Resource Contention-Aware Virtual Machine Management for Enterprise Applications

Andrew Fox, Andrew Turner, Hyong S. Kim
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA, USA
{afox1, andrewtu, kim}@ece.cmu.edu

Abstract—Consolidating Virtual Machines (VMs) in data centers is desirable as it reduces hardware and power costs. However, the performances of VMs on shared physical servers are not isolated from each other as they contend for the same server resources. This contention degrades the performance of delay sensitive applications and can increase response times by three orders of magnitude at high contention levels. In order to achieve Service Level Agreements (SLAs) under VM consolidation, resources must be allocated by considering the performance effects of contention. We therefore present VARACO, a contention-aware VM management system to achieve Quality-of-Service (QoS) targets for multi-tier web applications. VARACO models applications’ performances online using 13 server resource utilization and contention metrics. Resources are dynamically allocated using these models to achieve QoS targets. Our results show that application-level performance can be modeled 130% more accurately when resource contention is considered. We demonstrate VARACO by achieving a 90th percentile response time target of a sample application under VM consolidation.

Keywords—resource contention; automated resource control;

I. INTRODUCTION

Modern web applications are hosted in Virtual Machines (VMs) in data centers and Cloud computing environments. The applications are divided by functionality into tiers (proxy, web, database, image, etc.) and placed on separate VMs. Traditionally, to ensure satisfactory performance VMs are over-provisioned resources for peak periods, resulting in underutilized servers with resource utilization levels of only 5%-20% [1]. Data center administrators aim to consolidate VMs onto shared servers, thereby increasing utilization levels, which leads to reductions hardware and power costs. However co-locating VMs forces them to share and contend for computing resources as each VM no longer has unrestricted access to the server’s hardware. This resource contention can significantly degrade the performances of VMs and consequently their hosted applications.

Resource contention is especially significant when the CPU uses a work conserving scheduler. In a work conserving scheduler, each VM is allocated CPU shares and is granted access to the CPU in time slices proportionally according to the distribution of shares [2]. However, unlike non-work conserving schedulers where the CPU runs idle if a VM does not use all its shares, unused shares in a work conserving scheduler are released to other VMs that can utilize them. This increases the total throughput which is why work conserving schedulers are recommended for production environments [3]. However, this resource sharing makes it difficult to guarantee that a resource is available on demand. This causes access delays which increase the response times of the applications in VMs. Therefore, achieving Service Level Agreements (SLAs) is increasingly difficult when VMs are consolidated as a VM’s performance is no longer related just to its own resource utilizations, but also to the contending utilizations of co-located VMs. To achieve the best performance it is important that this resource contention is monitored and controlled. We show that by using contention as a control variable that we are able to consolidate VMs and achieve SLAs in situations where uncontrolled contention would have otherwise caused SLA violations.

We present VARACO (Virtual Machine Resource Adaptive Control), a contention-aware data center management system for multi-tier applications. VARACO allows achievement of SLAs in a data center with VM consolidation. VARACO models applications’ performances as a function of 13 server metrics, including the applications’ resource utilization levels and the contention observed due to resource sharing. The models are used to dynamically calculate VMs’ resource requirements to achieve applications’ SLAs. Modeling occurs online as changes in hardware, application architecture, or client behavior may affect applications’ resource requirements. The modeling algorithms are recursive, which is used to provide low computational and storage overhead.

The remainder of this paper is organized as follows. Section II discusses related works. Section III details the VARACO architecture. Section IV provides the mathematical details of the system. Section V discusses the experimental results. Conclusions are presented in Section VI.

II. RELATED WORK

The benefits of increasing server utilization through consolidation are shown in [4]. Statistical multiplexing is used to demonstrate that a 5% overbooking of resources beyond peak load can yield a 500% increase in utility. However, statistical multiplexing does not help guarantee application-level response time targets.
Work-conserving schedulers have been shown to decrease CPU violations by 1 to 3 orders of magnitude at high consolidation levels [3]. A work-conserving scheduler is used in [5] to create a model of application response times based on historical data of assigned CPU shares. However, their approach does not address the complexity of their NP-hard optimization or the storage requirements and validity of historical data.

Performance interference effects from resource sharing has been analyzed in [6] where they compare how different benchmark applications interfere according to different hardware resources. They attempt to cluster applications by types of interference effects. In [7] they attempt to allocate CPU so that VM performance is identical to when it is run in isolation, however it is done with a non-work conserving scheduler.

Static and adaptive controllers have been developed for regulating applications’ response times by setting VMs’ resource allocations. An adaptive autoregressive MIMO controller is designed in [8] using linear functions of CPU and disk utilization. A tuned PID controller is used in [1] to control both CPU and memory allocations. Queuing models, feed-forward predication, and feedback reactive control are used in [9] to dynamically allocate resources to VMs. However, these works do not consider the impact of resource contention as they assume non-work conserving schedulers and performance isolation between VMs. This causes underutilized VMs to waste resources.

Works such as [10] and [11] focus only on modeling application response times and do not consider resource control. In [10] application response time is determined offline as a model based on the application’s different request types. Also offline and using linear regression, [11] uses 11 different server resources to create a model of applications’ response times.

### III. VARACO Overview

VARACO is divided into two components, a Performance Modeler and a Resource Controller, connected by the architecture in Fig. 1. The Performance Modeler models applications’ response times online as functions of server and VM resource utilizations. The Resource Controller is divided into two subcomponents, the Application Controller and the Server Controller. The Application Controller uses the Performance Models to determine the resource requirements of each application, and the Server Controller then allocates resources based on availability.

VARACO models applications as black boxes since obtaining detailed application knowledge may not be feasible. The models are determined online to adapt to the most recent system behavior. Table I shows the 13 resources used in the VARACO model. Note that CPU contention is included as an independent metric which we define as the summation of the utilization of all other VMs on the server. In doing so, contention is modeled and allocated as its own resource type. Bytes and I/O operations are both included for network and disk metrics as they contribute in a different manner to application response time. The modeling is accomplished using a modified non-negative recursive least squares algorithm with directional forgetting. The mathematical details are presented in Section IV.

With any regression analysis there is a risk that the modeled relationships are only mathematical abstractions that overfit the observed data. To prevent this we enforce the condition that an increase in utilization of any resource cannot cause an improvement in response time. This is defined as the positive function constraint for the Performance Modeler, which requires all response time functions of server resources to have positive derivatives at all reachable points.

Although the positive function constraint is obvious for metrics such as disk activity where each additional disk access adds delay, it is less intuitive with regards to resource contention. A VM may independently increase its own CPU utilization due to an increase in its workload. As the CPU is modeled as a queue, this increase in workload increases the response time. Alternatively CPU contention can increase due to colocated VMs increasing their CPU utilizations. This contention blocks access of the target VM to the CPU which causes delay and increases application response time. Since utilization and contention are inherently linked by the same hardware, an increase in contention can lead to a decrease in utilization or vice-versa. Since both these scenarios can degrade application performance, the positive function constraint is used to distinguish whether the utilization or contention is the cause of the change in performance.

### IV. System Algorithms

This section details the algorithms used to construct VARACO according to the outline in Section III. The algorithmic model of the system and where each parameter is used is shown in Fig. 2.

#### A. Performance Modeler

VARACO’s performance models are recursively calculated, thereby adapting to new system behaviors. The application response time, $RT$, is modeled by

---

**TABLE I. VM and Server Metrics used in Performance Modeler**

<table>
<thead>
<tr>
<th>VM CPU Metrics</th>
<th>VM Network Metrics</th>
<th>VM Disk Metrics</th>
<th>Server Memory Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Utilization</td>
<td>Packets Received</td>
<td>Read Operations</td>
<td>Memory Used</td>
</tr>
<tr>
<td></td>
<td>Packets Sent</td>
<td>Write Operations</td>
<td>L2 Cache Misses</td>
</tr>
<tr>
<td>CPU Contention</td>
<td>Bytes Received</td>
<td>Bytes Read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bytes Sent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure 1: VARACO Architecture for an example Data Center containing two servers with two applications of two tiers each**
where \( \phi(k) = f(x(k)) \) is a vector of functions of the resource utilizations readings \( x(k) \), taken from Table I, of each application tier at time interval \( k \),

\[
\phi^T(k) = \left[ f_1(x_1(k)) \quad f_2(x_2(k)) \quad \cdots \right],
\]

and \( \theta(k) \) is a vector of coefficients relating each element of \( \phi(k) \) to the response time. \( \theta(k) \) is determined at each interval by minimizing the following non-negative Least Squares cost function of the observed and estimated response times, \( \hat{RT} \),

\[
J = \sum_{i=1}^{s_p} \lambda^{s_p-i} \| RT(i) - \hat{RT}(i) \| ^2, \theta(i) \geq 0.
\]

The function \( \phi(k) = f(x(k)) \) is constructed such that an increase in any component of \( x \) leads to an increase in \( \phi \). Maintaining \( \theta \) non-negative guarantees the positive function constraint of the Performance Modeler detailed in Section III. The functions mapping \( x \) to \( \phi \) are modeled linearly for disk and memory terms, while the CPU and network terms are approximated by queues where the response time is a function of the number of waiting requests in the queue such that

\[
f(x_c) = \frac{x_c^2}{1-x_c}.
\]

In (3), \( 0 < \lambda \leq 1 \) is an exponential forgetting factor used to decay old measurements in favor of new data, allowing for a dynamic model. As a precaution against poor estimates, the forgetting factor is decreased if the response time estimates drift too far from the observed values. This gives greater weight to new data in parameter estimation. The forgetting factor is set to

\[
\lambda = \begin{cases} 
\lambda_0 & \text{if } \| RT(k) - \phi^T(k)\hat{\theta}(k) \| \leq \frac{c}{2} \\
\lambda_0e^{-\beta RT(k)-\phi^T(k)\hat{\theta}(k)} & \text{if } \| RT(k) - \phi^T(k)\hat{\theta}(k) \| > \frac{c}{2} 
\end{cases}
\]

Where \( \lambda_0 \) is the base forgetting factor, \( c \) is the normalized target response time and \( \beta \) is a decay constant.

It is possible that more resources are monitored than those that affect the application performance. It is therefore possible that many monitored metrics are not being persistently excited. This can lead to the phenomenon of estimator windup where the modeling system becomes unstable. Therefore a recursive least squares directional forgetting algorithm [12] is used. This algorithm applies the forgetting factor only to data not in the null space of the information matrix, \( R \), of the system. The estimator equations are

\[
\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)[RT(k) - \phi^T(k)\hat{\theta}(k)]
\]

\[
K(k) = \frac{\hat{P}(k-1)\phi(k)}{1 + \phi^T(k)\hat{P}(k-1)\phi(k)}
\]

\[
\hat{P}(k-1) = P(k-1) + \frac{1-\lambda}{\lambda} \phi(k)\phi^T(k) R(k-1)\phi(k)
\]

where \( K \) is a gain vector indicating how the previous estimate and the correction should be combined, \( P \) is the covariance matrix of the resource utilizations, \( \hat{P} \) is a modified covariance matrix given the directional forgetting scheme, and \( \phi^T(k)R(k-1)\phi(k) \) represents a new functional forgetting factor. It is shown in [12] that with persistent excitation the information matrix is bounded both above and below.

To satisfy the Performance Modeler’s positive constraint condition a modified version of the nonnegative least squares algorithm [13] is used. The elements of \( \theta \) are split into passive, \( \hat{\theta}_p \), and active, \( \hat{\theta}_a \), sets where \( \epsilon_\ell \) is used to eliminate negative relationships and those likely derived from correlated noise. \( \theta \) is recalculated by (6) using only elements from the passive set. Let \( s \) be a vector of these new \( \theta \) parameters consisting of \( s_p \) and \( s_a \), the passive and active elements respectively. The following calculations are iterated through until no element of \( s_p \) is less than \( \epsilon \):

\[
Q = (s_p \leq \epsilon).
\]

\[
\alpha = -\min \left( \frac{\hat{\theta}(Q)}{\hat{\theta}(Q) - s(Q)} \right)
\]

\[
\hat{\theta} = \hat{\theta} + \alpha(s - \hat{\theta}).
\]

After each iteration, the elements of the passive set are re-determined and \( s \) is recalculated. When this loop exits the active elements of \( \theta \) are set to 0 and the passive elements are set to their respective \( s \) values.

A dynamic renormalization scheme, according to the maximum observed \( \phi \) values, is used to ensure the relative magnitudes of the \( x \) values do not inadvertently skew the results.
B. Resource Controller

1) Application Controller

The Application Controller reduces the state space by using only the controllable resources. \( B_1 \) reduces the full set of resource metrics \( \phi \) to the controllable metrics vector \( u \) by \( u(k) = B_1 \phi(k) \). (15)

The controllable metrics are the CPU utilization and CPU contention. The vector \( u \) contains the functions \( f \) of those respective metrics for each tier. Since \( B_1 \) is not full rank, the right pseudo-inverse \( B_1^\dagger \) is used when necessary. The resource allocation requirements are determined by minimizing the Application Controller cost function

\[
J_a = \|RT(k) - c\| + \alpha_1(RT(k) - c) + \alpha_2\|\tilde{RT}(k) - RT(k-1)\| + \alpha_3 u^\top(k)Wu(k)
\]

(16)

where \( \alpha_1, \alpha_2, \) and \( \alpha_3 \) are coefficients relating the importance of each term. The first term tracks the response time around the target value while the second term biases the response time below the target. The third term minimizes changes in the response time, and the final term regulates the allocations depending on their response time contributions. The matrix \( W \) is a diagonal matrix where \( W_{ij} = (\theta, \phi)_{ij}^\top \). This reduces allocations to metrics that heavily contribute to response time; for example, if contention significantly degrades a VM’s performance, then the contention allocation would be small. The solution to the resource requirement is

\[
u^*(k) = \left( B_1^\top \theta(k) \theta^\top(k) B_1 + \alpha_i W \right)^{-1} \cdot \left( B_1^\top \theta(k) \left( c^\top - \alpha_2 \tilde{RT}(k) + \alpha_2 RT(k-1) - \frac{\alpha_1}{2} \right) \right) \]

(17)

This resource requirement is then mapped back to the base form of each resource reading, denoted by \( v^*(k) \). This is the actual requested CPU and CPU contention. Similar to (15), \( v(k) = B_2 v(k) \), where for these metrics \( B_2 = B_1 \). Therefore, the resource requirement is

\[
v^*(k) = B_2 f^{-1}(B_1 v^*(k)).
\]

(18)

2) Server Controller

Applications are split into two categories, those with SLAs and those that receive best-effort service. Best-effort VMs receive the remaining resource shares after the VMs with Quality-of-Service (QoS) targets have been allotted their determined shares.

Each Server Controller receives the \( v^*(k) \) vector from all its hosted VMs. Due to conflicting requirements, share allocations may need to deviate by \( \Delta v \) from \( v^* \). These revisions are made by minimizing the changes in response times of the applications according to the Controller cost function

\[
J_s = \sum_{i=1}^n w_i \left( \frac{\partial RT}{\partial v_i} \Delta v_i \right)^2,
\]

(19)

where \( n \) is the number of VMs on the server, \( i \) is the index of each VM, and \( w_i \) are weights of each VM’s application by their relative importance in the data center. This ensures the most important applications receive shares during overload.

The derivative in (19) is given by

\[
\frac{\partial RT}{\partial v_i} = \theta_i^\top \left( \frac{\partial f(B_1 v)}{\partial v_i} \right)
\]

(20)

Only the elements of \( B_2 \) applicable to each VM are retained. Let \( v_i = [CPU_i, CPUC_i]^\top \), where \( CPU_i \) and \( CPUC_i \) are the CPU and CPU Contention allocations respectively. There are further constraints on the changes in resource allocations such that

\[
\begin{align*}
[1 & 0] \sum_{j=1}^n (v_{ij} + \Delta v_{ij}) = 1 \quad (21) \\
[0 & 1] (v_{ij} + \Delta v_{ij}) = [1 & 0] \sum_{j=1, j \neq i}^n (v_{ij} + \Delta v_{ij}) \quad \forall i \quad (22)
\end{align*}
\]

\[
\begin{align*}
\epsilon_2 \leq & v_{ij} + \Delta v_{ij} \leq [1 - (n-1)\epsilon_2] \quad \forall i \quad (23)
\end{align*}
\]

The condition in (21) ensures that the CPU allocations sum to 1. The condition in (22) ensures that contention of one VM is the sum of the CPU allocations of all the other VMs. The constraints of (23) bound each allocation between \( \epsilon_2 \) and \( 1 - (n-1)\epsilon_2 \). This creates a minimum VM CPU allocation, for example 1% CPU for OS housekeeping. This condition is also applied to best effort VMs. \( J_s \) is then minimized using MATLAB’s convex optimization toolkit and the new resource allocations are configured on the servers. The overhead of the convex optimization is generally negligible compared to the rest of the system since only the controllable variables are considered and the number of VMs on a CPU core is small compared to the number of VMs used for an application.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

VARACO is tested using the TPC-W benchmark [14]. TPC-W is an example three-tier web application consisting of an Apache proxy front-end, a Tomcat web server, and a MySQL back-end database. We run 400 emulated TPC-W clients in Browsing mode. For each experiment, the SLA to be achieved is defined as the 90th percentile response time in each 10-second time interval being below a target value. Modeler and Controller calculations are performed and implemented for every 10-second window. Each tier of TPC-W is placed in its own VM on separate servers. A second benchmark application is used to generate CPU contention on each host. This benchmark performs CPU intensive calculations upon receiving a request and has an SLA defined by best-effort service.

The physical hosts’ operating system is Red Hat 6.32 and runs the KVM hypervisor. Each server has a dual-core 3 GHz CPU with 4 GB of RAM. The hosts are connected via a Gigabit switch. The VMs are all pinned to the same single CPU core on their respective hosts. The Linux Completely Fair Scheduler (CFS) is used with VMs being given CPU preference by adjusting the Linux Control Groups subsystem
B. Performance Modeler Results

The isolated effects of resource contention on TPC-W’s performance are shown in Fig. 3. The application response time increases exponentially with respect to contention. The fit according to (4) is shown to validate the use of a queuing model. The performance degradation shown is generated by consolidating the VMs two per machine as shown in Fig. 4, and increasing the contention benchmark’s CPU utilization from 10%-75% in 5% increments. TPC-W’s response time degrades even at low levels of contention and increases by three orders of magnitude for high levels.

We demonstrate the importance of considering contention in the performance model by running VARACO with the contention metrics disabled. Fig. 5 shows the TPC-W response time estimates given by VARACO when the contention benchmarks oscillate their CPU utilizations between 10%-60%. The response time estimates track the observed response times poorly. Also, the coefficients for many of the resource metrics incorrectly oscillate to compensate for the unmodeled contention which is causing the large changes in response time. This inaccuracy means this model cannot be used to guarantee an SLA. Systems such as [8] which do not account for contention would fall victim to this when using a work-conserving CPU scheduler.

Using the same consolidation scheme as in Fig. 4, Fig. 6 shows VARACO’s response time estimates with the resource contention metrics enabled. Modeling contention allows TPC-W’s response time to be estimated much more accurately. The response time estimates are 130% more accurate on average than when contention is ignored. Since the performance modeler is a recursive algorithm VARACO gains more knowledge about the system as time progresses. For the experiment shown in Fig. 6 for example, the accuracy of the response time estimates improves 30% from the first to the last 250 time intervals. Fig. 7 shows a comparison of the real and estimated response times for the entire run and last 250 intervals when the estimator has settled. The estimates cluster around the \( \bar{RT} = RT \) \( (y=x) \) line indicating a good fit.

To strengthen VARACO’s results, we show that VARACO models real system behaviors by analyzing the resulting \( \theta \) metric coefficients. Fig. 8 shows the modeled CPU contention coefficient values for each tier of the TPC-W application. After an initial learning phase, the parameters reach a stable value as would be expected given the importance of contention and the model’s fit. As expected, the contention on the web tier is the largest contributor to the application’s response time. Due to the relative magnitudes, this emphasizes not only the importance of modeling contention, but also the importance of considering each application tier separately as different tiers have distinct utilization profiles and resource requirements.

VARACO’s positive function constraint is essential to maintaining a model representative of the physical system. If this constraint is relaxed, some coefficients travel in opposite positive and negative directions, effectively cancelling each other out. Although this could improve the final response time estimate, it makes no physical sense, and the resulting models could not be used in a viable control system.

C. Controller Results

In addition to estimating applications’ response times, VARACO is designed to control response times by altering VMs’ resource allocation levels. Using the consolidation scheme in Fig. 4, we dynamically change TPC-W’s resource allocation levels to achieve a 500 ms 90th percentile SLA, as shown in Fig. 9. We compare this to TPC-W’s response time...
when no control scheme is employed. Using VARACO, the amount of CPU allocated to each tier varies throughout the experiment as the contention experienced by each tier varies, as shown in Fig. 10. This allows TPC-W to achieve its SLA whereas static provisioning does not.

We also run an experiment controlling the response times of two TPC-W instances for the consolidated setup shown in Fig. 11. The TPC-W client load was configured such that the response times varied between 30 ms and 500 ms for an uncontrolled system. Fig. 12 shows the resulting response times for each of the TPC-W instances when controlled with target values of 300 ms. With the control scheme the response times stay below the target value for the entire run, thus achieving the SLA under VM consolidation.

VI. CONCLUSIONS

This work shows the significant performance degradation applications can experience due to resource contention in virtualized environments and the importance of considering it when trying to achieve SLAs with consolidated VMs. We present VARACO, a VM management system that models multi-tier applications and allocates the resources required to achieve SLAs. We show that response time modeling is 130% more accurate when considering resource contention than when not. We also show how applications controlled by the VARACO system can achieve their SLAs in situations where statically allocating resources would cause violations. By considering resource contention and dynamically allocating applications resources, VMs can be consolidated while meeting QoS requirements, leading to more efficient data center operation without violating SLAs.

REFERENCES


