A Performance Interference Model for Managing Consolidated Workloads in QoS-Aware Clouds

Qian Zhu
Accenture Technology Labs
50 W San Fernando Street, Suite 1200
San Jose, CA 95113
qian.zhu@accenture.com

Teresa Tung
Accenture Technology Labs
50 W San Fernando Street, Suite 1200
San Jose, CA 95113
teresa.tung@accenture.com

***Abstract*—Cloud computing offers users the ability to access large pools of computational and storage resources on-demand without the burden of managing and maintaining their own IT assets. Today's cloud providers charge users based upon the amount of resources used or reserved, with only minimal guarantees of the quality-of-service (QoS) experienced by the users applications. As virtualization technologies proliferate among cloud providers, consolidating multiple user applications onto multi-core servers increases revenue and improves resource utilization. However, consolidation introduces performance interference between co-located workloads, which significantly impacts application QoS.**

A critical requirement for effective consolidation is to be able to predict the impact of application performance in the presence of interference from on-chip resources, e.g., CPU and last-level-cache (LLC)/memory bandwidth sharing, to storage devices and network bandwidth contention. In this work, we propose an interference model which predicts the application QoS metric. The key distinctive feature is the consideration of time-variant inter-dependency among different levels of resource interference. We use applications from a test suite and SPECWeb2005 to illustrate the effectiveness of our model and an average prediction error of less than 8% is achieved. Furthermore, we demonstrate using the proposed interference model to optimize the cloud provider's metric (here the number of successfully executed applications) to realize better workload placement decisions and thereby maintaining the user's application QoS.

***Keywords*-Cloud computing; performance interference; QoS-aware;**

I. Introduction

Cloud computing provides an unprecedented opportunity for on-demand computing. The prominence of cloud computing, as evidenced by the deployment and growth of commercial cloud platforms enables businesses to replace their own IT infrastructures with the large pools of compute and storage resources provided by the cloud. In particular Infrastructure-as-a-Service (IaaS) cloud environments employ virtualization technologies to encapsulate applications in virtual machines (VMs) and enable co-hosting independent applications on shared physical resources by providing fault isolation, thereby preventing failures in one application's VM from propagating to others.

However, virtualization does not guarantee performance isolation between VMs [11]. While the hypervisor (*a.k.a.* virtual machine monitor) slices resources and allocates shares to different VMs, the behavior of one VM can still affect the performance of another adversely due to the shared use of resources on the server. For example, a disk I/O bound application may be co-located with another application which is also disk I/O intensive. The result of such assignment is that two VMs may access the shared datastore simultaneously, causing an increase in the I/O latency and further leading to performance degradation for both applications during periods of disk contention.

Furthermore, virtualization limits the visibility to the cause of the performance degradation from consolidated VMs. Specifically, an application running in the same virtual machine on the same server at different times will see wide disparity in performance based on the work performed by other VMs on the same host [11]. This is referred to as *performance interference*. For cloud users, performance interference implies that paying for a quantity of resource is not equal to achieving an application QoS (e.g., that an application finishes in a specified amount of time).

To overcome the challenges imposed by performance interference, it is critical to understand the interference quantitatively. Existing work has proposed models to study the performance interference [13], [7], [8], [18], [12], [14]. However, previous work only focused on a particular type of resource, e.g. CPU or cache interference. Moreover, they did not consider the time variance in application workloads.

In this work, we propose a performance interference model where we consider the interference from all types of resources. The main contribution is an influence matrix which estimates the additional resources required by an application needed to overcome the interference of consolidated applications and achieve a desired QoS. Then we introduce a consolidation algorithm that leverages the interference model to optimize the provider's metric. Our contributions are four-fold:

- An interference model provides a holistic view of interference coming from all types of resources and also accounts for time variance in application's resource usage.
- A consolidation algorithm based on the interference model achieves an optimal consolidation configuration.
- An evaluation of our proposed model and algorithm uses a test suite of applications and the SPECWeb2005 benchmark. The prediction accuracy of our interference

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.25

model is less than 8% in average.
- The proposed consolidation algorithm is able to outperform a static optimal approach and a state-of-art approach, *Q-Clouds* [13], in optimizing the provider's metric, i.e. the number of successfully executed applications, with negligible overhead.

The rest of the paper is organized as follows. We describe the performance interference problem due to server consolidation in Section II. Then the details of interference model and consolidation algorithm is presented in Section III. In Section IV, we report the experimental evaluation results. We compare our work with related research efforts in Section V and conclude in Section VI.

II. Problem Description

Managing application QoS remains a key challenge for cloud infrastructures. A fundamental problem is that resource contention from co-existing virtual machines on a shared server impacts application performance. However, the cloud user cannot decide where to deploy her application or control the presence of other applications. The solution today is either to accept cloud's limited guarantees which results in application performance degradation for the user or to over-provision resources which leads to inefficiency in the provider. In this section, we first highlight the performance interference problem caused by server consolidation. Then, we describe the type and structure of applications which we consider in this work.

Performance Interference with Consolidated VMs If cloud users ran their applications on dedicated servers, there would be no interference and resource level guarantees provide application QoS guarantees. However, a basic principle of cloud computing is resource sharing and scaling. Given that each hosted application is unlikely to fully utilize all resources all the time, consolidating multiple VMs results in more efficient utilization of server resources, reduction in the total number of servers, and thus reduction in the provider's costs. This consolidation leverages virtualization technologies, where the applications are encapsulated in VMs which advocate a decoupled view of the compute, network, and storage resources they can access from the hardware resources. On the servers hosting multiple VMs, such resources are partitioned among VMs with a scheduling policy to decide how each VM gets its share.

While virtualization provides fault isolation and improved manageability, it does not provide perfect performance isolation. Consolidated VMs may compete for all types of resources, such as CPU cores, memory bandwidth, and disk I/O. Hosting VMs on a multi-core package that incorporates a shared last level cache (LLC) creates an opportunity for the VMs to interfere with each other.

We illustrate the problem of performance interference from consolidating multiple applications onto the same server in Figure 1. The experiment was conducted using a server with an AMD dual-core $Opteron^{TM}$ processor($2.4GHz$). The server has 20GB of main memory and 74GB local disk space. Without loss of generosity, we chose a benchmark application that is disk I/O bound to run in a VM. Then we deploy a second VM on the same server which hosts an application that is either CPU, memory, disk, or network intensive. The execution time of hosting the disk application VM alone on the server was taken as the baseline. Then we measured the application performance in the presence of the second VM. The ratio is the performance slowdown. We also varied the workload intensity to see how the application performance is impacted (here our workload is the size of the data file). As

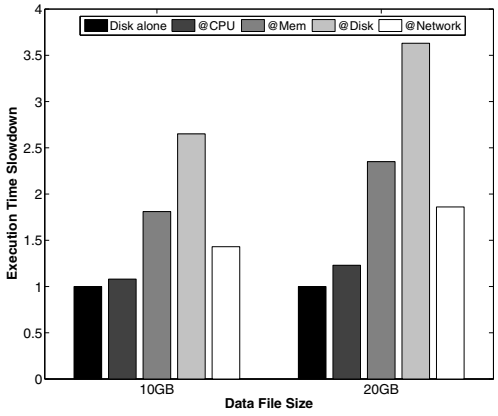

Fig. 1. Performance Degradation due to Consolidation

illustrated in the figure, the performance degradation in terms of execution time experienced by the disk I/O application varies based on the types of resources with which the co-located VMs compete. When a disk bound application shares the server with an application with the similar resource consumption behavior, the performance can be slowed down as much as 265%. Moreover, as we increase the data file size, i.e., more workload, the performance degradation becomes more significant. The extent of degradation clearly depends on the combination of applications that are co-located. Therefore, a quantitative interference model is critical for determining the impacts of consolidating multiple VMs on application QoS.

Applications and Metrics The applications we consider in this work comprise a series of dependent services, which we denote as $S_1, S_2, ..., S_n$. One service could be *data-dependent* and/or *control-dependent* on another service. Examples of such applications include scientific workflows [1], [5], [2], business processes [3], and N-tier enterprise web applications. The average resource usage of individual application services is significantly smaller than their peak values and usually, they do not peak at the same time, which creates opportunities for consolidation. The required resource consumption (CPU, memory, disk and network) from each service S_i also depends on the application workload which could vary during the execution. Note each service S_i is hosted in a VM V_i, which then could be deployed onto data center servers. Then according to the application demand on resource usage, resources are dynamically allocated. The application performance degrades

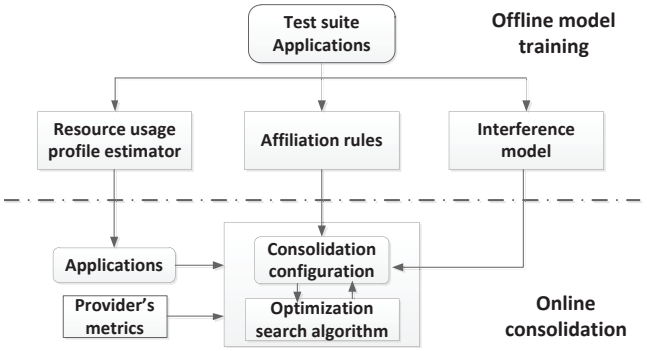

Fig. 2. Overall Design of Consolidation Approach

if the host server cannot satisfy the requested resources.

Each application is specified with one or more QoS metrics that need to be achieved. We assign the execution time as the application QoS metric in this work. Additionally, we associate every application with a time deadline where an application is considered as successfully executed if it completes before the deadline. There are multiple cloud providers which vary in terms of resource capacity, pricing policy, virtual machine (VM) starting time, VM scheduling as well as affiliation rules. Such affiliation rules facilitate the mapping from applications to the underlying physical servers, based on the characteristics of the application resource consumption. For example, a server with a powerful CPU is more suitable for a CPU-intensive application than a server with a large memory. Details on such affiliation rules will be discussed in Section III. As a result, they provide different levels of confidence for hosting different types of applications. A cloud environment comprises of heterogeneous computing nodes denoted as $N_1, N_2, ..., N_m$. Assigning a VM, V_i, to a server N_j raises the following questions: First, does N_j provide the resource capacity to satisfy the requests? Second, if the provider consolidates V_i with other VMs that are already running on N_j, would there be an impact on V_i's performance leading to a missed application deadline? We choose the number of successfully executed applications as the cloud provider's metric that needs to be maximized.

We propose a performance interference modeling approach to quantitatively predict performance degradation due to server consolidation so that the user's application QoS metric, i.e. the deadline, is guaranteed. Meanwhile, the provider leverages the proposed model to maximize the number of successfully executed applications and to improve the resource utilization.

III. Performance Interference Model and Consolidation Algorithm

This section presents our proposed solution to efficiently consolidate applications in cloud computing environments, by considering the performance interference among such co-located applications. We first give an overview of our approach. Next, we discuss the details on applying models to estimate the resource contention among consolidated applications and its impact on the performance in terms of execution time. Specifically, the models are a resource usage profile estimator, an interference model and a set of affiliation rules. Finally, we present the consolidation algorithm which targets at optimizing the provider's metric, the number of successfully executed applications.

The overall design of our approach is illustrated in Figure 2. It consists of two major components: online consolidation and offline model training. Recall that an application comprises of multiple services. The online consolidation algorithm assumes that for each application service to be scheduled, a *resource usage profile* representing the CPU, memory, disk, and network utilization is available. We train a hidden Markov model (HMM) to predict such profiles.

Based on the resource usage profile, we apply a set of *affiliation rules* to map application VMs onto the shared servers. The initial mapping gives us a good start for an efficient consolidation configuration. To account for resource contention, a performance interference model estimates the impact of consolidated applications on resource consumption and the performance degradation. We propose an *influence matrix* to represent such interference quantitatively. Note that we base the model training data on a test suite of applications which covers a spectrum of different resource usage characteristics including CPU-intensive, memory-intensive, disk I/O-intensive and network I/O intensive. We host each test-suite application in a single VM and test this VM both assigned on an isolated or a shared server.

Next, the online consolidation algorithm leverages the models trained offline and an optimization algorithm based on hill climbing to search for the optimal consolidation where the provider's metric is maximized without violating the application QoS metrics. Considering the dynamics of application workloads, we invoke the consolidation algorithm periodically during the course of application execution.

A. Resource Usage Profile Estimator

The relationship between resource usage and the performance of an application is complex. To facilitate our analysis, we use a resource usage profile to capture the key characteristics of application resource consumption. The resource usage profile is a vector which contains performance metrics such as CPU usage%, CPU sys%, Mem bandwidth, Disk reads and writes etc. We denote such metrics as C_1, C_2, ..., M_1, ..., D_1, ..., N_1 in Equation 1. Let us consider an application where a data file first needs to be loaded from the disk then the application proceeds with analyzing the data. In this case, The data loading phase is disk I/O bound while the data analysis consumes CPU cycle intensively. Moreover, when multiple data files need to be accessed for the application processing, the intensity of consuming both the disk I/O and CPU cycles could vary based on the file sizes. Thus, Due to the dynamics in the application workload, it is critical to capture the time variance in resource consumption. Therefore, instead of using a single value for each metric, we use a time series. The profile

is impacted by both the application workloads which is time-variant and the host server resource availability. Note that the resource usage profile represents the resource consumption of an application when it runs on a dedicated server, i.e., without resource contention from other applications.

For simplicity, let us consider an application, $app1$, with a single service. The resource usage profile for $app1$ is denoted as the following:

$$R_{app1} = < C_1, C_2, C_3, ..., M_1, M_2, ..., D_1, ..., N_1, ... >, \\ C_1 = < C_1^{t_1}, C_1^{t_2}, C_1^{t_3}, ..., C_1^{t_n} > \quad (1)$$

, where t_i is the time point. The resource usage profile serves an input to the performance model where a succinct format significantly reduces the training and prediction overhead. The profile simplification is two-fold: first, correlation analysis reduces the number of performance metrics in the vector; Second, represent each time series with sampled points. From the performance metrics data we have collected from running the test suite applications we identify correlations between metrics. Then performing a *Pearson Correlation* analysis ascertains the pair-wise metric correlation, and we remove one of the metrics in a pair if it is strongly correlated with the other. By doing so, only uncorrelated performance metrics remain in the profile. We then sample data points from the time series from each metric. The sampling rate trades off between the overhead of maintaining the measurements and the accuracy of preserving the time series pattern. We consider the time variance of resource usage as two applications with the same type of resource requirement may not compete for the resource simultaneously. In this case, it might be okay to consolidate them onto the same server. We define a *global time space* which every application execution is mapped to so that the time points used to take sample data is universal among all applications. Our work selects a sampling point whenever the change of resource consumption from any of the test suite applications exceeds a set threshold. The threshold is adjustable and we use 10% for our offline training.

As stated previously, the application resource usage depends on the workload which could be dynamic throughout the application execution. We apply the *Kalman filter* [22] to predict the application workload with the assumption for our applications that future behavior is related to execution history. An unpredictable burst in the workload is beyond the scope of this work thus is not considered. The training process varies both the workload and the resource capacity of the hosting VM for each test suite application. We performed a *10-fold cross validation* over the training data. A *Hidden Markov Model (HMM)* has been applied for estimating the resource usage profile. This HMM generates the resource usage profile at sampled time points for given application workloads and VM resource availability.

A hidden Markov model(HMM) is a statistical model that assumes that a sequence of observations $\mathcal{O} = \mathcal{O}_1, \mathcal{O}_2, ...\mathcal{O}_T$ is generated from a set of N states ($Q = Q_1, Q_2, ..., Q_n$), with state transition probabilities (a_{ij}) between them, and emission probabilities $b_j(\mathcal{O}_t)$, denoting the probability of an observation $\mathcal{O}_t$ being generated from the state j. We use a first order HMM in which the current state is only dependent on the previous state. We generate our HMM for resource usage profile estimation in the following way. A hidden state S_t is the resource use profile at time step t while the observation is a set that comprises of the values of the application workload and the current resource availability. The transition matrix characterizes the distribution over the states for the performance related metrics. a_{ij} is calculated as the ratio between the number of transitions from state S_i to S_j and the total number of transitions originated from S_i. We use multivariate Gaussian distribution for $b_j(\mathcal{O}_t)$, for each of the hidden states.

Given the above HMM, the task of estimating the resource usage profile is the following: Given the predicted application workload and VM resource availability, $\mathcal{O} = \mathcal{O}_1, \mathcal{O}_2, ...\mathcal{O}_T$, and the model λ, find the most likely resource usage state sequence $\mathcal{Q}_{max} = q_1, q_2, ..., q_T$. Based on an observation sequence $\mathcal{O}$ and a set of states $\mathcal{Q}$, we use the Forward-Backward algorithm to learn the parameters of the HMM (i.e., λ) [15]. Then using the trained HMM and the sequence of observations (i.e., application workload and resource availability of the VM), the Viterbi algorithm obtains the best hidden state sequence [15]. Thus we estimate the resource usage profile by executing it shortly on the server.

B. Interference Modeling

If the application VM $app1$ is hosted on a dedicated server, we represent the relationship between the application resource usage profile and its execution time as

$$T_{app1} = f(R_{app1}) \quad (2)$$

where f is a function trained from a *Support Vector Machine (SVM)* regressor, given the estimated resource usage profile, R_{app1}, as the input. However, when there are consolidated applications, the performance of $app1$ may degrade due to resource contention. We train a performance interference model to predict the resource usage for consolidated applications.

The intuition is to adjust the resource usage of $app1$ based on the resource consumption as well as the potential interference of other applications that will be consolidated onto the same server. A *dilation factor* over the resource usage to $app1$ accounts for the impact of resource contention from co-located applications (i.e., an extra requirement compared to if it were running on a dedicated server). We study interference from all types of resources and consider the time-variant resource usage. An *influence matrix* is proposed for adjusting the resource usage profile. We now present the details of the influence matrix. We use the influence matrix to calculate the dilation factor to the resource consumption of $app1$ due to another application, app_2, that will be co-located on the same host. As such applications may compete resources such as CPU, memory bandwidth, disk I/O etc. with $app1$, or, it may cause unexpected cache activities, the performance of both applications will be degraded. Let us denote the resource

usage profile of $app1$ after the consolidation as R'_{app1} and the influence matrix is denoted as $\mathcal{M}$. Therefore,

$$d_{factor} = R_{app1} \times \mathcal{M}$$

$$\mathcal{M} = \begin{array}{c} \\ C_1 \\ C_2 \\ \vdots \\ M_1 \\ \vdots \\ D_1 \\ \vdots \\ N_1 \end{array} \begin{array}{c} \begin{array}{cccccccc} C_1 & C_2 & ... & M_1 & ... & D_1 & ... & N_1 \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & ... & a_{1i} & ... & a_{1k} & ... & a_{1m} \\ a_{21} & a_{22} & ... & a_{2i} & ... & a_{2k} & ... & a_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2} & ... & a_{ii} & ... & a_{ik} & ... & a_{im} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & ... & a_{ki} & ... & a_{kk} & ... & a_{km} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mi} & ... & a_{mk} & ... & a_{mm} \end{pmatrix} \end{array}$$

$$R'_{app1} = R_{app1} \cdot d_{factor} \tag{3}$$

The influence matrix $\mathcal{M}$ is a matrix where m is the number of metrics in the resource usage profile. Each row or column corresponds to a metric in Equation 3 where a_{ij} represents the impact coefficient of metric j on metric i. The result of multiplying the resource usage profile, R_{app1}, with the influence matrix is a row vector. We refer it to as the *dilation factor*. An element d_{R_i} represents how much the i^{th} metric gets dilated due to the interference from other applications (e.g., app_2). As we consider the contention from all types of resources, the impact from all the metrics in R_{app1} on the i^{th} metric, if there is any, contributes to the value of d_{R_i}:

$$d_{R_i} = \sum_{k=0}^{m} r_k \times a_{ki}, \text{ where } r_k \text{ is the } k^{th} \text{ metric in } R_{app1} \tag{4}$$

Then the dot product of R_{app1} and the dilation factor results in the adjusted resource usage profile R'_{app1} which serves as the input to Equation 2 for predicting the performance of $app1$ after consolidating another application $app2$ on the same server, i.e., $f(R'_{app1})$.

Now we present the details of generating the influence matrix. For each of the applications in the test suite, we vary the resource usage intensity. For example, vary the CPU consumption percentage from 10% to 100% with an interval of 10% for the CPU-intensive applications. Then we consolidate the application with one or more applications from the test suite on a shared server. Let us denote the application as $app1$. At sampled points, measure the resource usage profile of $app1$ before and after consolidating with other applications. We refer the profiles as R^{bc}_{app1} and R^{ac}_{app1}, respectively. Then the ratio between the two vectors denotes the dilation in terms of resource usage. We regress the ratio of the i^{th} metric, d_{R_i}, on the profile R^{bc}_{app1} to train the coefficients, i.e. a_{ki}, for the i^{th} column of the influence matrix.

Note that the influence matrix corresponds to individual applications thus it could lead to a large number of matrices with non-negligible overhead from training and maintainance. Based on our experiments, we observe that applications of similar behavior of resource consumption could share the same influence matrix. Therefore, instead of having one influence matrix for each application, we generate a matrix for a group of applications where within the group, the application resource usage profiles are similar. The similarity of two resource usage profiles is calculated as follows. For the pair of resource usage vectors at each sampled time point, we calculate their Euclidean distance. The average value over all the time points is taken as the similarity between the two profiles. Applications are put into the same group if their similarity value is below a certain threshold (which is 0.3 in this work).

$$d^i_{factor} = R_{app1} \times \mathcal{M}_i \text{ where } i = 1, 2, 3$$
$$R'^i_{app1} = R_{app1} \cdot d^i_{factor}$$
$$R'_{app1} = w_i \times R'^i_{app1} \text{ where } w_i = 1/s_i \text{ and } \sum_{i=1}^{3} s_i = 1 \tag{5}$$

Equation 5 demonstrates how to apply the trained influence matrix. We first estimate the application resource usage profile using the proposed approach presented in subsection III-A. Then we compare the estimated profile with the profiles from the test suite applications. Choose the k most similar resource profiles based on the similarity value. The value of k can be adjusted so that a small value might impact the accuracy of resource usage estimation while a large value incurs estimation overhead. We choose k to be 3 in this work.

Next step retrieve the influence matrices associated with the selected resource usage profiles and apply Equation 3 to predict the adjusted resource usage profile for the application. As stated previously, applications which share similar resource consumption characteristics could use the same influence matrix. Finally, the application resource usage profile after consolidation is the weighted average over the 3 estimations, where the weight is the reciprocal of its normalized similarity value (the sum of the 3 normalized similarity values is equal to 1). Note that if the application usage profile is different from all the existing profiles, we must train its influence matrix.

C. Affiliation Rules

Based on the estimated resource usage profile, we could apply our proposed interference model to decide whether co-hosting two applications on the same server will violate their deadlines. However, due to the large number of potential consolidation configurations, a random start would likely lead to significant search overhead and risk the application at missing its deadline.

We designed a set of fuzzy rules to initialize the search for the optimal consolidation configuration. In a procedure called *fuzzification*, we define fuzzy sets by applying membership functions to map the input variable into this set with a degree value in a continuous interval between 0 and 1. For example, if the CPU frequency is 3.0GHz, the membership function of fast fuzzy set may map it to this set with a degree of 0.9. Or, if the application CPU usage is over 80% for half of its execution, we may map it to the CPU intensive set with a degree of 0.75.

Next, all possible fuzzy rules are automatically generated by enumerating fuzzy sets as the antecedents and consequents

Antecedents	Consequents
application is CPU intensive **AND** the server has a powerful CPU	OK to host
$app1$ is disk intensive **AND** $app2$ is disk intensive	Do NOT consolidate

TABLE I
EXAMPLES OF AFFILIATION RULES

of these IF-THEN rules. Such rule set needs to be further pruned because some of the rules are not applicable. We use applications from the test suite to create cases to match the antecedents of the rules. We refer to this total number of cases as N_T. Then count the number of cases is where the consequences of the rule are satisfied and we denote it as N_S. The ratio, N_S/N_T, denotes the probability of the rule. Rules with probability less than 0.5 are removed. Moreover, a rule is more likely to be executed if it has a higher probability than other rules. We show examples of these fuzzy rules in Table I.

Finally, we trigger a fuzzy rule once the antecedents are matched. Note that such affiliation rules offer a good initialization for the search of optimal consolidation. But it does not guarantee achieving the application QoS metric. Therefore, we need to leverage the interference model that is presented from the previous subsection.

D. Consolidation Algorithm

Algorithm III.1: ONLINECONSOLIDATION(App, T_d, R_S)

INPUT App: An application
T_d: Application deadline
R_S: Resource availability for all servers
OUTPUT Optimized consolidation configuration
for each $S_i \in App$
// Estimate the resource usage profile
$R_i = EstimateResourceUsage(S_i, R_S)$;
// Apply affiliation rules for mapping
$C_{opt} = ApplyAffiliationRules(R_i, R_S)$;
// Predict application performance
$T_p = PredictExecutionTime(R_i, C_{opt})$;
// Search for the optimal consolidation configuration
while $(true)$
$C' = SearchConsolidationConfig(S_i, R_S, T_d)$;
if $NumApp(C') > NumApp(C_{opt})\&T_p \leq T_d$
$C_{opt} = C'$;
$break$;
// Update resource availability for each server
$R_S = UpdateServerResourceAvail(C_{opt})$;

Fig. 3. Online Consolidation Algorithm

We present the details of the online consolidation algorithm to efficiently accommodate applications with the goal of guaranteeing of achieving application QoS. Figure 3 presents the consolidation algorithm. For each of the services from the submitted application, execute the application for a short period of time to generate the resource usage profile. Recall that our proposed resource usage profile estimator requires application historical data to predict future workload as well as the resource usage characteristics.

We run the application long enough to collect 30 data samples in order to predict the resource usage for an adjustable time window in the future. During the application execution, we periodically invoke the resource usage profile estimator to capture the dynamics in the application workload so that its resource usage profile is up-to-date. Based on the resource usage profile, the consolidation algorithm maps the application onto the servers by applying the affiliation rules. Recall that the rules offer a good starting consolidation configuration which does not necessarily guarantee the application QoS metric or to optimize the cloud provider's metric, which is the number of successfully executed applications.

In the next step, we use our interference model to determine whether we will be able to avoid missing the application deadline for the both the applications that are already running on the server and for the one that will be consolidated onto the same server. The result of this step adjusts the consolidation configuration from the fuzzy logic so that the application QoS metric can be guaranteed. However, the resulting configuration might not optimize the provider's metric. Therefore, a search algorithm based on hill climbing searches for a better consolidation configuration. We decide that the current consolidation is an improvement if the number of successfully executed applications could be increased. Once such a configuration has been decided, the status of the available resources on servers are updated as well. Due to the dynamics of the application workload, we trigger the consolidation algorithm periodically if a potential violation on the application QoS metric is detected. Specifically, there are two scenarios that require to invoke the consolidation algorithm. First, when the estimated application resource usage profile is significantly different from the actual resource consumption. In this case, the application may be handling unexpected workload or experiencing rare behaviors. Secondly, if there are new applications submitted to the system, we will trigger the consolidation algorithm to accommodate them with an optimal configuration. Other than the two previously stated scenarios, we also trigger the algorithm at a certain time interval in order to avoid any application QoS metric violation and guarantee that the current consolidation configuration is optimal. Such time interval can be adaptive and in our experiment, it is set to be every 3 minutes. Applications might need to be paused and migrated according to the new consolidation scheme.

IV. EXPERIMENTAL EVALUATION

This section presents results from a number of experiments we conducted to evaluate our proposed performance interference modeling and consolidation algorithm.

A. Experimental Setup

We implement and evaluate our performance interference model and consolidation algorithm on a virtualized cluster, which consist of 4 HP ProLiant BL465c G1 blades Each blade has an AMD 2$\times$ dual-core $Opteron^{TM}$ 2216 HE processors($2.4GHz$) with a 20GB of main memory and 74GB local disk space. The AMD processor incorporates a two level

cache hierarchy, where each core has its own L1 (128KB) and L2 (1024KB). The cluster is provisioned with 7 TB of SAN storage. The blades within the cluster are interconnected with switched 1Gb/s Ethernet. In our experiments, we run 20 VMs that are consolidated onto these 4 blades, some run the Ubuntu 11.04 operating system while the others run Windows 7. All VMs reside on VMware ESX 5.0 hosts and are provisioned using vSphere 5.0.

We created a test suite of benchmark applications to cover a spectrum of resource usage characteristics from CPU-intensive, memory-intensive, disk I/O-bound to network-intensive. Table II summarizes the applications. In the CPU-intensive category, `PrimeGen` generates all the prime numbers within a user specified range. `Dense Matrix Multiplication` multiplies two dense matrices. `Busy Loop` iterates two double-precision operations with a sleep time between loops. `N-body` simulates a dynamical system of particles where n is the number of bodies [4]. `Matrix Transpose` and `Sparse Matrix-Vector Multiplication` move a significant amount of data, thus are memory-intensive. `Smith-Waterman` is a computational biology application which performs local sequence alignment,i.e. to determine similar regions between two nucleotide or protein sequences [16]. Although it spends a reasonable amount of time computing, the kernels launched are very small. So we consider it as a memory-bound application. While the `Random Access` performs random accesses to the memory and involves a lot of cache activities. For both the disk and network categories, we have one representative application for each group. Each application exposes one or more parameters that can be adjusted to change the intensity of resource consumption. For example, changing the sleep time in `Busy Loop` impacts its CPU usage.

Moreover, we use the test suite applications to compose complex applications (exposing complex resource usage behaviors) for our experimental evaluation. Note that although our work can be extended to applications that are composed of multiple services, we consider applications with a single service in this work. In order to simulate the time variance in application workloads, we use the enterprise workload, the SPECWeb2005 benchmark. The amount of work performed by a SPECWeb VM changes with the number of transactions processed, which is varied to simulate temporal load variations.

We compare our proposed solution against the following goals:

- Demonstrate that our proposed performance interference model effectively estimates application performance with the presence of co-located applications.
- Demonstrate that our consolidation algorithm improves the number of users application QoS that is accommodated into the cloud environment, which leads to provider's revenue gain.
- Demonstrate that the overhead of our algorithm is negligible, and the algorithm scales to a large number of applications.

Application	Resource Type
PrimeGen	CPU-intensive
Dense Matrix Multiplication	CPU-intensive
Busy Loop	CPU-intensive
N-body	CPU-intensive
Matrix Transpose	Memory-intensive
Sparse Matrix-Vector Multiplication	Memory-intensive
Smith-Waterman	Memory-intensive
Random Access	Memory-intensive
Disk File Copy & Paste	Disk I/O
Network File Transfer	Network I/O

TABLE II
TEST SUITE APPLICATIONS

B. Interference Model Validation

In this subsection, we evaluate the performance interference model, and show that our model accurately estimates the application performance in presence of consolidated applications.

We consider both *simple applications* and *composite applications* A simple application is a single application from the test suite, while a composite application is a mix of multiple test suite applications. The motivation to create composite applications is to represent real-world applications that are a mix of resource usage characteristics. Thus, we check if our interference model accurately predicts the performance of applications with complex resource usage characteristics in the presence of other consolidated applications. The workload intensity is varied among applications. In this experiment, we randomly select test suite applications to create six composite applications, denoted as `C_App1`, `C_App2`, ..., `C_App6`.

We first evaluate our interference model using the simple applications. Each application VM occupies one CPU core. We run three instances of SPECWEb2005 at 25%, 75% and 100% intensity, respectively, to simulate the different scenarios of resource contention and workloads. For the rest of the CPU cores, three applications are randomly selected from the ten test suite applications and the three SPECWeb2005 instances.

Figure 4 reports the resulting average prediction error and the standard deviation for each of the workloads. The average prediction error is less than 5% across all the simple applications. Although we collect the training data from these applications, the resource consumption behaviors as well as the workload from SPECWeb2005 instances are unknown. This demonstrates our proposed interference model is able to estimate the impact of resource contention from co-located applications which is critical for accurate predictions for application performance.

Next, we use the six composite applications for model validation. Similarly, the VMs that host the composite applications are randomly consolidated with three applications chosen from the simple/composite applications as well as the SPECWeb2005 instances running at different levels of intensity. We measure the application execution time and

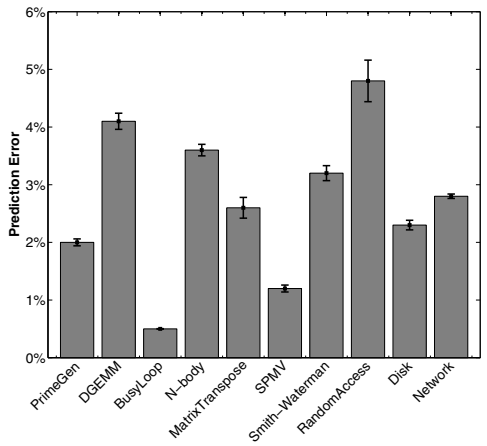

Fig. 4. Average and Maximum Prediction Error Using Simple Applications

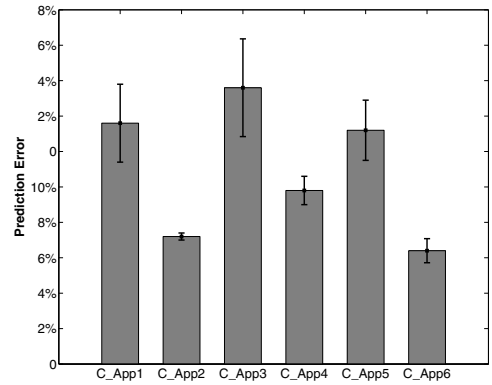

Fig. 5. Average and Maximum Prediction Error Using Composite Applications

demonstrate the average prediction errors and the corresponding standard deviation in Figure 5. As we can observe from the figure, our proposed interference model achieves a useful prediction accuracy: the average prediction error is less than 8% for all the composite applications. This demonstrates the effectiveness of our interference model in predicting application performance degradation caused by consolidated applications, even when the application expose complex resource usage characteristics with dynamic workloads. Since there exists a wide variance in the amount of degradation across the consolidation scenarios, as illustrated in Figure 1 from Section II, accurate performance prediction from our interference model serves as an important step for achieving an optimal consolidation configuration. We evaluate our proposed consolidation algorithm in the next subsection.

C. Performance Comparison

We now evaluate our proposed consolidation algorithm and demonstrate that the number of successfully executed applications is close to the total number of applications submitted in cases where application workloads are static and dynamic.

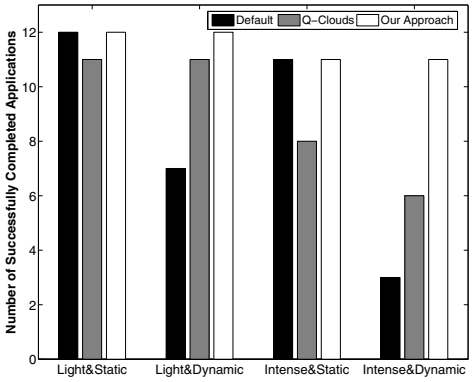

Fig. 6. Performance Comparisons of `Default`, `Q-Clouds` and Our Approach

We compare our approach with two algorithms. The first is a default case where the system assumes that resources required to meet the application QoS do not change as applications are consolidated (i.e. QoS-unaware resource allocations). However, we run an exhaustive search and take the consolidation configuration which leads to the largest number of successful applications. It is referred to as `Default`. The other algorithm is referred to as `Q-Clouds` where the resources allocated to the application VMs are adjusted due to consolidated applications based on a control model [13]. The additional resources is allocated to the VMs to maintain the application QoS metric come from the available resources after the initial assignment.

We use a set of 12 applications, each of which is a either a simple or a composite application. There are two characteristics for the application set that can be varied: workload intensity and time-variant dynamics. Specifically, the workload intensity is categorized as *light* and *intensive*. While the time-variant dynamics includes *static* and *dynamic* to denote if the workload is temporally variant. Thus, we have four combinations of the application set. Recall that an application is successfully executed when it completes before the deadline.

We demonstrate the results in Figure 6. For the four scenarios we considered, our proposed consolidation algorithm handled all or close to all submitted applications successfully. In the static cases, `Default` achieved the best performance as it was the optimal consolidation configuration based on the exhaustive search that stays optimal throughout the run.

However, when the workload gets dynamic, the optimal configuration from the beginning of consolidation does not hold throughout the whole execution, i.e, *app1* and *app2* start interfering with each other at a certain point which results in significant performance degradation. `Default` finished 7 out of 12 submitted applications when the workload is light as there are resources available the servers can provide to handle additional demands. When the workload becomes intensive, the number of successfully executed applications drops to 3

as applications start to compete for the same type of resources shortly after they start due to changing resource usages. In this case, it is important to take the time variance in resource consumption into account.

Q-Clouds, has the ability to deal with the workload dynamics. However, it is based on the assumption that there is enough over-provisioning to handle the additional demands caused by consolidated applications. Under Q-Clouds, performance is impacted when the application workloads are intensive as there would not be enough excess resources left for later adjustment. As a comparison, our proposed approach could estimate the resource usage profile periodically and update the consolidation configuration based on the interference caused by the current workload. The results demonstrate that our approach is able to estimate the performance interference in presence of consolidated applications and to capture the time-variant characteristics of the application workload. Thus, by leveraging our interference model and consolidation algorithm, the cloud provider is able to optimize its metric.

D. Algorithm Overhead and Scalability

We now evaluate the overhead of our consolidation algorithm specifically from training the performance interference model and from executing the online consolidation algorithm.

The interference model training requires executing the application test suite with different levels of workload intensity. As the execution time of all applications is in the order of minutes, the training time of the resource usage profile estimator, affiliation rules and the interference model takes less than 1 hour. Since this is offline and has to be done one time, we argue that the overhead is reasonable.

The online consolidation algorithm triggers an application deadline will be missed due to newly consolidated application VMs. Thus, the more dynamic and intense the application is, the more the algorithm will be invoked for a better consolidation configuration. We use simulation for evaluating the scalability of our algorithm. The result is illustrated in Figure 7. We vary both the number of applications and the workload dynamics. Note we use composite applications in this experiment. In the case where the application workload is static, the overhead of running the proposed algorithm is around 2.3 seconds for 16 applications and it scales to 6.8 seconds when the total number reaches 128. While there is dynamics in the workload, we need to run the algorithm more due to the variation. In this case, it takes 4.3 seconds to provision 16 applications while 128 applications require less than 20 seconds. Although Default performs the best when the application workload is static, the exhaustive search is in the order of magnitude of hours for 128 applications thus not applicable. In comparison, the overhead of our proposed algorithm is negligible and the consolidation algorithm scales.

V. Related Work

We now discuss the research efforts relevant to our work from the areas of interference analysis and modeling, and QoS-aware consolidation.

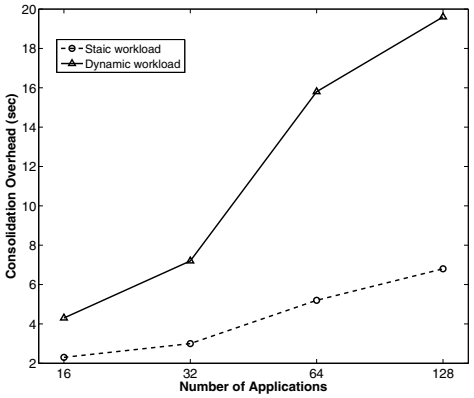

Fig. 7. Scalability of Our Consolidation Algorithm

Interference Analysis and Modeling: Existing work has contributed to the performance interference effects of consolidated VMs by considering each type of resources individually: CPU [13], [7], [21], cache [8], [18], [19] and I/O [6], [12], [14]. Nathuji *et al.* propose Q-Clouds, which is an online MIMO model to capture the performance interference effects in terms of resource allocations [13]. This work adjusts processor allocation for applications based on the required SLA, compensating for performance degradation by allocating additional resources. Furthermore, the authors use *Q-States* to differentiate various levels of SLAs so that they can make best utilization of the resources. Govindan *et al.* studied the cache interference by proposing a simulated cache [8]. The cache activities are then estimated by simulating their interference. TRACON proposed by Chiang *et al.* utilizes modeling and control techniques from machine learning [7]. The application performance is inferred from an interference prediction model which is based on the resource usage. Similarly, we investigate the interference effects of consolidated applications by proposing a quantitative model. However, our work is distinctive in the following ways. First of all, instead of focusing on a specific type of resources, our proposed approach provides a holistic view of interference cross all the resources. The impact between resources is captured in the influence matrix. Moreover, we also considered the time variance in resource usage by using a time series for each performance metric in the resource usage profile.

QoS-aware Consolidation: In cloud computing environments, research has investigated on achieving the application QoS metrics in the presence of consolidation [10], [11], [20], [17], [9]. Koh *et al.* proposed an interference model based consolidation scheme where each application is associated with a resource usage vector [11]. Then a clustering algorithm is adopted where applications within a cluster represent similar resource usage characteristics. Therefore for a new application, it is first grouped into one of the clusters and consolidated with applications that are least likely to cause performance interference. Our proposed consolidation algorithm leverages

the interference model to predict the application performance so that each application is guaranteed to complete before its deadline. Moreover, the goal of the proposed algorithm is to optimize the provider's metric, which is the number of successfully completed applications. The consolidation algorithm is based on a search for an optimal consolidation configuration and is invoked periodically. Srikantaiah *et al.* proposed a multi-dimensional bin packing algorithm for energy-aware consolidation [17]. Our algorithm is based on a hill climbing search with negligible overhead.

VI. Conclusion

Cloud computing provides an unprecedented opportunity for on-demand computing. As one of the key advantages of cloud computing is resource sharing and scaling, it becomes imperative to consolidate multiple workloads for efficient hardware utilization. However, virtualization technologies do not offer performance isolation. Therefore, it is critical to understand how resource contention from consolidation impact application performance and to manage application QoS in the clouds. In this work, we propose an interference model where an influence matrix captures the dilation in terms of resource usage from the co-located applications by considering the impacts of all types of resources. As another distinctive feature, our approach takes into account the time variance in application resource usage. Therefore, the dynamics in application workloads factor in consolidation decision. We evaluate our models and consolidation algorithm using applications from a test suite and the SPECWeb2005 benchmark. The prediction error is less than 8% across all applications. We are able to optimize the provider's metric, which is to maximize the number of successfully executed applications, with negligible overhead.

References

[1] "Montage: An astronomical image engine." [Online]. Available: http://montage.ipac.caltech.edu/

[2] "Southern california earthquake center, community modeling environment (cme)." [Online]. Available: http://www.scec.org/cme

[3] W. M. P. V. D. Aalst, "Business process management demystified : A tutorial on models , systems and standards for workflow management," *Lectures on Concurrency and Petri Nets*, vol. 3098, pp. 1–65, 2004.

[4] S. J. Aarseth, *Gravitational N-body Simulations: Tools and Algorithms*. Cambridge, UK: Cambridge University Press, 2003.

[5] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis," *Workflows for eScience*, vol. 5, pp. 39–59, 2007.

[6] G. Casale, S. Kraft, and D. Krishnamurthy, "A model of storage i/o performance interference in virtualized systems," in *Proceedings of the 31st International Conference on Distributed Computing Systems Workshops(ICDCSW'11)*, Minneapolis, Minnesota, Jun. 2011, pp. 34–39.

[7] R. C. Chiang and H. H. Huang, "Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis(SC'11)*, Seattle, Washington, Nov. 2011, pp. 1–12.

[8] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing(SOCC'11)*, Cascais, Portugal, Oct. 2011.

[9] C. H. Hsu, S. C. Chen, C. C. Lee, H. Y. Chang, K. C. Lai, K. Li, and C. Rong, "Energy-aware task consolidation technique for cloud computing," in *Proceedings of the 3rd IEEE International Conference and Workshops on Cloud Computing Technology and Science(CloudComd'11)*, Athens, Greece, Nov. 2011, pp. 115–121.

[10] W. Hwang, Y. Roh, Y. Park, K. W. Park, and K. H. Park, "Hyperdealer: Reference-pattern-aware instant memory balancing for consolidated virtual machines," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing(CLOUD'10)*, Miami, Florida, Jul. 2010, pp. 426–434.

[11] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Proceedings of the IEEE International Symposium on In Performance Analysis of Systems & Software(ISPASS'07)*, San Jose, CA, Apr. 2007, pp. 200–209.

[12] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, "Performance analysis of network i/o workloads in virtualized data centers," *IEEE Transactions on Service Computing*, vol. 14, Jun. 2011.

[13] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems(EuroSys'10)*, Paris, France, Apr. 2010, pp. 237–250.

[14] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing(CLOUD'10)*, Miami, Florida, Jul. 2010, pp. 51–58.

[15] L. R. Rabiner, "A tutorial on hidden markov models and selected applications," 1989, pp. 257–286.

[16] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, pp. 195–197, Mar. 1981.

[17] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 USENIX Workshop on Power Aware Computing and Systems(HotPower'08)*, San Diego, CA, Dec. 2008.

[18] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, Dec. 2009.

[19] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *Proceedings of the 22nd Annual International Conference on Supercomputing(ICS'08)*, Island of Kos, Greece, Jun. 2008, pp. 175–184.

[20] A. Verma, G. Dasgupta, T. Kumar, N. Pradipta, and D. R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 conference on USENIX Annual technical conference (USENIX'09)*, San Diego, CA, Jun. 2009.

[21] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. F. Arlitt, and Z. Wang, "Probabilistic performance modeling of virtualized resource allocation," in *Proceedings of the 7th international conference on Autonomic computing(ICAC'10)*, Washington D.C., Jun. 2010, pp. 99–108.

[22] G. Welch and G. Bishop, "An introduction to the kalman filter," Department of Computer Science, University of North Carolina at Chapel Hill, Tech. Rep., Jul. 2006.