilization. That would generate more detailed performance models, which can be used to design better heuristics. 2. More complex benchmarks that mix different resource types and run on multiple hosts would add further insights into consolidation of distributed workloads, throwing light on issues such as data locality and network synchronization. 3. Time-based models that capture the variation of resource demands with time is another important direction to experiment with.

Overall, the insights presented here provide useful guidance to both the customers of VM management tools as well as the designers of new consolidation heuristics. These observations validate the following effective and easy-to-deploy approach to VM consolidation: Employ a simple FFD-based heuristic that rely on just few critical resources, whose demands

can be easily estimated. Pin each VM to a specific core and a separate disk to minimize the interference of one VM on the other and achieve better performance isolation. For the designers of more sophisticated heuristics and customers seeking more aggressive benefits from VM consolidation, it points out additional requirements to be met—namely, a more precise model of performance aggregation specific to each resource type and workload type and more accurate estimates of the resource demands and quality of workload running on each VM.

Finally, this paper also identifies fresh opportunities for engineers to improve current virtualization platforms. For instance, it highlights the need for more cache-aware resource allocation in both the virtualization platform as well as by VM consolidation tools. Another great improvement needed is better I/O performance isolation, especially for disk I/O. The recent work called mClock [6] on handling I/O throughput variability in hypervisors is a step in the right direction. Third, virtualization platforms need to accommodate the increasing heterogeneity in the number of processors, the larger number of cores per processor, and more complex cache models in evolving processor architectures such as the Intel Single-chip Cloud Computer [17]. Overall, we believe this paper identifies pitfalls and opportunities in virtual machines consolidation and serves as a good starting point for future research in this area.

References

[1] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231–262, 2001.

[2] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, Oct. 2001.

[3] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. In *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco CA, Apr. 2008.

[4] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proc. of the ACM SIGMETRICS Conference*, Banff, Canada, June 2005.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, pages 46–93, Boston, MA, USA, 1997. PWS Publishing Co.

[6] A. Gulati, A. Merchant, and P. J. Warman. mClock: Handling throughput variability for hypervisor IO scheduling. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, Oct. 2010.

[7] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proc. of EuroSys Conference*, Lisbon, Portugal, Mar. 2007.

[9] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-Clouds: Managing performance interference effects for QoS-aware clouds. In *Proc. of the Eurosys Conference*, Paris, France, Apr. 2010.

[10] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proc. of the Workshop on Compilers and Operating Systems for Low Power (COLP)*, Barcelona, Spain, Sept. 2001.

[11] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *IEEE Computer*, 38(5):39–47, May 2005.

[12] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proc. of the Workshop on Power Aware Computing and Systems (HotPower)*, San Diego, CA, Dec. 2008.

[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proc. of the International Conference on World Wide Web (WWW)*, Banff, Canada, May 2007.

[14] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, Apr. 2007.

[15] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[16] Windows Azure. `http://www.microsoft.com/windowsazure/windowsazure/`.

[17] Intel research :: Single-chip cloud computer. `http://techresearch.intel.com/ProjectDetails.aspx?Id=1`.

[18] Microsoft Systems Center Virtual Machine Manager. `http://www.microsoft.com/systemcenter/virtualmachinemanager`.

[19] VMware DRS - dynamic scheduling of system resources. `http://www.vmware.com/products/vi/vc/drs.html`.

[20] Citrix XenServer. `http://www.citrix.com/xenserver`.