A Quantitative Analysis of Performance of Shared Service Systems with Multiple Resource Contention

Seung-Hwan Lim†, Jae-Seok Huh†, Young-Jae Kim†, Galen M. Shipman‡ and Chita R. Das†

†Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
{seulim, das}@cse.psu.edu

‡National Center for Computational Sciences
Computer Science and Mathematics
Oak Ridge National Laboratory
Oak Ridge, TN
{huhj,kimy1,gshipman}@ornl.gov

ABSTRACT
IT service providers employ server virtualization as a basic building block of their platforms to increase the cost effectiveness. The economic benefits from server virtualization come from higher resource utilization, reduced maintenance and operational costs including energy consumption. However, those benefits require efficient assignments of virtual servers or jobs to a limited number of physical hosts. The primary measures of evaluating efficiency of an assignment are that all the system resources are utilized effectively and the performance of each virtual machine is consistent with the desired performance bounds. While satisfying such SLA/performance bounds is essential for many classes of applications, the interference among the virtual machines makes such assurance admittedly difficult. This is primarily due to the complexity of estimating the completion time of jobs on assigned virtual servers.

In this paper, we present a mathematical model for estimating the processing time of jobs in shared virtual servers by treating it as a conventional job scheduling problem. Given a job set, its total processing time is the fundamental objective function for a class of optimization problems, commonly known as job scheduling problems. The classical model that assumes the total processing time as the sum of individual processing times, unfortunately does not capture the multiple-resource contention to process a job. A job needs several resources such as processing cores and memory on the physical platform, hence the resource contention does not exhibit the simple classical behavior. In this work, we establish a general quantitative model for the total processing time for machines with multiple resources. A job is characterized by vector valued loading statistics and the total processing time of a job set is given by a quadratic function of those loading vectors. We validate the model with actual measurements of applications running times on a Xen-based virtualized infrastructure. The experimental results indicate that the proposed model is effective in capturing the resource contention in a virtualized platform. The average error rate of the estimated running time with synthetic workloads is 10.7% for identical jobs and 6.0% for heterogeneous jobs, and is within 6% with realistic workloads.

1. INTRODUCTION
Server virtualization serves as a key technical component to realize server consolidation, reducing the total cost of ownership with reasonable performance degradation [19]. In order to overcome the under-utilization of production servers [9], IT service providers collocate multiple virtual servers on a physical host to share system resources such as processing units, memory, and I/O peripherals, thereby increasing the energy efficiency of the system. However, a major challenge of virtualization is in providing scalable, predictable and robust system performance due to high variance in performance of the consolidated virtual servers on physical hosts [2, 7]. This is primarily because of the contention for various shared physical resources by the applications running in virtualized environments.

The predictable performance provisioning problem in consolidated servers can be translated to the job scheduling problem. In the design of efficient job schedulers, a proper estimation of the performance variation due to the resource contention between jobs assigned on a host machine is of fundamental importance [24]. An inappropriate estimation can lead to unpredictability and high variance in the performance of systems as reported by IT service providers [2, 7]; they described the performance degradation of I/O-bound jobs highly unpredictable and exhibit high variance. Such a perception of unpredictability is actually the result of inappropriately limiting the concept of the workload of a job only to its CPU usage. Even with negligible CPU usages, I/O-bound jobs, obviously due to the I/O-resource contention, can experience large performance degradation. This is not acceptable for many classes of applications that need performance guarantees. The performance of scheduled jobs should be predictable with an appropriate estimator that can capture the multiple-resource contention problem in a virtualized environment and is the main motivation of this paper.

Job scheduling problem is an optimization problem in which the solution is the optimal assignment of jobs to the
given set of machines. More precisely, suppose we are given a set of machines, \( \mathcal{M} = \{1, \ldots, M\} \), and a set of jobs, \( \mathcal{J} = \{1, \ldots, N\} \). Consider a collection of disjoint subsets \( \{\mathcal{J}_1, \ldots, \mathcal{J}_M\} \) of \( \mathcal{J} \) such that \( \cup_{\mu=1}^{M} \mathcal{J}_\mu = \mathcal{J} \). The assignment of \( \mathcal{J}_\mu \) to the machine \( \mu \) for \( \mu = 1, \ldots, M \) determines the measure of size, \( T_\mu(\mathcal{J}_\mu) \), of \( \mathcal{J}_\mu \) running on the machine \( \mu \). Then, the total size associated to a specific assignment defined by

\[
T(\{\mathcal{J}_1, \ldots, \mathcal{J}_M\}) = \text{extremum}(T_1(\mathcal{J}_1), \ldots, T_M(\mathcal{J}_M))
\]  

becomes the objective function of the optimization problem.

The total processing time of \( \mathcal{J}_\mu \) on machine \( \mu \) has been the most important and traditional measure. In this case, the total measure \( T \) – with maximum in the place of the extremum – is called the makespan associated to the job assignment. In the classical setting, the processing time on the machine \( \mu \) for jobs in \( \mathcal{J}_\mu \) is assumed to be the sum of individual processing times

\[
T_\mu(\mathcal{J}_\mu) = \sum_{j \in \mathcal{J}_\mu} T_\mu(j),
\]

where \( T_\mu(j) \) the neutral – undisturbed by the presence of other jobs – processing time of job \( j \).

An online scheduling algorithm makes a decision for each newly arrived job. A comprehensive analysis of online scheduling problems for parallel machines can be found in \([20]\). We can also identify the scheduling problems with the classical linear processing time model described above as the bin packing problems \([5, 8]\) where the total size is the sum of individual sizes. The importance of a class of online scheduling algorithms is rapidly increasing in recent computing environments – online job scheduling with migration. With the renaissance of virtualization, the migration cost of a job already assigned and running on a machine to another machine has decreased dramatically, enabling more scalable, energy efficient, and load balanced systems \([14, 15, 22, 23]\). Development of well-performing online algorithms with job migration on virtualized environments is one of the eventual future goals of our research.

### 1.1 Motivation: deficiency of the classical model

As presented in the previous section, the estimation of the total processing time, \( T_\mu(\mathcal{J}_\mu) \), of an assigned job set on a machine forms the base of a scheduling algorithm and its performance. Recall, in the classical linear model, a job – or equivalently, a workload – can be entirely characterized by its neutral processing time, i.e. the processing time of the job under the absence of other jobs. The failure of this simple model in practical computing environments can be illustrated by considering the following situation.

Consider two types of jobs to be hosted in a cluster of two identical machines, namely type I and II. The jobs in type I and type II only requests CPU and I/O, respectively. Suppose the job set originally contained two jobs: one of type I with the processing time of 2 minutes and one of type II with the processing time of 3 minutes. Quite naturally, a scheduler will place each job on each machine in the 2-machine cluster, and the expected makespan is 3 minutes.

The classical estimate. Suppose, after 1 minute of execution arrived a new job: one of type I with 2 minutes of processing time. The classical estimation of the new makespan becomes 3 minutes (= max(1 + 2, 2)) with assigning the new job to the first machine and 4 minutes (= max(1, 2 + 2)) with the second machine. The optimal choice is the former with 3 minute of makespan. Notice, in this calculation, we exclude the 1 minute which has already passed.

**New estimate with multiple-resource contention.** Now, let us take the resource request pattern of each job-type into account. Let us also neglect all the overhead during initialization, exit, context switching, et cetera. On an idealized machine, a type I job and a type II job can coexist without causing any resource contention; the former requests only the CPU and the latter requests only the I/O of the same machine. For the above extreme types of jobs, the total processing time of two \( \tau \)-minute jobs of different types can quite reasonably assumed to be \( \tau \) minutes – not \( 2\tau \) minutes as estimated by the classical model.

Consider the two different assignments of the new job after 1 minute of execution: (1) If the new job of type I is assigned to the first machine which has a job of the same type in its job set, the total processing of the first machine is simply the sum of individual processing times, which is 3 minutes. The makespan of the cluster becomes 3 minutes (= max(1 + 2, 2)). (2) If the new one is assigned to the second machine on which a type II job is already running, the total processing time of the second machine becomes 2 minutes (= max(2, 1)). Hence, the makespan becomes 2 minutes (= max(1, max(2, 1))).

The difference, or enhancement, between the classical estimate and our new – quite qualitative at this point – estimate can be summarized as follows.

<table>
<thead>
<tr>
<th>model</th>
<th>optimal assignment</th>
<th>makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>classical</td>
<td>to machine 1</td>
<td>3 minutes</td>
</tr>
<tr>
<td>multiple-resource</td>
<td>to machine 2</td>
<td>2 minutes</td>
</tr>
</tbody>
</table>

The results illustrate that, by considering the multiple-resource contention, the optimal makespan can be reduced to 2/3 of what is expected by the classical simple model. In other words, the overall throughput of the cluster is improved 1.5 times by fully utilizing all the resources on the machine. A similar achievement of higher efficiency via workload multiplexing can be observed in an empirical study on large scale data centers \([9]\). Later in this paper, we will prove quantitatively the fact that the classical model assumes all the jobs are requesting the same resource – a special case of our general model.

A simple truth is that a better estimation results in better performance, which brings in the first fundamental question to be answered in this work:

**Question 1.** What is the general quantitative model for performance of systems with multiple-resource contention?

**Modeling of jobs and machines**

From the quite extreme example in the previous section, we can conclude that, for a better model, we have to consider the fact that an actual machine consists of multiple resources and jobs on it, in general, can request any of the resources \([6]\), as shown in Figure 1. Recall, in the classical model, a job can be entirely characterized by its processing
A job may contend for one of the resources with other jobs

<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>Disk I/O</th>
<th>Network I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 4</td>
<td>Job 2</td>
<td>Job 3</td>
<td>Job 1</td>
</tr>
</tbody>
</table>

A loading vector: A job can be described by the portion of time in accessing each resource.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>Disk I/O</th>
<th>Network I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>14%</td>
<td>5%</td>
<td>80%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Figure 1: We model the processing time of each job when multiple jobs are sharing multiple resources.

time, during which the job access a single resource only – in computing environments, CPU might be considered as the resource. For multiple resources, on the contrary, a job, or the workload done by the job, cannot simply be measured by the time duration between the start and the end of the job; we need more information on the request distribution of the job for each of the resources. Thus, we need to find the answer of the following question to answer the first one:

**Question 2.** How a job that requests multiple resources can be quantitatively modeled?

In this work, we modeled a job as a sequence of hypothetical **job slices**, each of which is devoted to a single resource. A job, i.e. the entire sequence of job slices, is statistically characterized by the probability of requesting/accessing each resource by a (random) single slice. The result is a vector-valued probability of resource-requests. Each job will be characterized by this **loading vector**.

Finally, in order to derive the formula for question 1, the behavior of a machine with multiple resources should be concretely defined.

**Question 3.** What is the quantitative model of a machine with multiple resources?

We model a machine as a collection of **resource-queues** – one queue for each shared resource. A slice of a job on a machine is assumed to be queued in one of the resource-queues with the probability given by the loading vector.

Multiple jobs on a machine will populate the resource-queues, hence, a job-slice can be slowed down due to the queue-waiting time. A quadratic function of loading vectors will be established to determine the dilation of jobs. Eventually, the resource contention by a job set on a machine will be described in terms of **dilation factors** – how each job is slowed down statistically under the presence of other jobs.

### 1.2 Our contributions

The contributions and ingenuities of this work can be summarized as follows.

- We present a quantitative model for the total processing time of jobs with multiple-resource contention for a virtualized data center/cloud computing platform. The model is more generic and can be applied to estimate the execution time of jobs in a shared environment. The formulation is developed by
  (i) modeling of a job as a sequence of job-slices with multiple-resource-access statistics,
  (ii) modeling of a machine as a collection of resource-queues,
  (iii) statistical characterization of the workload of a job in terms of the loading vector, and
  (iv) derivation of the dilation factor as a function of loading vectors.
- The dilation factor is proposed as the fundamental measure of performance degradation due to the multiple-resource contention so that it can be used as an objective function of the optimization problem; more precisely, we can prevent increasing in the total dilation factor to minimize the total performance degradation.
- The proposed model is validated by experiments on actual computing environments with both synthetic and realistic sample programs.

We provide experimental results from a Xen-based cluster using microbenchmarks and an application benchmark, FileBench [1]. The experimental results from microbenchmarks indicate that the average error rates of the estimations from the proposed model are 10.7% for identical jobs and 6.0% for heterogeneous jobs. However, the relative errors of the estimations from the classic processing time model are larger than from the proposed model, which are 21.5% for identical jobs and 31.9% for heterogeneous jobs. From actual measurements with the fileserver and mailserver workloads in FileBench benchmark, we show that the proposed model nicely estimates the expanded processing time of workloads within a 6.0% error rate.

**Road-map.** This paper is organized as follows. In Section 2, we present this work's theoretical features: (i) definitions and concepts such as jobs, machines, loading vectors, and dilation factors, (ii) the general formula for the dilation factor, and (iii) analysis of interesting special cases. Section 3 describes the experimental settings of this study. The validation results with both synthetic and realistic test samples are presented in Section 4, followed by the related work in Section 5. The discussions and summary of this work are presented in Section 6.

### 2. MATHEMATICAL MODELING

As illustrated by the simple example in Section 1, if jobs on a single machine compete over multiple shared resources, the dilation of their processing times can exhibit significantly complicated behavior. Recall, for a single-resource machine, time-sharing jobs slow down uniformly and proportionally to the number of competing jobs, which is not true anymore if they compete over multiple jobs; jobs can even have different dilation factors. As the first step to the clear description of the phenomenon, we begin with the modeling of machines with multiple resources.

#### 2.1 Modeling: machines and jobs

A machine with multiple resources can be viewed as a collection of multiple resource-queues; each resource corre-
sponds to a queue and requests from jobs running on the machine are queue on them. Let the machine has \( m \) resources, i.e. \( m \)-queues. We assume a resource-request from a job can be issued only after the previous request from the same job is completed. In order to describe this behavior clearly, we introduce the concept of the job-slice – a hypothetical atomic time during during which a job on the machine can request only a single resource usage. For convenience in the derivation, we denote the amount of time duration by \( \delta \). Thus, we assume a job consists of requests on resource-\( i \) \((i = 1, \ldots, m)\), and each request takes \( \delta \) independently of \( i \).

Remark 1. The hypothetical job-slice is the result of effort to unify the different measures of resources; in terms of the job-slices of size \( \delta \) in a unit of time, the amount of the work done by resource-\( i \) during a job execution can be measured by the number of the slices devoted to resource-\( i \). Hence, by the slice count, we can compare the amounts of work between different resources.

Remark 2. The value \( \delta \) is absolutely arbitrary and so is the count of the slices which depends on \( \delta \). Moreover, such a counting is not so feasible. Then, what is the use of the concept? A brief answer to the question is as follows: the number of job-slices for resource-\( i \) divided by the total number of job slices in a job is constant independently of the choice of \( \delta \). Thus, the amount of work done by resource-\( i \) from the request of a job is considered to be the amount of the time spent by resource-\( i \) divided by the total execution time of the job, which results in \( \delta \)-independent formulas.

Thus, a job is considered as a sequence of job-slices each of which requests a single resource. A request is viewed as the submission of the slice into one of the resource-queues. The job can proceed to the next slice only if the current slice in a queue is completed. Let there be \( n \) jobs running on the machine. We can immediately observe the following two facts: (1) there are at most \( n \) slices (including the one under the service) in any one of the \( m \) queues. (2) At any time, there are total \( n \) slices in the entire queues. The fact (1) implies that the time for a slice to be completed is at most \( n \delta \).

Remark 3. Notice that the actual amount of a unit of work, for example the amount (in MB) of file processed during \( \delta \) for the I/O-resource, depends on hardware and operating system – in general, the computing environment. It can even depend on the system configuration; for example, if the operating system utilizes I/O buffer cache, some of the I/O requests in the logical structure of a program turn effectively into memory requests. Thus, it is not feasible to estimate the multiple-resource workload by investigating the program source and hardware specification. A program’s workload will be obtained by regression analysis of its processing time, which will be illustrated in this paper.

Recall the only required information for the classical model – corresponding to single-resource machines in our point of view – is the neutral processing time of the job. In general multiple-resource model, additional information is required: the slice-resource-queue correspondence. The amount of information is impractically large if we want to obtain the complete map. Hence, we take a statistical approach.

### 2.2 The loading vector

Let us characterize the workload of a job with the number of slices corresponding to resource-\( i \) divided by the total number of slices, that is, the time for accessing resource-\( i \) divided by the total time without any other job on the same machine – we use the term neutral for such a situation. The non-negative value denoted by \( p_i \) can be interpreted as the probability of a job-slice getting queued into queue-\( i \). We define the loading vector of the job by the \( m \)-dimensional vector \( p \) with \( p_i \) as its \( i \)th element. Obviously,

\[
\|p\|_1 = \sum_{i=1}^{m} p_i \leq 1. \tag{3}
\]

Notice the inequality: we allow a job to have slices which does not request any resource. Such an idling slice simply spends time \( \delta \) without populating any of the \( m \) queues. We define two classes of jobs: a job is

1. idling if \( \sum_{i=1}^{m} p_i < 1 \)
2. busy if \( \sum_{i=1}^{m} p_i = 1 \).

Given a set of \( n \) jobs, denote by \( p_\bullet \) the loading vector of job-\( j \) and by \( p_{ij} \) the \( i \)th component of \( p_\bullet \). An \( m \) by \( n \) matrix can be formed with \( p_{ij} \) as its elements. We call this matrix the loading matrix of the job set. Each column vector of the loading matrix is the loading vector of the corresponding job. The loading vector and the loading matrix are our statistical characterizations of the workload of a job and a job set respectively.

### 2.3 The main theorem: the dilation factor

Consider \( n \) jobs on an \( m \)-resource machine characterized by a given \( m \) by \( n \) loading matrix. Suppose a slice of job-\( j \) is queued into queue-\( i \). Then, the neutral processing time \( \delta \) of the single slice will be dilated such that

\[
\delta \rightarrow \delta(1 + \text{the number of slices of other jobs in queue-}i) \]

where the first term in the parentheses is the service time of the slice under considerations and the second term is its waiting time. Hence, the conditional expectation of the diluted processing time of the single slice of job-\( j \), assuming it’s in queue-\( i \), is given by

\[
\delta \rightarrow \delta(1 + \sum_{k=1}^{n} p_{ik} \cdot \delta) \tag{4}
\]

Since \( p_{ij} \) is the probability of the queuing a slice of job-\( j \) into queue-\( i \), the expectation of the diluted processing time \( T_j \) of job-\( j \) is given by

\[
T_j = \tau_j \left( 1 + \sum_{i=1}^{m} p_{ij} \right) + \sum_{i=1}^{m} \tau_i \left( 1 + \sum_{k=1}^{n} p_{ik} \cdot \delta - p_{ij} \cdot \delta \right) \tag{5}
\]

where \( \tau_j \) is the neutral processing time of job-\( j \). Notice, the term \( (1 - \sum_{i=1}^{m} p_{ij}) \) represents the probability of idling which causes no dilation. Define the total loading vector \( \overline{p} \) by

\[
\overline{p} = \sum_{j=1}^{n} p_j \tag{6}
\]
The above relation can be rewritten in vector notation by
\[ T_j = \tau_j \left( 1 + \mathbf{p}_j \cdot \mathbf{\overline{p}} - \mathbf{p}_j \cdot \mathbf{p}_i \right) \] (7)
Thus, the job is slowed down by the factor of \( T_j/\tau_j \) due to the resource contention. We summarize the result by introducing the dilation factor \( \lambda_j = T_j/\tau_j \) as follows.

**Theorem 1.** Given a job set on a machine characterized by the loading vectors \( \mathbf{p}_j \) (\( j = 1, \ldots, n \)), the dilation factors \( \lambda_j = T_j/\tau_j \) (\( j = 1, \ldots, n \)) are given by
\[ \lambda_j = 1 + \mathbf{p}_j \cdot \mathbf{\overline{p}} - \mathbf{p}_j \cdot \mathbf{p}_i \] (8)
**Proof.** The derivation is given above. \( \square \)

Simple but very useful formula follows immediately when all the jobs are identical.

**Corollary 1.** If all of \( n \) jobs are identical \( (\mathbf{p}_j = \mathbf{p}) \), the dilation factors are identical \( (\lambda_j = \lambda) \) and are given by
\[ \lambda = 1 + (n - 1) \mathbf{p} \cdot \mathbf{p} \] (9)
**Proof.** Apply \( \mathbf{\overline{p}} = n \mathbf{p} \) to theorem 1. \( \square \)

The above corollary for identical jobs is utilized as the fundamental formula to obtain experimentally the loading vector of a job: (1) we obtain the loading factor by measuring the dilated time of \( n \) identical jobs for \( n = 1 \) to a sufficient number. (2) The data for various \( n \) can be analyzed by regression to obtain \( \mathbf{p} \).

The dilation factors are identical in another situation.

**Corollary 2.** If there are 2 jobs, the dilation factors are identical \( (\lambda_1 = \lambda_2 = \lambda) \) and are given by
\[ \lambda = 1 + \mathbf{p}_1 \cdot \mathbf{p}_2 \] (10)
**Proof.** Apply \( n = 2 \) and \( \mathbf{\overline{p}} = \mathbf{p}_1 + \mathbf{p}_2 \) to theorem 1. \( \square \)

### 2.4 Special case: 1-resource-busy jobs (the classical model)

In this section, we illustrate that the classical linear processing time model is actually a special case of our general model, where all the jobs compete over the same resource.

**Lemma 1.** Assume 1-resource-busy jobs: \( p_{kj} = 0 \) for any \( j = 1, \ldots, n \) and for any \( k = 1, \ldots, m \) except an index \( 1 \leq i \leq m \) and \( \| \mathbf{p}_j \|_1 = 1 \). Then, the dilation factors are identical \( (\lambda_j = \lambda) \) for all jobs and given by
\[ \lambda = n \] (11)
**Proof.** Since all the jobs are busy, \( p_{kj} = 1 \), hence, all jobs are identically given by \( \mathbf{p}_j = \mathbf{p} \) where \( \mathbf{p}_i = 1 \) and \( \mathbf{p}_i = 0 \) for any \( k \neq i \). Then, \( \mathbf{p} \cdot \mathbf{p} = 1 \) and, by the corollary in the previous section, the dilation factors are identical and given by
\[ \lambda = 1 + (n - 1) \mathbf{p} \cdot \mathbf{p} = 1 + (n - 1) = n \] (12)
\( \square \)

Let \( \tau_j \) denote the neutral processing time of job-\( j \) in a job set of size \( n \). We define the total processing time \( T \) of the job set as the time duration from the starting time of the first initiated job to the ending time of the last completed job. We claim that, for 1-resource-busy jobs, \( T = \sum_{j=1}^{n} \tau_j \) independently of the starting and ending time of each job.

**Theorem 2.** Suppose during the total processing time, there’s no idling gap, i.e. the machine is always populated by at least one job. Then, the total processing time is the sum of individual neutral processing times independently of the starting and ending times of the jobs. That is,
\[ T = \sum_{j=1}^{n} \tau_j \] (13)
**Proof.** For any overlapping of jobs, there is a corresponding refinement \( [t_k, t_{k+1}, \ldots, t_{m+1}] \) such that \( t_k = T \) and, in each subinterval \( [t_k, t_{k+1}] \), the number of running jobs denoted by \( n_k \) is constant. Let \( \Delta_k = t_k - t_{k-1} \). Let \( I_k \) denote the set of indices of active jobs in the \( k \)th subinterval and by \( I_k \) the set of indices of subintervals, where job-\( j \) is active. Let \( \tau^j_k \) is the completed amount of job-\( j \) in subinterval-\( k \), that is, in the subinterval, \( 100 \tau^j_k/\tau_j \)% of job-\( j \) is completed. (1) Since the dilation in each subinterval is identical for all active jobs, \( \Delta_k = n_k \tau^j_k \) for all \( j \in I_k \) for any \( k \). Adding this for all \( j \in I_k \), we obtain \( n_k \Delta_k = n_k \sum_{j \in I_k} \tau^j_k \), hence, \( \Delta_k = \sum_{j \in I_k} \tau^j_k \). (2) Then, \( T = \sum_{k=1}^{n} \Delta_k \) and, \( \Delta_k = \sum_{j \in I_k} \sum_{j \in I_k} \tau^j_k \). (3) By rearranging the indices, \( T = \sum_{j=1}^{n} \sum_{k \in I_j} \tau^j_k \). (4) Since all jobs should be completed, \( \sum_{j \in I_k} \tau^j_k = \tau_j \), which completes the proof. \( \square \)

**Remark 4.** In general cases, such a simple formula for total processing time does not exist. Consider a 2 job system with \( \mathbf{p}_1 = (1, 0) \) and \( \mathbf{p}_2 = (0, 1) \). Then, \( \lambda_1 = \lambda_2 = 0 \), that is, there’s no dilation of processing time. Let \( \tau_1 = \tau_2 = 1 \) minute. If the two jobs started at the same time, they will be completed at the same time after 1 minute. If one of them started first and the other job started at the time of completion of the first job, the total processing time will be 2 minutes. Depending on the overlapping, the total processing time can vary from 1 minute to 2 minutes. But, still the classical estimate becomes the upper bound of the total processing time in any case.

**Remark 5.** (1-resource-idling jobs) Even if there’s only one requested resource, the classical model cannot be applied to a job set with an idling job. Consider a simple job set of \( n \) identical jobs requesting only resource-i. Then, the loading vectors can be represented by a single scalar parameter \( p \). Then, \( \lambda = 1 + (n - 1) p^2 = (1 - p^2) 1 + p^2 n \), that is, \( \lambda \) is a linear interpolation of 1 and \( n \) with respect to \( p^2 \), since \( p < 1 \) and \( \lambda < n \).

### 2.5 Special case: 2-resource-busy jobs

The usefulness of this model comes from the fact that each loading vector \( \mathbf{p}_j \) can be represented by a single scalar parameter \( p_j \). Without loss of generality, we can take non-requested resources out of considerations and assume
\[ \mathbf{p}_j = (p_j, 1 - p_j) \] (14)
Then, \( \mathbf{\overline{p}} = (\mathbf{\overline{n}} - \mathbf{\overline{p}}) \) where \( \mathbf{\overline{p}} = \sum_{j=1}^{n} p_j \), and
\[ \lambda_j = 1 + p_j \mathbf{\overline{p}} + (1 - p_j)(n - \mathbf{\overline{p}}) - p_j^2 = (1 - p_j)^2 \]
\[ = 1 + n - \mathbf{\overline{p}} - n p_j + 2 p_j \mathbf{\overline{p}} - p_j^2 = (1 - p_j)^2 \] (15)
(16)
The result can be further simplified if the jobs are identi-
cal, i.e. $p_j = p$. Then, $\mathbf{p} = np$ and, for any $j = 1, \ldots, n$,
\[
\lambda_j = 1 + n - 2 np + 2 np^2 - p^2 - (1 - p)^2 \quad (17)
\]
\[
= 1 + n((1 - 2p + p^2) + (n - 1)p^2 - (1 - p)^2) \quad (18)
\]
\[
= 1 + (n - 1)(p^2 + (1 - p)^2) \quad (19)
\]

To emphasize the convenience of the formula, we summarize the result as a theorem.

**Theorem 3.** Assume 2-resource-busy identical jobs with the loading vector given by $(p, 1 - p)$. Then, the dilation factors are identically given by
\[
\lambda = 1 + (n - 1)(p^2 + (1 - p)^2) \quad (20)
\]

In other words, if the dilation factor is known, we can obtain the loading vector by the formula:
\[
p = \frac{1}{2} \left( 1 \pm \sqrt{1 - 2n \lambda \over n - 1} \right) \quad (21)
\]

**Proof.** The derivation is given above. \qed

The above relation suggests a simple experimental strategy to obtain the loading vector: (1) measure the neutral processing time $\tau$, (2) measure the dilated processing time $T$ via $n$-process experiments for some $n > 1$, (3) evaluate the dilation factor given by $\lambda = T/\tau$, then (4) evaluate the parameter $p$. Notice that there are two possible solutions for $p$ and the model does not distinguish them. But, we can choose the appropriate one utilizing system’s resource monitor.

### 2.6 The total dilation factor

We define the total dilation factor $\bar{\lambda}$ by the sum of $\lambda_j$ for all jobs. Then, by the formula given in theorem 1,
\[
\bar{\lambda} = \sum_{j=1}^{n} \lambda_j = n + \|\mathbf{p}\|_2^2 - \sum_{j=1}^{n} \|p_j\|_2^2 \quad (22)
\]

Notice, this simple formula involves only the $L_2$ norms of the loading vectors; informally the absolute values of the loading vectors. In order to present online version of the formula, assume a set jobs are already running on a machine with known characteristic parameters such as $p_j$, $\mathbf{p}$, $\lambda_j$, and $\bar{\lambda}$. Consider the following two cases:

1. A new job with the loading vector $\mathbf{p}'$ is added to the job set:
   \[
   \bar{\lambda}_+ = (n + 1) + \|\mathbf{p} + \mathbf{p}'\|_2^2 - \sum_{j=1}^{n} \|p_j\|_2^2 - \|\mathbf{p}'\|_2^2
   \]
   \[
   = \bar{\lambda} + 2 \mathbf{p}' \cdot \mathbf{p} \quad (23)
   \]

2. From the job set, a job-\(k\) is removed: identifying $\mathbf{p}' = \mathbf{p}_k$.
   \[
   \bar{\lambda}_- = (n - 1) + \|\mathbf{p} - \mathbf{p}'\|_2^2 - \sum_{j=1}^{n} \|p_j\|_2^2 + \|\mathbf{p}'\|_2^2
   \]
   \[
   = \bar{\lambda} - 1 - 2 \mathbf{p}' \cdot \mathbf{p} + 2 \|\mathbf{p}'\|_2^2 \quad (24)
   \]

Thus, the increment (decrement) is given by
\[
\delta \bar{\lambda}_\pm = \pm (1 + 2 \mathbf{p}' \cdot \mathbf{p}) + \begin{cases} 0 & \text{for +} \\ 2 \|\mathbf{p}'\|_2^2 & \text{for -} \end{cases} \quad (25)
\]

**Figure 2:** An overview of our experimental environment.

Notice, the decrementation formula (with $-$ subscript) is used only to reset the current state when a job is completed, hence, removed from the job set. The incremental formula (with $+$ subscript) is more crucial in job scheduling algorithms. Recall remark 4; a simple formula for the total processing time as in the classical model does not exist anymore for the general multiple-resource model. Even though it is technically possible to compute the total processing time from the thorough job overlapping information, the total processing time might not be that useful/convenient criteria for job scheduling.

In this paper, we suggest a new objective function for the optimization problem. Notice that the total dilation can be viewed as a measure of performance degradation. Hence, the minimization of the performance degradation, i.e. the increase in the total dilation factor can be considered as an alternative way to utilize available resources optimally. If we accept this new criteria, the optimization problem can be stated as follows:

*Given a new job with the loading vector $\mathbf{p}'$, find the assignment to a machine $\mu$ which minimizes $\mathbf{p}' \cdot \mathbf{p}_\mu$.*

### 3. EXPERIMENTAL ENVIRONMENT

This section describes the experimental settings designed to validate the performance model described in section 2. These experimental results indicate three points.

- We can estimate the completion time of jobs more accurately than the classical processing time model.
- With the dilation factor, $\lambda$, we can perform fair evaluation of completion times of candidate schedules of jobs.
- Adopting our processing time model in scheduling algorithms leads to scalable and predictable performance.

#### 3.1 Target system overview

The overview of our experimental environment is depicted in Figure 2. A virtual machine is an independent operating system instance in virtualized environments. Thus, applications running on a virtual machine have an illusion that
they exclusively access the virtualized resources. A special
guest machine, Dom-0, can directly access the physical re-
sources, especially I/O devices. The hypervisor manages
shared resources such as processors, memory, I/O subsystems
and network devices in order to provide fair performance
for each virtual machine instance. All the access requests
to the hardware resources from applications running
on a virtual machine flow into the hypervisor and then Dom-
0, if necessary. The resource contention among collocated
virtual machines similar to the processes in non-virtualized
environments [3].

We experimented with the hardware platform of four dedi-
cated cluster nodes running on the Linux 2.6.18 kernel. Each
physical host machine is equipped with two 64-bit AMD
Opteron 250 processors and 4GB Memory, connected to
both the 10Gbps SDR infiniband and 1Gbps Ethernet. We
installed Xen-3.4.2 on each physical machine. Each guest
dedicated virtual machine runs on the Linux 2.6.18 kernel. The virtual
hard disks of guest virtual machines are located in a Net-
work File System (NFS) partition, thus I/O requests from
guest machines may invoke network traffic. Thus, we care-
fully monitored that the network was not fully saturated in
all the experiments. In addition, 512 MB of RAM is assigned
to each guest virtual machine. The reserved memory for the
Domain-0 increases as the number of hosted guest virtual
machines increases since the size of memory of Domain-0 af-
fects the performance of the I/O operations from guest do-
mains due to the buffer cache hit. All the guest virtual ma-
chines are configured to use the same CPU, while Domain-0
can use any of two CPUs in our platform.

3.2 Workloads

We employed one application per virtual machine as work-
loads. We constructed a microbenchmark to control the
loads on CPU, I/O and memory for generating synthetic
workloads. For the CPU workload, a total of 700 Mbytes of
random numbers are generated. I/O workloads are gener-
at by a mixture of operations of both read and write for
70 files without utilizing buffer cache of a filesystem, where
the size of each file is 10 Mbytes. In order to create memory
workloads, we used the I/O workloads with accessing buffer
cache of a filesystem. As realistic workloads, we employed
pre-defined configuration of fileserver and varmail workloads
from a local storage system benchmark, FileBench [1].

Metrics. We compare the total processing time from the
proposed model and classical model with actual measure-
ments. The total processing time from each model is given by

\[ T_{total,classic} = \sum_{i=0}^{n} T_{t1} \]  

(27)
\[ T_{estimation} = \lambda_j \times T_{t1} \]  

(28)
\[ T_{total,estimation} = \max(\lambda_j \times T_{t1}) \]  

(29)

where \( i \) represents the jobs placed on a physical machine and
\( T_{t1} \) is the running time of job \( i \) when it is the only job in a
machine.

What our model computes is the dilatation factor, \( \lambda \) for each
job. For job \( j \), the corresponding \( \lambda_j \) is given by the formula:

\[ \lambda_j = 1 + p^j \cdot \bar{p} - p^j \cdot p^j, \]  

(30)

where \( p^j \) is the \( j \)th column vector of the loading matrix and

<table>
<thead>
<tr>
<th>Workload ID</th>
<th>characteristics</th>
<th>loading vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>I/O bound</td>
<td>(1.0,0.9)</td>
</tr>
<tr>
<td>W2</td>
<td>I/O-CPU bound</td>
<td>(0.8,0.2)</td>
</tr>
<tr>
<td>W3</td>
<td>I/O-mem bound</td>
<td>(0.1,0.9)</td>
</tr>
<tr>
<td>W4</td>
<td>CPU bound</td>
<td>(1.0,0.0)</td>
</tr>
</tbody>
</table>

Figure 3: The formulated model in Section 2 predicts the
processing time of identical jobs better than the classic
model. (The numbers in parentheses denote the relative
errors from our model estimation and classic approach)

\[ \bar{p} = \sum_{j=1,2} p^j \] in two-resource-busy model.

In all the following figures, \( e \) represents the relative error
of the estimation given by

\[ e = \frac{|\text{measure} - \text{estimation}|}{\text{measure}} \]  

(31)

4. MODEL VALIDATION

In this section, we validate the model with synthetic and
realistic workloads.

4.1 Validation with synthetic workloads

4.1.1 Identical jobs

We demonstrate with identical jobs for the validation of
the proposed multi-resource processing time model is scal-
able. The experimental results concerns multiple identical
jobs in order to validate the proposed multi-resource pro-
cessing time model is scalable. Table 1 provides the resource
loading vectors of four workloads used to compute the esti-
mated values in the experiments. We lay out the profiles of the
workloads used in these experiments in Table 2. In the
experiments, we repeated 100 runs to obtain consistent
results to verify our model. Figure 3 indicates that the pro-
posed process time model nicely estimates the average run-
ing time of collocated jobs. The relative error is 10.7% on
average and the maximum is 22.6%. We confirm that the
accuracy of the estimation from our model is better than
or equal to the classical linear processing time model that
is simply the linear sum of individual processing times. We
find that the estimated values from two-resource-busy model
show increasing relative error with W3, which is an I/O and memory bound workload. We believe that adopting threeresource-busy model will advance the accuracy in estimating the processing time of W3 since in all the estimations, we only assumed CPU and I/O as the resources to be shared.

### 4.1.2 Heterogeneous jobs

Next, we show the proposed model can estimate even when different types of jobs are being scheduled in a machine.

Let us take two jobs from each of W1 and W2 in the profile shown in Table 2. The conventional processing time model estimates the total processing time of W1 and W2 as 67+160 = 227 seconds. However, using our processing time model, $\nu = (0.1 + 0.8, 0.9 + 0.2) = (0.9, 1.1), \mu_1 = (0.1, 0.9), \mu_2 = (0.8, 0.2)$ from the definitions of the total loading vector and loading vector. Hence, the total dilation factor $\lambda_T$ for W1 and $\lambda_T$ for W2 is given by,

$\lambda_T = 1 + (0.1 \cdot 0.9 + 0.9 \cdot 1.1) - (0.1 \cdot 0.1 + 0.9 \cdot 0.9) = 1.26$

$\lambda_T = 1 + (0.8 \cdot 0.9 + 0.2 \cdot 1.1) - (0.8 \cdot 0.8 + 0.2 \cdot 0.2) = 1.26$

Actually, we can show that, for two jobs, $\lambda_T = \lambda_T$ always. The obtained values mean that the processing time will be slowed down by the factor of 1.26, therefore,

$T_1 : 67 \rightarrow 67 \times 1.26 = 84.42$

$T_2 : 160 \rightarrow 160 \times 1.26 = 201.60$

which results in $\max(84.42, 201.60) = 201.60 \ s < 227 \ s$ of the classical linear model. Notice, we assumed that those jobs are repeating, thus, the loading factors remained the same. If we assume single units of jobs - 67 s and 160 s - started to run at the same time. Job 1 will end after 84.42 s. After job 1 ends at 84.42 s, our active job set will contain job 2 only. During the first 2-job session, job 2 will be slowed down such that (84.42/201.60) = 41.875% of the job is done. For the remaining 58.125% of the job, the job will run at its neutral speed. Thus, the processing time of the job 2 will be given by

$T_2 = 84.42 + 0.58125 \times 160 = 177.4 \ s$

Hence, the total processing time will be 177.42 s (job 1 ends earlier).

We can prove that the classical model results in the same total processing time (67+166) s independent of the overlapping jobs. Unlike the classical model, our multi-resource model predicts different total processing times depending on the starting and ending sequence of the jobs. This indicates the total processing time might not be a proper measure of performance.

Figure 4 confirms that the proposed model estimates the running time of each workload when W1 and W2 are multiplied and the relative errors of the estimated values are 15.8% and 2.4%. As we increase the number of hosted virtual machines, the relative error is still within 20%. The average relative error is 6.1% and the maximum is 9.94%. We confirm that the proposed model can predict the total job length more accurately than the classical model. The deviation between the classic processing time model and actual performance tends to increase as the number of colocated jobs increases from two to four. This will create challenges in finding optimal job assignments in actual systems with more virtual machines.

### Impact on scheduling algorithms

Next, we discuss the impact of the proposed model on a scheduling algorithm, the list scheduler [12], which is widely adopted in the industry [13] and shows good theoretical performance bounds [18]. As Figure 5 shows, let us assume that we have two jobs of W1 and one of W2 at time t. Let us denote them as W1, W1, W2, W2. Then, using the conventional model, we may schedule

Machine 1 : W1, W1, W2 (67 + 67 = 134)

Machine 2 : W2 (160)

The max length of the schedule is 160 in M2. If a new job W3 arrives at time t, list scheduler (a greedy algorithm proposed in [12]) will allocate the job to M1 since it will be idle before W2 in M2 is completed. If W1 is assigned to M1, the max length of the schedule would be 134+67 = 201 s.

In the proposed model, however, the loading matrix for the job set, \{W1, W1, W2, W2\} is given by

$$
\begin{bmatrix}
0.1 & 0.1 & 0.8 \\
0.9 & 0.9 & 0.2
\end{bmatrix}
$$

<table>
<thead>
<tr>
<th>Workload</th>
<th>running time (s)</th>
<th>CPU util (%)</th>
<th>memory util (%)</th>
<th>Block I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 (I/O bound)</td>
<td>67.07</td>
<td>5.39</td>
<td>0.94</td>
<td>Dom-0</td>
</tr>
<tr>
<td>W2 (CPU bound)</td>
<td>160.03</td>
<td>2.50</td>
<td>78.93</td>
<td>Dom-0</td>
</tr>
<tr>
<td>W3 (I/O-mem bound)</td>
<td>67.87</td>
<td>2.56</td>
<td>87.97</td>
<td>Dom-0</td>
</tr>
<tr>
<td>W4 (CPU bound)</td>
<td>241.26</td>
<td>0.1</td>
<td>99.9</td>
<td>Dom-0</td>
</tr>
</tbody>
</table>
We expect

\[ \lambda_1 = 1 + (0.8, 0.2) \cdot (0.9, 1.1) - (0.8, 0.2) \cdot (0.8, 0.2) = 1.26 \]

\[ \lambda_2 = 1 + (0.1, 0.9) \cdot (0.9, 1.1) - (0.1, 0.9) \cdot (0.1, 0.9) = 1.26 \]

Hence, \( \lambda_{new} = 2 \times 1.82 + 2 \times 1.26 = 6.16 \).

To summarize, if we add the job \( W_{13} \)

1. to Machine 1: \( x \) increases 3.64 \( \rightarrow \) 8.92

2. to Machine 2: \( x \) increases 3.64 \( \rightarrow \) 6.16

Thus, we may schedule \( W_{13} \) to machine 2 since it results in less total dilation. Assigning \( W_{13} \) to machine 1 allows to schedule the next job \( W_{14} \) to machine 1 at \( t + 120.6 \). Thus, the total job length becomes 187.6 s, compared with 211.02 s when \( W_{13} \) is assigned to machine 1 as to the estimated total processing time.

Note that this example assumes the same scheduling mechanism for both cases, but only evaluating a schedule by the total dilation leads to more efficient schedule. In short, the proposed processing time model is capable of offering a fair measure to evaluate candidate schedules, which can impact on the effectiveness of a system.

### 4.2 Validation with realistic workloads

This section presents the validations with realistic workloads. We employed fileserver and varmail workloads from FileBench [1] for the analysis. The resource loading vectors of both workloads are shown in Table 3. In Table 4, we present the average resource usages when we run only one virtual machine of the fileserver workload or the mailserver workload. In order to obtain the average resource usages

<table>
<thead>
<tr>
<th>Workload</th>
<th>loading vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fileserver</td>
<td>(0.02, 0.98)</td>
</tr>
<tr>
<td>Mailserver</td>
<td>(0.10, 0.90)</td>
</tr>
</tbody>
</table>

![Figure 5: The ability to estimate the accurate processing time](image)

![Figure 6: Comparison of processing time for different combinations of realistic workloads](image)
Table 4: FileBench Workload Profiles.

<table>
<thead>
<tr>
<th>Workload</th>
<th>running time (s)</th>
<th>CPU util (%)</th>
<th>memory util (%)</th>
<th>Block I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fileserver (I/O bound)</td>
<td>378.15</td>
<td>0.89</td>
<td>1.03</td>
<td>22.97</td>
</tr>
<tr>
<td>Mailserver (CPU bound)</td>
<td>43.43</td>
<td>0.07</td>
<td>89.49</td>
<td>3.03</td>
</tr>
</tbody>
</table>

and resource loading vectors, we repeated 40 runs for each workload and confirmed that the average running time is consistent.

Let us estimate the running time of each workload when we multiplex fileservers and mailserver workloads into one physical machine. Given the loading matrix as in Table 3, the dilation factors for the cases of assigning two workloads are as follows

\[ \bar{\lambda} = 1.88 \quad \text{one fileserver with one mailserver} \]
\[ \bar{\lambda} = 1.96 \quad \text{two fileservers} \]
\[ \bar{\lambda} = 1.81 \quad \text{two mailservers} \]

In Figure 6, the experimental results confirm that the estimation from the model is close to the actual measurements. The total dilation becomes 3.76 with the mixture of one fileserver and one mailserver, 3.92 with two fileservers, and 3.62 with two mailservers. Mixing one mailserver and one fileserver results in a total processing time of 433.08s in measurements, compared with 458.1s in estimation. We estimate the running time of the mailserver workload to be 87.62s, which shows only 0.65% error rate with actual measurements.

Let us now obtain \( \lambda \) for more number of multiplexed workloads. \( \bar{\lambda} \) represents the total dilation factor for one fileserver and two mailservers and \( \bar{\lambda} \) is for two fileservers and one mailserver.

\[ \bar{\lambda}_1 = \lambda_{file} + 2 \cdot \lambda_{mail} = 2.8 + 2 \cdot 2.7 = 8.2 \]
\[ \bar{\lambda}_2 = 2 \cdot \lambda_{file} + \lambda_{mail} = 2 \cdot 2.8 + 2.7 = 8.3 \]

In Figure 6, the experimental results show that the average running time of the mailserver workload is 133.48 s, compared with 125.6 s from the estimation. For the fileserver workload, the measurement is 498.3 s and the estimation is 497.2 s. The experimental results from FileBench also show that the error rates of the estimation from the proposed model is small.

When we analyze the resource utilization, the mailserver workload is CPU bound and the fileserver workload is I/O bound. Then, a scheme based on the analysis of resource utilization may decide to multiplex those two workloads in a physical machine without significant performance degradation. However, the loading matrix in Table 3 shows the two workloads have similar distribution in accessing resources in a system. Thus, our processing time model can predict the significant performance degradation by assigning the fileserver and mailserver workloads to a physical host. This result suggests that we can lower the unpredictability in I/O bound jobs in cloud services [2] with an assignment scheme based upon the proposed processing time model. In sum, we demonstrate that the proposed model provides a quantitative measure for the performance degradation with multiplexed workloads on virtualized platform.

5. RELATED WORK

The work related to resource provisioning and virtual machine placement is summarized in this section.

Since job behavior depends on the allocated resources [6], a variety of resource provisioning methods have been explored in prior work [6, 10, 14, 17, 25, 27]. Resource provisioning addresses predicting required resources for each job [6, 17, 27] and avoiding the contention among jobs [10, 14]. In [17], the authors designed a dynamic virtualized resource management scheme based on control theory. Since the size of memory affects the I/O performance, managing memory resources among virtual machines has been discussed in [27]. By managing the contention for shared memory, the performance can be improved [10]. The authors in [14] provided a scheme that enhances the scalability and performance by avoiding possible CPU contention among highly correlated virtual machines. Since power or energy is an important system resource, provisioning power or energy of systems is also of interest in the community [4, 11]. In short, resource provisioning discusses allocating resources among workloads and may not guarantee or predict performance of hosting machines.

Placing virtual machines on a set of physical machines has been addressed in recent prior work [15, 21–23]. The authors in [21] showed that their proposed VectorDot algorithm can be an index of performance degradation. The algorithm computes the inner products of resource usage vectors associated with the virtual machines. When we have offline profile of workloads for each virtual machine, avoiding placing virtual machines that have similar workload pattern improves the energy efficiency and provided more stable performance [22]. A dynamic scheduling of virtual machines in data centers is considered in [23] to reduce power consumption of data centers. Including the above work, most of the virtual machine placement studies are conducted based upon the CPU demand. A recent study, however, found that placing virtual machines according to network usage, instead of CPU demand, enhances the scalability of a data center network [15]. A common trend in the virtual machine placement is that most of the prior work considered guaranteed resource usage of each virtual machines rather than providing performance bounds.

Unlike the prior work, we provide a model that predicts the performance of virtual machines. Since the performance of a system is proportional to the resource usage until the usage reaches a threshold [16], a convention in placing virtual machines is to place them according to the resource usage until the threshold values. However, with mixed workloads in data centers, we can achieve more graceful performance degradation [9, 14]. Therefore, we propose a performance model that provides the performance impact when we host a combination of virtual machines or jobs in a server. In addition, the issue raised from I/O-bound work in [2, 26] implies that we need to consider the importance of multiple resource requests in processing a job. Thus, we deem that a
job contends for multiple shared resources with other jobs.

The closest problem setting to our model is the bin packing problem, specifically the multi-bin packing problem. Bin packing problem finds a solution to pack items into single or multiple bins so as to minimize the number of bins used \[5\]. In the bin packing problem, the total size of packed items is the linear sum of the sizes of individual items. However, this study mentions that obtaining the actual size of all the packed items might be smaller than the total size of each item. In the multi-bin packing, we model bins and the sizes of items as vectors, but it does not capture the contention when we place items to bins. Our model explains a system with multiple resources and the contention over those multiple resources. With the model in this study, service providers can set a performance bounds that they can offer and create more scalable and predictable facilities regardless of workload types.

6. CONCLUDING REMARKS

This section explains how to exploit this model in existing scheduling algorithms and possible domains that can utilize this model, together with the summary of this study.

6.1 Discussion and future work

In designing or evaluating a scheduling algorithm, the fundamental operation to estimate its performance is to calculate the overall processing times of the assigned jobs. We can use the proposed model to estimate this processing time more accurately than the previous models. The required overhead is that we need to find the loading vector of a job to find the expected total processing time. As we described in section 2, we can identify the loading vectors by comparing the total processing time of multiple instances of identical jobs with the neutral processing time of the job. The current processing time model might not fit to the situation that multiple homogeneous resources are provided to mitigate the resource contention. One of the assumptions of this study is that the resource access patterns of each job are independent. Otherwise, we cannot easily calculate the probability of the resource contention by the inner product of loading vectors. Thus, we plan to expand our model to capture the situation when jobs are correlated.

Since sharing multiple resources among jobs is a common practice in computing systems, we can find a plenty of applications for the proposed model. Some of possible applications are as follows: Estimating the total processing time of jobs when we have heterogeneous processors such as CPU and GPU might be possible. Since CPU and GPU are different types of resources to be shared among jobs and the total processing time of a job depends on the portion of accessing times of each processor. Similarly, when we have multiple layers of caches and want to find the total accessing time of a job, we can apply our model. Even, when we have multiple tiers of systems such as multi-tier data centers or multi-tiered storage systems, the proposed model can be employed to estimate the completion time of each application.

6.2 Summary

In this study, we derived a novel processing time model of virtual machines. To the best of our knowledge, no proper work has proposed a model to capture the contention for physical resources between co-running jobs on virtual machines. We estimate the processing time of workloads on virtual machines by considering that multiple shared resources may be involved to process a job. This resource requirement is represented by a loading vector, which in turn is used to estimate the dilution in individual job execution times. We validate the proposed with experiments using synthetic and real workloads. From the validation results using multiple instances of identical synthetic workloads, the average relative error is 10.7% and for the heterogeneous workloads, the average relative error is 6.1% and the maximum error is 9.94%. With realistic benchmarks, our model also predicts the processing time of workloads when we multiplex them within a 6% error rate. The relative error of our prediction is lower than the classical processing time model. We also demonstrated how the proposed model can be used in estimating the overall execution time of a scheduling algorithm with dynamically arriving workloads. We believe such a model will be more valuable in satisfying the SLAs in evolving virtualized data centers and cloud computing platforms.

7. REFERENCES

[10] Alexandra Fedorova, Sergey Blagodurov, and Sergey Zhuravlev. Managing contention for shared resources...


