# Runtime Demand Estimation for Effective Dynamic Resource Management

Canturk Isci, James E. Hanson, Ian Whalley, Malgorzata Steinder and Jeffrey O. Kephart
IBM Thomas J. Watson Research Center, Hawthorne, NY 10532
{canturk,jehanson,inw,steinder,kephart}@us.ibm.com

## Abstract

*Systems management techniques that allocate resources to running entities, such as processes and virtual machines (VMs), often require estimates of the resources required by each of these resource consumers. For example, many proposed virtual machine placement algorithms attempt to allocate VMs to physical hosts in such a way as to minimize the number of physical hosts that are occupied, while ensuring that each VM receives the CPU required to do its task adequately. The common practice is to assume that the CPU requirement is equal to the current CPU utilization, or to use a prediction of it over an appropriate time horizon.*

*In this paper, we demonstrate that, when multiple VMs or processes co-reside on a physical host, the measured CPU utilization may provide a poor estimate of the actual requirement. We derive a simple, much more accurate alternative estimate of CPU demand, implement it, and demonstrate its superiority experimentally. Furthermore, we demonstrate that using our demand estimation framework in conjunction with dynamic resource allocation in a virtualized environment greatly improves the effectiveness of dynamic placement, resulting in one-shot convergence to optimal placement and significant improvements in the overall performance of the individual VMs.*

## I. Introduction

In order to function well, systems management algorithms that dynamically control the placement of applications on physical hosts must be able to estimate accurately the resource requirements of the individual applications. For example, consider a virtualization manager that is responsible for migrating virtual machines (VMs) from one physical host to another as workloads fluctuate, priorities change, or VMs enter or leave the system. In order to make well-informed placement decisions, the virtualization manager requires a good estimate of the resource needs of each VM. Similarly, an application placement controller [1] that is responsible for deciding how many instances of each application to run, and where to run them, must know how much resource each application is likely to consume on any prospective target host.

At present, there are two main approaches to estimating the resource requirements of VMs and other applications, both of which are unsatisfactory. In the first approach, used by several prior studies [2], [3], [4], [5], the application's current *measured* resource usage is taken as an estimate of its *required* resource usage. However, this estimate is valid only for applications that do not share physical resources—an assumption that is violated to an ever-increasing degree in today's environments. In today's infrastructures, dozens of applications and VMs may be packed onto the same host. As we demonstrate in this paper, the resource requirements estimated by this method can be significantly less than the true value, leading to strongly suboptimal allocations and unnecessarily long periods of adjustment. With this approach, the algorithm repeatedly discovers that its expectations have not been met, forcing it to perform multiple allocations. The second approach entails legacy implementations tightly coupled with the underlying virtualization technology for specific implementations [6], preventing them from being generally applicable.

The purpose of this paper is two-fold. First, we introduce a new resource demand estimation methodology that is accurate, robust, lightweight and general. We demonstrate its superiority to utility-based demand estimation through a broad set of experiments conducted with a complete real-system prototype implementation. Second, through experiments on synthetic and real data center workloads, we demonstrate that our demand estimation technique can be used to improve the efficiency of dynamic resource allocation substantially.

Our demand estimation approach is general enough to be employed in a wide variety of platforms, environments and scenarios involving dynamic allocation of applications or VMs to physical hosts. While we confine the experiments reported in this paper to a specific virtualization technology, our solution can be retrofitted into a variety of existing virtualization managers, and could be employed by other application managers such as WebSphere Extended Deployment [1].

The rest of this paper is organized as follows. In Section 2, we motivate the need for an improved CPU demand estimation approach. In Section 3, we describe and derive our CPU demand estimation method. After discussing our prototype system implementation in Section 4, we present in Section 5 results that demonstrate *(i)* the superior accuracy of our estimator, *(ii)* substantial improvements in dynamic VM placement decisions with the help of our estimator, and *(iii)* significant performance benefits resulting from the improved placement decisions. We discuss related work in Section 6, and summarize our conclusions in Section 7.

## II. Motivational Example

The advantages of our work, and the motivation for it, are best demonstrated with a simple example, shown in Figure 1. Consider a virtualized cluster of three hosts, each with a total CPU capacity of 100%, and eight VMs, each of which can demand up to 100% of a host's capacity. A dynamic resource manager is responsible for allocating virtual machines to physical hosts. Its goal is to ensure that the number of hosts

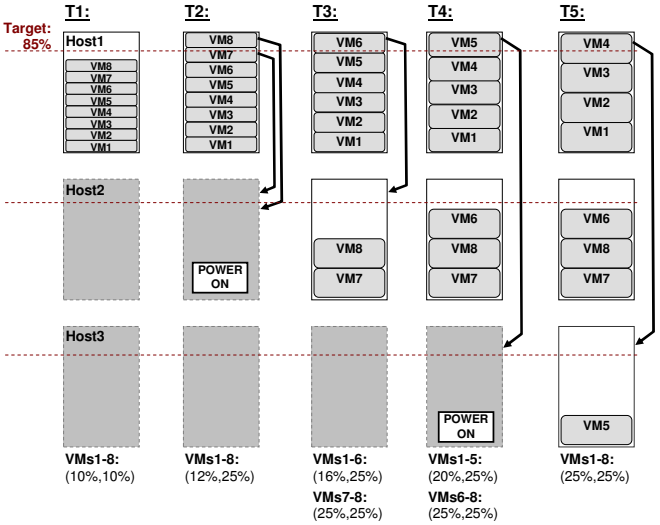



**Fig. 1. Dynamic resource balancing with usage-based demand estimation. The bottom values show (<Usage>%,<demand>%) for each VM.**

that are powered on is minimized, while ensuring that each host's utilization does not exceed 85% and each VM receives the CPU resource that it requires. To accomplish this task, the resource manager is permitted to exercise three controls: migrate a VM, power on a host, and power off a host.

Initially, at step $T1$, all eight VMs are close to idle, each demanding 10% of a host's CPU. Here, all VMs fit within a single host ($8 \cdot 10\% = 80\% < 85\%$), and so the dynamic resource manager has powered down two of the three hosts.

Next, at step $T2$, each VM's demand increases to 25%. Clearly, the best solution is to power on two more hosts, and migrate VMs such that no more than three occupy any host. However, observe what happens if the dynamic resource manager bases its estimate of CPU demand on the observed utilization. At step $T2$, the physical host is overloaded, with a CPU utilization approaching 100%. Due to competition, each VM consumes about 12% of Host 1's capacity. Since the resource manager believes that each VM requires just 12% of a host's capacity, the best solution appears to be to power on one host and migrate two VMs to it, leaving six VMs on the first host; this would appear to result in a resource usage on the first host that is less than 85%.

Once the migration is performed, it comes to light that removing two VMs from Host 1 is *not* sufficient after all. Each of the remaining six VMs expands to about 16% of Host 1's capacity. Since Host 1 continues to be overloaded, one more VM is migrated to Host 2 at step $T3$, with the expectation that such a move will reduce Host 1's utilization to about 80% without raising Host 2's above 85%.

At step $T4$, five VMs remain on Host 1. Each has expanded its CPU usage to 20%, so Host 1 is still overloaded. This forces the resource manager to perform one more migration, which in turn requires it to power on Host 3, as Host 2 cannot contain a fourth VM without exceeding 85% usage. At this point, the VMs finally receive the CPU capacity that they require. However, since Host 1 is violating the constraint that its CPU utilization not exceed 85% usage, a final migration of a VM to Host 3 is performed at step $T5$.

The example of Figure 1 illustrates two related problems that are addressed by this paper. First, estimates of resources required by a VM (or, more generally, any process) are consistently too low if they are based solely upon the observed utilization. Second, these underestimates cause dynamic resource managers that employ them to converge too slowly to an efficient VM placement (or, more generally, an efficient allocation of resources). In this example, the dynamic resource manager inched towards an efficient allocation in which eventually all eight VMs obtained the 25% CPU that they required in a sequence of four potentially time-consuming steps. The demand estimation technique introduced in this paper allows this good state to be reached in a single step.

## III. VM Demand Estimation Framework

One of the core ideas of our work is to track easily-observed hypervisor scheduling metrics to derive reliable estimates for actual VM resource demand. The fundamental set of metrics used is listed in Table I. These metrics can be defined for each VM, as well as each physical processor. Our demand estimation method tracks these metrics over a *sampling period*, $T$, which is smaller than the average decision making period used by a dynamic resource manager.

| | |
|---|---|
| **CPU Used** | Time spent while using the CPU |
| **CPU Wait** | Time spent waiting for some resource other than the CPU |
| **CPU Ready** | Time spent waiting for CPU to be available |
| **CPU System** | Time spent in the hypervisor/kernel |

**TABLE I. Hypervisor scheduling metrics used in demand estimation.**

The *CPU Used* metric tracks the amount of time a VM was actually running on the physical hardware. *CPU Wait* describes the time spent waiting on some other resource such as disk or network I/O. During wait time, the VM cannot progress even if there is available CPU. *CPU Ready* captures the time when the VM was ready to run on the CPU, but had to relinquish the processor to some other VM or task competing for the same resource. During ready time, the VM could actually make useful progress had it been given more processing resources. Finally, *CPU System* represents the time spent in lower-level hypervisor or kernel tasks. Each VM's lifetime is distributed among the four states *CPU Used*, *CPU Wait*, *CPU Ready*, and *CPU System* depending on *(i)* resources available to the VM, and *(ii)* the VM's actual desired demand. Therefore, these four metrics are the fundamental components for accurate CPU accounting.

In general, the sum of all four metrics should approximate the total elapsed time for a VM. For an idle VM, *Used* and *Ready* times are close to 0%, while *Wait* is around 100% of the total elapsed time for the VM. For a completely CPU-bound VM, the sum of *Used* and *Ready* is expected to come close to 100% of the total elapsed time for the VM.

### A. Intuitive Derivation

Figure 2 shows a hypothetical example that illustrates how a VM's time is distributed among the four states. System time is omitted from the description for clarity. We consider 8 VMs executing on a single host, which has 2 CPUs for a total CPU capacity of 200%. Each VM has a CPU demand of 80%. In

U: Used | R: Ready | W: Wait

| t: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VM1: | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R | R | R |
| VM2: | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R | R | R |
| VM3: | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R | R |
| VM4: | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R | R |
| VM5: | R | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R |
| VM6: | R | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U | R |
| VM7: | R | R | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U |
| VM8: | R | R | R | U | R | R | R | U | R | R | R | U | R | R | R | U | W | R | R | U |

**Fig. 2. Example timeline of 8 VMs showing the distribution of time into CPU accounting states.**

the example, $t$ represents the time quantum during which only two VMs can run—one for each of the two cores. For simplicity, we assume the VMs are scheduled in a pairwise round-robin fashion. That is, at $t = 1$, VM1 and VM2 run, while the remaining six VMs wait their turn; at $t = 2$, VM3 and VM4 acquire the cores, while the others wait; and so on. In addition, the VM load is constructed such that each VM performs CPU operations worth $4t$ time, followed by a $1t$ idle period. This load pattern, with its period of $5t$, is repeated to achieve the 80% CPU demand over a sampling period $T \gg 5t$. The three different states are represented by $U$ (*CPU Used*), $W$ (*CPU Wait*) and $R$ (*CPU Ready*). The columns of the figure represent the time slices of length $t$.

At $t = 1$, all VMs are ready to run. The first two VMs are scheduled on the CPUs, and are therefore in the $U$ state. The other VMs stay in $R$ state, as they are ready to run, but no CPUs are available. At $t = 2$, VM1 and VM2 relinquish the CPUs (and go into the $R$ state), and the CPUs are allocated to VM3 and VM4. This pattern continues until $t = 13$. At this point, VM1 and VM2 run for the fourth $t$ interval. After this, both VM1 and VM2 have each received $4t$ CPU time, and so have completed their CPU-bound operations. Therefore, at $t = 14$, they enter the $W$ state, and remain idle for $1t$. VM3 and VM4 acquire the CPUs to perform their fourth $t$ interval of CPU-bound execution and then (at $t = 15$) go into the $W$ state. This continues for every pair of VMs. After each VM has idled in the $W$ state for $1t$, it starts competing for resources again. As the highlighted region shows, the overall execution pattern has a period of $16t$. Therefore, a sampling interval of $T \gg 16t$ will capture a similar distribution of states to that shown in the highlighted $16t$ region.

Focusing, therefore, on the representative $16t$ region, we see that each VM shows a state distribution of $\{U = 4t, W = 1t, R = 11t\}$. While the amount spent in $R$ is generally dependent on the amount of consolidation and competition between VMs, $W$ and $U$ can be expected to be moderately consistent regardless of the level of overcommitment. From these observations, we can state that the demand of a VM is captured by the ratio of how much time it spends in the $U$ (Used CPU) state and the $W$ (Wait CPU) state. We see that the ratio of the *Used CPU* time ($4t$) to the sum of *CPU Used* and *CPU Wait* ($5t$) represents the actual CPU demand for each of the VMs (80%). Based on this, the estimated actual demand for a VM, $i$, can be represented as shown in Equation 1, where $T$ represents the sampling period for the tracked VM metrics.

$$CPU\ Demand_i = \frac{Used_i}{Used_i + Wait_i} = \frac{Used_i}{T - Ready_i} \tag{1}$$

## B. Analytical Derivation

The result shown in Equation 1 can also be derived analytically. At each time quantum $t$, a VM can be in one of three possible states: *(i)* $u$ (*Used*) = VM "gets" the CPU; *(ii)* $w$ (*Wait*) = VM is waiting on another resource; *(iii)* $r$ (*Ready*) = VM is waiting on the CPU resource.

Let $T_u$, $T_w$, and $T_r$ be the accrued counts for each of the states over the sampling interval of $T$. Assume that the VM's behavior is statistically stationary, such that for $T \gg t$ the following ratios defined in Equation 2 are time-independent and well-defined. Moreover, as the VM is only in one of these states during any quantum, $U + W + R = 1$.

$$U = \frac{T_u}{T},\ W = \frac{T_w}{T},\ R = \frac{T_r}{T} \tag{2}$$

***No Resource Contention Case:*** When there is no resource contention, the VM receives all of the CPU it requires, so $R = 0$. Since the VM's demand is identical to its usage, demand estimation reduces to the trivial case, $CPU\ Demand = U_0 = U$. The goal of our approach is to estimate $U_0$ given values of $U$, $W$, and $R$ when there *is* CPU contention.

***Resource Contention Case:*** Here we treat the CPU scheduling algorithm as a random process in which, at each time quantum, the CPU is "given" to the VM with probability $1 - p_{busy}$, and to something *other* than the VM with probability $p_{busy}$. In this case, where there is contention for resources, we have $T_r > 0$ and $R > 0$ for the contending VMs.

Further, we assume that the sequence of VM states $s_0 s_1 s_2 \ldots, s_n \in \{u, w, r\}$ in this case differs from that in the no-contention case *solely* in the rate at which the VMs progress over time. If, at a given time, the VM would have been in state $u$, but the CPU is given to another VM, then the VM goes into state $r$ and its program counter does not advance. Alternatively, if the VM is in state $w$ when the CPU is given to something else, the VM is unaffected.

When there is no contention, the sequence of VM states consists solely of states $u$ and $w$, and has the length of $T = T_u + T_w$. When there is contention, this sequence is transformed by replacing each $u$ state with a sequence of $r$*$u$, where $r$* denotes a sequence of 0 or more $r$ states. The length of the $r$* sequence is geometrically distributed with a probability of success of $1 - p_{busy}$, and its expected value is $p_{busy}/(1 - p_{busy})$. The sequence thus becomes longer and lasts $T'$.

When this transformed sequence is measured over interval $T$, we obtain the following set of CPU accounting relations.

$$T'_u = T_u \frac{T}{T'},\ T'_w = T_w \frac{T}{T'},\ T'_r = T'_u \frac{p_{busy}}{1 - p_{busy}},\ T = T'_u + T'_w + T'_r \tag{3}$$

Now, starting with measured values $U$, $W$, $R$, we have the following equations.

$$U = \frac{T'_u}{T} = \frac{T_u}{T'} = U_0 \frac{T}{T'} \tag{4}$$

$$W = \frac{T'_w}{T} = \frac{T_w}{T'} = (1 - U_0) \frac{T}{T'} \tag{5}$$

$$R = \frac{T'_r}{T} = \frac{T - T'_u - T'_w}{T} = 1 - \frac{T'_u + T'_w}{T} = 1 - \frac{T}{T'} \tag{6}$$

Solving Equations 4 and 5 for $U_0$ produces Equation 7, while solving Equations 4 and 6 leads to Equation 8.

$$CPU\ Demand = U_0 = \frac{U}{U+W} \tag{7}$$

$$CPU\ Demand = U_0 = \frac{U}{1-R} \tag{8}$$

## IV. Real-System Implementation: Experimental Measurement and Evaluation Platform

We have developed and validated a fully-functional prototype for our demand estimation framework and for our implementation of demand-estimation-based dynamic resource management. Figure 3 shows a simplified architectural view of this prototype. Its main components are:

**(i)** The virtualized environment including the hosts, VMs, and the management endpoint.

**(ii)** The *Load Driver*, which manages the load on the VMs and collects application-level performance data.

**(iii)** The *Data Collection and Performance Evaluation* component, which gathers the application-level performance data from the Load Driver and visualizes each VM's progress at runtime for our quantitative evaluations.

**(iv)** The *Demand Estimation and Dynamic Resource Management* layer, which performs on-the-fly monitoring of hypervisor-level performance statistics, demand estimation and dynamic resource management.

The virtualized environment we use to validate our prototype is based on VMware VirtualCenter v2.5 and ESX v3.5. The virtual machines run common Linux distributions.

The *Load Driver* is a distributed component that helps manage the desired amount of load within each VM. Each VM includes a *Load Driver Client* that communicates with the central *Load Driver Server*. The server uses separate trace definitions for each VM that specify the time-varying nature of the load in terms of memory usage and CPU utilization. The server updates the clients when their load level changes and the clients change the intensity of the load they are running accordingly. The clients also update the server each time they complete a fixed unit of work. The server can then determine the time it took to complete each unit of work for each VM. These durations, referred to as "*service times*", represent the current *application-level* performance of each VM. An increase in the service time of a VM, therefore, indicates that the VM is suffering from performance degradation.

The *data collection and performance evaluation* component is the processing backend of the Load Driver. It tracks application-level performance for each VM and provides a visualization of each VM's performance using the LiveGraph real-time data visualization, analysis, and logging framework.

The *Demand Estimation and Dynamic Resource Management* layer, implemented as a *virtual appliance*, is the main component of our prototype. It manages the cluster of hosts and VMs, tracks the inventory, determines cluster-level and host-level capacities and demands, makes placement decisions, and applies the management actions across the cluster. These functions are carried out by three main subcomponents: *(i)* the *Performance Monitor*, which acts as an API client

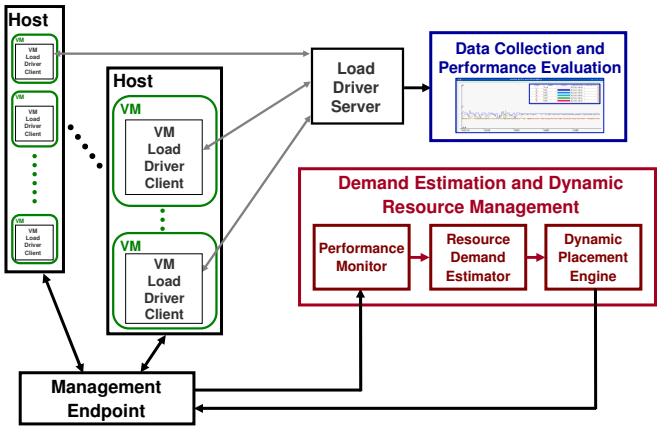



**Fig. 3. Developed prototype implementation and evaluation framework.**

for the hosts and the management endpoint and collects the host and VM performance metrics that are required by our demand estimation technique; *(ii)* the *Demand Estimator*, which uses information from the Performance Monitor and applies our demand estimation methodology for each VM; and *(iii)* the *Dynamic Placement Engine*, which computes and enacts resource allocation actions to satisfy the estimated VM demand requirements, and host and cluster capacity constraints. The Dynamic Placement Engine uses simple, standard heuristics to compute VM placements that operate in a similar fashion as in prior work [4] for our experiments.

It is worth noting that our overall demand-estimation-based dynamic resource management framework is a fully black-box approach: both demand estimation and dynamic resource management operate without looking inside the VMs. All monitoring is done from the view of the hypervisors or the management endpoint. Demand estimation is restricted to using hypervisor-specific characteristics. Dynamic placement relies only on capacity measures, utilization information, and the demand estimates.

The Load Driver, and the Data Collection and Performance Evaluation components in our prototype system are merely used to provide direct quantitative evaluations. The Load Driver Client within each VM is also used to only generate the load on the VM and to provide a detailed, "application-level" performance characterization for purposes of experimental evaluation.

## V. Experimental Results

Here, we first show the accuracy of our demand estimation method in comparison to utilization-based proxies for resource demand. Then the subsequent subsections describe experimental results that illustrate the favorable impact of our demand estimation upon dynamic VM placement.

### A. Evaluation of Demand Estimation Accuracy

Here we present several experiments that assess the accuracy of our CPU demand estimator in comparison with the traditional utilization-based estimator. In the first experiment, we create 5 VMs and use the load driver described in Section IV to set their CPU demands to the levels shown in Table II.

Initially, all VMs are placed on a single host with two physical CPUs (equivalent to 200% CPU capacity), and they compete for system resources with the same priority. Since the total CPU demand for the 5 VMs is 300%, the host is overcommitted with an *overcommit ratio* (defined as the ratio of total CPU demand to host capacity) of 1.5*X*.

| | CPU Demand |
|---|---|
| **VM1** | 100% |
| **VM2** | 80% |
| **VM3** | 60% |
| **VM4** | 40% |
| **VM5** | 20% |

**TABLE II. CPU demand configurations for the five VMs used in demand estimator validation.**

Figure 4 shows the actual VM demand, estimates produced by our estimator, and estimates produced by a utilization-based estimator. The results are averaged over 7 runs, and the error bars show $\mp 1$ *standard deviation*.

Overall, our demand estimator is substantially more accurate and reliable than the utilization-based estimator. Across all experiments, our estimator achieves an RMS error of only 6.6%, while the same error for the utilization-based estimation is 32.6%. The fluctuations across runs are small and comparable between the two cases, averaging 2% across all VM configurations, and never exceeding 4%.

***Demand Estimation with Different Overcommit Ratios***

In a second experiment, we explore how the accuracy of our demand estimation varies with the overcommit ratio for a fixed number of VMs. We create 5 VMs with equal CPU demands ranging from 20% to 100% in steps of 20%, resulting in overcommit ratios ranging from 0.5*X* to 2.5*X*.

Averages over four experimental runs are depicted in Figure 5. For utilization-based estimation, the estimates are always lower than the actual demand, and both the absolute and the relative error grow consistently as the level of overcommitment increases, from 3.7% absolute error at 0.5X to 61.5% at 2.5X, with an average of 29%. In contrast, the errors for our demand estimation technique are much smaller and do not grow substantially with increasing overcommit ratio. Here, the errors range from 0.7% to 6.5%, with an average of 3%. These results show that, with higher overcommitment, utilization-based estimates deviate substantially from actual demand, while our demand estimation method follows actual demand closely in all cases.

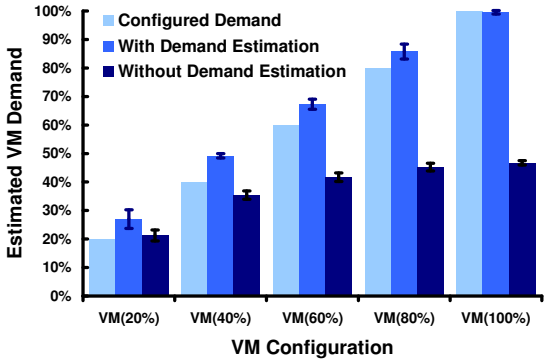



**Fig. 4. Demand estimation results with our prototype implementation.**

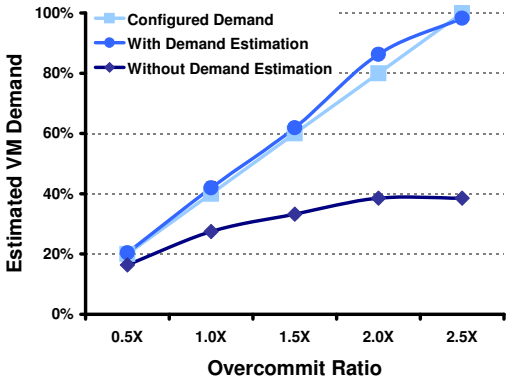



**Fig. 5. Demand estimation accuracy with increasing level of overcommitment.**

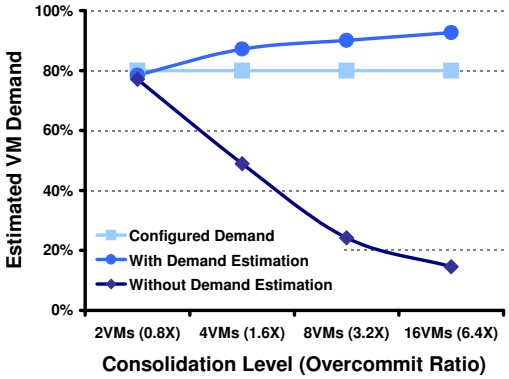



**Fig. 6. Demand estimation accuracy with increasing level of VM consolidation.**

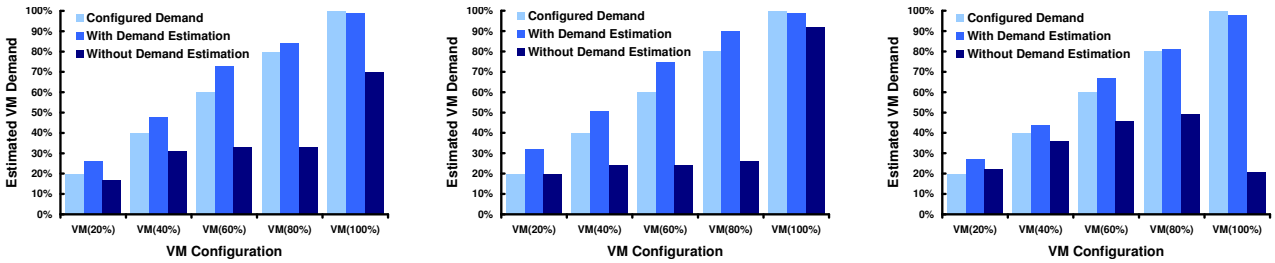
***Demand Estimation with Different Consolidation Levels***

Next, we study how the accuracy of our demand estimation approach is affected by higher levels of VM consolidation. For this evaluation, we increase the number of VMs on a host, while keeping the demand of each VM at 80% CPU in all cases. Starting with 2 VMs, the number of VMs on the host (and hence the consolidation level and overcommit ratio) is doubled up to 16 VMs (and 6.4*X* overcommit ratio). We cannot increase consolidation further due to memory limitations imposed by the VMs. We repeat each configuration four times and show the averages in Figure 6.

Figure 6 shows that, even at the highest levels of consolidation, our demand estimator performs quite well, and significantly better than the utilization-based estimator. While the latter yields errors ranging from 2.9% to 65.5%, with an average of 38.8%, our demand estimation method achieves errors ranging from 1.6% to 12.7%, averaging 7.7% in this experiment. With increasing consolidation, our demand estimation technique tends to *over*estimate demand somewhat, in contrast to the strong *under*estimate of the utilization-based approach. However, even under such stringent constraints, our demand estimator proves to be a very reliable proxy of actual resource demand.

## B. Demand Estimation with Resource Constraints

In practice, there are several management operations that can profoundly affect the resource scheduling and utilization of VMs, including setting *(i)* relative priorities (via shares); *(ii)* resource caps (via limits); and *(iii)* resource guarantees




(a) Priorities. (b) Resource guarantees. (c) Resource limits.

**Fig. 7. Prediction results with different VM priorities, resource guarantees and resource limits.**

(via reservations). For our demand estimation technique to be generally useful, it must robustly determine the true CPU demand of the VMs under these constraints. To assess our technique's effectiveness under these conditions, we consider three additional scenarios using the set of 5 VMs of Table II.

Figure 7(a) shows the demand estimation results when the priority of VM1 (VM(100%)) is increased by doubling its shares. As can be seen, the CPU usage of VM1 increases dramatically from the default case (Figure 4), while the other VMs experience a decrease. Nonetheless, the estimated demands of VMs still closely track their actual requirements.

Figure 7(b) illustrates the impact of setting the CPU reservation of VM1 to 100% of the host CPU capacity. The VM1 CPU usage follows its demand, while the other VMs have their usage further reduced. Yet our estimation technique continues to closely track the actual demand for all VMs.

Finally, Figure 7(c) shows what happens when resource limits are used to constrict VM1 to around 23% of the physical CPU capacity. VM1's CPU usage is reduced to about $1/5$ of its actual demand, but again demand estimation tracks the actual resource requirements very well.

Overall, even under a range of common resource constraints that affect the scheduling and utilization of the individual VMs, our estimation technique robustly determines the true CPU demand with good accuracy.

## C. Resource Management with Demand Estimation

Having established the superior accuracy and robustness of our demand estimation method, we now return to the scenario described in Figure 1 and demonstrate the favorable impact of our improved estimation upon dynamic VM placement.

Using the Dynamic Placement Engine described in Section IV, we perform two experiments starting with the 5 VMs of Table II placed on a single host and completing once the VM placement stabilizes. In the first, the Dynamic Placement Engine uses utilization-based CPU demand estimates, and in the second it uses the estimates produced by our method. In both cases we use a utilization threshold of 90% as the target maximum utilization for each host.

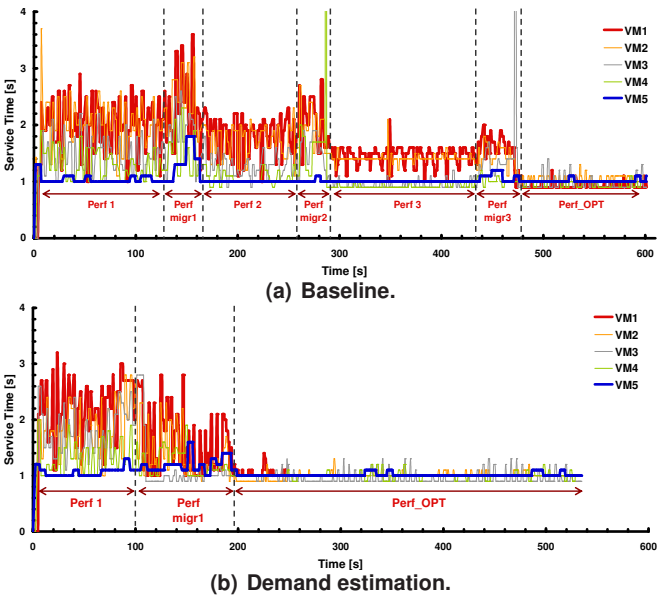
Table III shows the timeline of placement decisions applied in our prototype with and without demand estimation. Without our demand estimation, the placement engine performs as was illustrated earlier in Figure 1, requiring three iterations to reach the desired state—first, VM5 is migrated to the second host, then VM4, and finally VM3. This is because the CPU usage of individual VMs inflate after each step. In contrast, when the Dynamic Placement Engine is provided with our demand estimations, it is able to perform all of the necessary balancing actions in a single iteration.

| Timeline | Without Demand Estimation | With Demand Estimation |
|---|---|---|
| *Iteration 1* | Migrate VM5 → Host2 | Migrate VM5, VM4 & VM3 → Host2 |
| *Iteration 2* | Migrate VM4 → Host2 | ALL DEMANDS MET |
| *Iteration 3* | Migrate VM3 → Host2 | |
| *Iteration 4* | ALL DEMANDS MET | |

**TABLE III. Timeline of dynamic placement actions, with and without demand estimation.**

By reducing the number of iterations required to reach an efficient assignment of VMs to hosts, our demand estimation technique also improves average performance measurably, as illustrated in Figure 8. We define a VM's performance in terms of *service time*—the time required to perform a defined unit of work generated by the load driver. Figure 8 displays the service times for each VM during the experiment both with and without demand estimation. $\text{Perf}_i$ denotes the performance in an intermediate configuration, $\text{Perf}_{\text{migr},i}$ denotes the performance during a particular migration, and $\text{Perf}_{\text{OPT}}$ represents the final optimal performance.

In both cases, all VMs eventually reach the ideal service time of 1s. However, the length of the transient and the VM performance during the transient are radically different. Without demand estimation, the three iterations are clearly discernible, as is the performance impact on the 5 VMs




(a) Baseline.

(b) Demand estimation.

**Fig. 8. Overall execution timelines and corresponding performance for all VMs for baseline and demand-estimate-based cases.**

during the intermediate iterations. Initially, VM1 and VM2 (which demand 100% and 80% CPU respectively) suffer the highest performance degradation, as they remain on a heavily-loaded server until the final iteration, when enough competing VMs are finally migrated to other servers. However, with demand estimation, all three VMs are migrated during the first (and only) migration interval, resulting in better and more consistent VM performance, with a transient phase that lasts less than half as long as was necessary without demand estimation.

The performance gains from demand-estimation-based dynamic placement are also significant in this case. Our approach reduces the performance degradation of VM1 and VM2 by 2$X$, and VM3 and VM4 by 1.5$X$. There is no significant improvement for VM5, which gets close to its 20% CPU entitlement even in the overcommitted case.

### D. Performance Evaluation with Data Center Traces

Thus far, we have demonstrated the accuracy of our demand estimation method and its beneficial effect on VM placement by using a set of VM load scenarios that were designed to explore the parameter space. Here, we use real data center workloads to demonstrate the substantial end-to-end performance benefits of demand-estimation-based resource management.

For these experiments, we obtained load profiles for over fifty VMs in a real data center, and selected from them a representative set of 14 VMs. We used these VMs to create an aggressively consolidated virtualized environment with dynamically varying workload characteristics. Figure 9 shows their individual and aggregate behavior over an 18 hour period.

In our experiments we focus on the 6 hour window highlighted in Figure 9. This region represents a demand burst after a long lower-utilization region. This is a critical operating region for a dynamic resource manager, as the consolidated resources start to exhibit contention. Therefore, an effective dynamic resource allocation scheme has to quickly identify the bottleneck and carry out remedial actions.

We replay this 6-hour scenario in our prototype implementation, where we employ our resource management framework at 5 minute intervals. We perform the same data

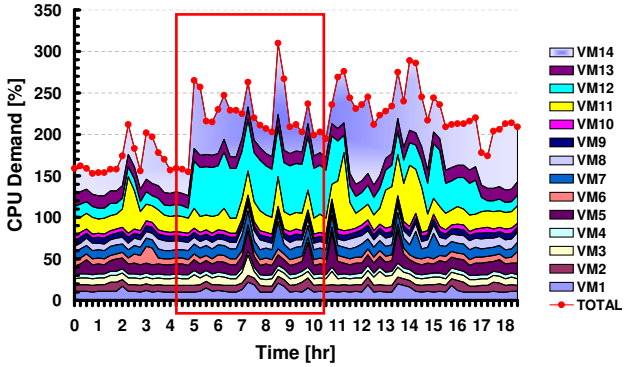



**Fig. 9. Dynamic CPU demand characteristics of selected VMs from actual data center profiles. The highlighted region is the 6 hour window evaluated in our real-system experiments.**

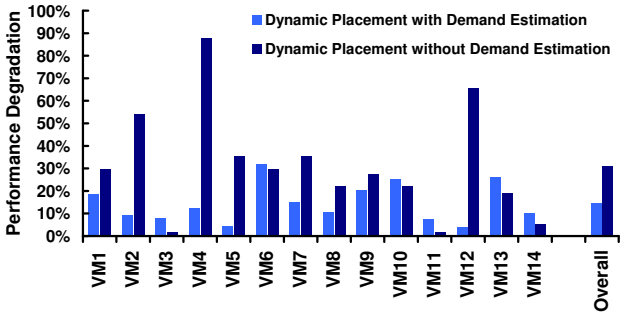



**Fig. 10. Performance degradation of the data center VMs with and without demand-estimation-based dynamic placement.**

center experiments with and without demand estimation to show the comparative benefits of our demand estimation approach.

Figure 10 shows the individual and aggregated VM performance results with and without demand estimation. In the aggregate, demand estimation reduces performance degradation by a factor of about 2$X$, from 30% to 15%. Individually, most but not all VMs experience better performance when demand estimation is employed — for example, VM12's performance degradation is reduced from over 65% to less than 5%. The few VMs that fare better *without* demand estimation do so because they receive a disproportionately large share of resources during poor intermediate allocations.

The results show noticeable performance degradations also with demand estimation. Some level of such performance degradation is expected after the demand burst, as the increased demand causes resource contention until all the corrective actions are applied. This is further emphasized here due to two more reasons. First, the high level of consolidation applied in these experiments leads to additional overheads incurred during hypervisor scheduling. Second, our current placement engine does not take long-term demand characteristics into account, and bases its decisions on the observed VM behavior in its last observation period. Therefore, it cannot proactively identify the optimal allocation for the future dynamic variations of demand with its limited view. Our current work involves integrating demand forecasting into our placement engine to predict future VM load patterns, which can help further improve demand-estimation-based allocation decisions for such time-varying VM behavior.

Overall, the results show the substantial benefits of our demand estimation method. Without demand estimation, the placement engine takes incremental and potentially ineffective steps to resolve resource contention. In contrast, demand estimation helps adapt to changing VM behavior much more quickly, leading to significantly more effective management.

## VI. Related Work

Several previous studies investigate methods that track VM resource consumption for the purpose of managing virtualized systems. Some explicitly use CPU usage as a proxy for demand [7], [8], [9], [4], [10]. Others introduce intervening hooks to the overlaying virtualized applications for resource management and capacity planning [2], [11], [12], [13], [5]. Some of these techniques argue for the

necessity of application-level intervention to distinguish demand from usage [5]. In contrast, we obviate this need by introducing a clear VM-, application- and OS-agnostic technique that determines actual resource demand without disruptive interventions to VMs or applications.

Riel [14] also points out that demand does not equal usage in both native and virtualized systems and discusses some indicators of resource contention, but does not develop from them an estimation technique as presented here. Other works also propose application and VM performance modeling and estimation techniques [15], [16], [17], [18]. These approaches rely on unconstrained environment requirements and track resource usage to develop forecasting or application-level performance modeling techniques. These studies do not provide a generally applicable representation of actual resource demand.

Some prior work also aims to address the problem of identifying the resource requirements of applications and VMs. Cota-Robles et al. [19] inspect application and operating system behavior inside the VM to deduce resource contention. Pacifici et al. [20] also consider a dynamic CPU demand estimation problem for web applications. They use statistical and classification methods to determine the CPU demand for different web request types. Song [21] proposes determining VM CPU load based on its interrupt response time. While these techniques indirectly determine whether a VM or application is receiving its resource requirements, they are either intrusive or rely on application domain knowledge. Moreover, these approaches do not provide a direct measure of a VM's actual demand. In comparison, our technique provides a simple, robust, non-intrusive and commonly-applicable solution to the demand estimation problem.

## VII. Conclusion

This paper introduces a resource demand estimation method for applications and VMs that is accurate, lightweight and robust, and shows that it substantially improves the efficiency of dynamic resource allocation. The method is completely agnostic to specifics of the application and the underlying operating system, and requires no hooks inside the VM or application. Derived from commonly-available system metrics, it is generally applicable on a wide range of platforms.

Evaluations with a real-system prototype show that our demand estimator is significantly more accurate in tracking actual demand than the commonly-used utilization-based proxies, with a 5$X$ reduction in estimation errors from 32.6% to 6.6%. Moreover, while usage-based models deviate further from reality as overcommitment and consolidation increase, our approach consistently tracks actual demand, reducing estimation errors by up to 9$X$ in some circumstances.

Further experiments demonstrate that our CPU demand estimates can substantially improve dynamic resource allocation in virtualized environments. Compared to a utilization-based implementation, our demand-estimation-based dynamic placement framework reduced performance overheads by an average of 2$X$ on real data center workloads, and by as much as an order of magnitude on specific VMs. By accurately estimating the actual requirements of VMs even when there is strong resource contention, our approach transforms an iterative resource allocation process into a one-shot decision, leading to faster convergence to an optimal allocation.

## References


[1] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th international conference on World Wide Web*, 2007.

[2] Y. Ajiro, "Migration planning system and server migration planning method," Patent published, no: 20080209043, 2008.

[3] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *Proceedings of the 15th Int'l World Wide Web Conference*, 2006.

[4] M. Steinder, I. Whalley, J. Hanson, and J. Kephart, "Coordinated management of power usage and runtime performance," in *Proceedings of the Network Operations and Management Symposium (NOMS)*, 2008.

[5] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the USENIX NSDI*, 2007.

[6] VMware Inc., "Resource Management with VMware DRS," VMware Inc., Whitepaper, 2006.

[7] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments," in *Proceedings of the Network Operations and Management Symposium (NOMS)*, 2006.

[8] A. Kochut and K. Beaty, "On Strategies for Dynamic Resource Management in Virtualized Server Environments," in *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer Systems*, 2007.

[9] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure," in *Proceedings of the International Conference on Autonomic Computing*, 2006.

[10] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari, "Server Workload Analysis for Power Minimization using Consolidation," in *Proceedings of the Usenix Annual Technical Conference*, 2009.

[11] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and Performance Management of Virtualized Computing Environments Via Lookahead Control," in *Proceedings of the International Conference on Autonomic Computing*, 2008.

[12] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. M. Chess, "Server Virtualization in Autonomic Management of Heterogeneous Workloads," in *Proceedings of the International Symposium on Integrated Network Management*, 2007.

[13] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Placement of Applications in Virtualized Systems," in *Proceedings of the ACM Middleware Conference*, 2008.

[14] R. V. Riel, "Resource Demand on Linux, Resource allocation, Goldilocks style," in *Proceedings of the Linux Symposium*, 2006.

[15] K. Beaty, N. Bobroff, and A. Kochut, "Dynamic Placement of Virtual Machines for Managing Violations of Service Level Agreements (SLAs)," Patent published, no: 20080295096, 2008.

[16] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. R. Santos, "Performance Models for Virtualized Applications," *Lecture Notes in Computer Science*, vol. 4331, pp. 427–439, 2006.

[17] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[18] C. Tsai, K. G. Shin, J. Reumann, and S. Singhal, "Online Web Cluster Capacity Estimation and Its Application to Energy Conservation," *IEEE Transactions on Parallel Distributed Systems*, vol. 18, no. 7, pp. 932–945, 2007.

[19] E. Cota-Robles and K. Flautner, "Real-time scheduling of virtual machines," Patent issued, no: 7356817, 2008.

[20] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Dynamic Estimation of CPU Demand of Web Traffic," in *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, 2006.

[21] W. Song, "Method and device for scheduling true cpu resources for a virtual machine system," Patent published, no: 20090031304, 2009.