D-Factor: A Quantitative Model of Application Slow-Down in Multi-Resource Shared Systems

[1]Seung-Hwan Lim*, [2]Jae-Seok Huh, [2]Youngjae Kim, [2]Galen M. Shipman, and [1]Chita R. Das

[1]The Pennsylvania State University
University Park, PA 16802
{seulim, das}@cse.psu.edu

[2]Oak Ridge National Laboratory
Oak Ridge, TN 37831
{huhj, kimy1, gshipman}@ornl.gov

ABSTRACT

Scheduling multiple jobs onto a platform enhances system utilization by sharing resources. The benefits from higher resource utilization include reduced cost to construct, operate, and maintain a system, which often include energy consumption. Maximizing these benefits comes at a price – resource contention among jobs increases job completion time. In this paper, we analyze slow-downs of jobs due to contention for multiple resources in a system; referred to as *dilation factor*. We observe that multiple-resource contention creates non-linear dilation factors of jobs. From this observation, we establish a general quantitative model for dilation factors of jobs in multi-resource systems. A job is characterized by a vector-valued loading statistics and dilation factors of a job set are given by a quadratic function of their loading vectors. We demonstrate how to systematically characterize a job, maintain the data structure to calculate the dilation factor (loading matrix), and calculate the dilation factor of each job. We validate the accuracy of the model with multiple processes running on a native Linux server, virtualized servers, and with multiple MapReduce workloads co-scheduled in a cluster. Evaluation with measured data shows that the D-factor model has an error margin of less than 16%. We also show that the model can be integrated with an existing on-line scheduler to minimize the makespan of workloads.

Categories and Subject Descriptors

C.4 [**Computer System Organization**]: Performance of systems—*modeling techniques, performance attributes*

General Terms

Management, Performance

*Seung-Hwan Lim is currently affiliated with Oak Ridge National Laboratory.

SIGMETRICS'12, June 11–15, 2012, London, England, UK.

Keywords

Shared resource management; Cloud computing; Performance modeling; Application running time

1. INTRODUCTION

Sharing a system with multiple workloads enhances system utilization, thereby lowering the economic [10] and environmental dent [33]. Recent advents in server virtualization bolster this sharing policy since it allows multiple operating system instances to share a physical machine [4]. However, sharing a system comes at a price – contention for system resources. Contention for system resources leads to high variance in performance, which would deter hosting latency sensitive applications that require tight performance guarantees [3]. The high variance in performance is often considered as unpredictable performance [3], due to the deficient understanding of performance variation of a workload from contention for multiple resources [13, 17]. This in turn would make the usual, but costly over-provisioning design more attractive, thereby defeating the whole concept of resource consolidation and cloud computing. Therefore, performance quantification of jobs (or performance degradation) in a multi-resource shared platform is essential to both facilitate co-hosting of applications and meet the service requirements. However, performance prediction in such an environment is admittedly a challenging problem and to our knowledge, no efficient technique is available today.

We may capture the performance variation of jobs in shared systems with slow-down due to resource contention. Let *dilation factor* denote the slow-down of a job due to resource contention. Prior efforts to estimate dilation factor falls into two groups – modeling based solutions [21, 24] or measurements based solutions [13, 16, 18, 20]. Modeling based solutions often use resource usage statistics of each job to construct the model. For example, queuing theory based approaches use system parameters like job arrival rates and service rates of resources [21], while control theory based approaches use the relationship between allocated amount of resources and workload behavior [24]. However, resource access behavior of a job depends on co-located jobs and specific hardwares, especially when both jobs and machines are heterogeneous [17,28,35]. For instance, co-located jobs can alter cache hit ratio of a job in multi-core environments [17, 27] or disk access latencies in shared storage systems [35], which can change resource access statistics of a job. Therefore, in order to improve the accuracy of a model, prior modeling

based approaches require detailed resource access behavior of each job.

In order to address such challenges, measurement based approaches have been proposed [13,16,18,20]. Koh *et al.* [18] predicted the performance degradation of co-located workloads using recorded similar workloads in terms of their resource usage statistics, similar to program similarity analysis [16]. Recently, Govindan *et al.* [13] and Mars *et al.* [20] independently proposed techniques that employ probe jobs to infer behavior of workloads when they are co-located, which does not require resource usage statistics. However, those approaches do not provide generic strategies to estimate dilation factors of co-located jobs. Therefore, we desire to develop a simple, but generic model to estimate dilation factors of co-located jobs, without depending on detailed description on research access behavior of each job.

In order to develop such a model, we may start from a simple model, linear sum of original completion times of jobs, which has been used to obtain makespan of jobs in scheduling algorithms [30]. With linear sum, we may obtain dilation factor of each job from dividing the makespan by individual completion time of each job. The benefit of the linear sum model is that it only relies on the completion times of jobs. However, linear sum may not be able to provide desired accuracy due to non-linearity between individual completion times and makespan in multi-resource shared environments [5,11,17]. For example, completing two co-hosted jobs – one 100 *sec* CPU-job with one 100 *sec* I/O-job – will take less than 200 *sec*, contradict to estimation from the linear sum of completion times of both jobs. The deficiency of linear sum model is the lack of ability to consider multiple resources at the same time. Thus, we propose a model that extends the linear sum model to consider multiple shared resources, which only relies on the completion times of jobs instead of resource usage statistics. With the proposed model, we overcome challenges in using prior analytical model based approaches to estimate dilation factors of co-located jobs. Also, we provide a mathematical foundation on using well-known workloads, probe jobs in [13, 20], to infer behavior of co-located workloads.

To provide accurate estimation of slow-down of jobs due to resource contention in shared environments like data centers/cloud computing platforms, we propose a generic quantitative model for dilation factors of n-jobs contending for m-resources, called the *dilation factor model* (*D-factor model*). We view *a job* as a sequence of accessing one of the system resources and a shared system as a collection of resources. For this, we characterize the resource access statistics of a job by a vector-valued loading statistics, *a loading vector*. We obtain the factor of dilated completion time, the *dilation factor*, of each job from a quadratic relation of loading vectors for jobs in the system, which can be utilized in optimization problems such as job scheduling. In addition, the proposed D-factor model can profile a job by only measuring completion time of a job, which does not require any instrumentations such as monitoring tools and modifying software/hardware infrastructures as have been used in prior studies [13]. This is an important characteristic for cloud environment, where application operators on virtual machines often cannot monitor physical resources.

We validated the proposed model with actual measurements on cluster infrastructures running native Linux and Xen Hypervisor [4], with synthetic workloads, FileBench [2], and MapReduce. The experimental results from synthetic workloads indicate that the average error rates of the estimations from the proposed model are 7% for a system on native Linux and 10.3% for a virtualized system. However, the estimation error from the linear completion time model could be up to 98% in the worst case. For the FileBench benchmark, the proposed model estimates the completion time of workloads within a 6.0% error. We used three popular applications to benchmark MapReduce– `Sort`, `Grep`, and `PiEstimator`. In experiments with MapReduce, we increased the number of instances of each MapReduce application up to four and co-hosted three different MapReduce applications. The margins of errors are about 16% for identical MapReduce applications and 11% for heterogeneous MapReduce applications. We also demonstrate how the D-factor model can extend a scheduler for single-resource systems into a scheduler for multi-resource systems for reducing the makespan of workloads by about 20%.

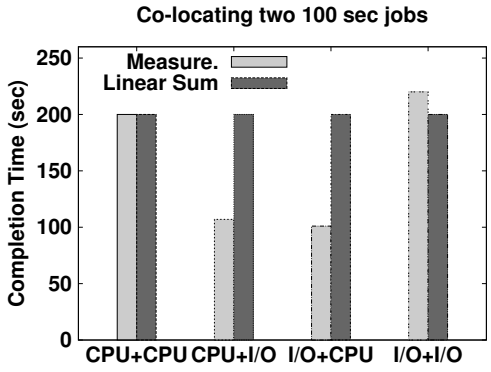


Figure 1: Non-linearity in slow-down of a job in a multi-resource system. CPU represents a CPU-bound job and IO represents an I/O-bound job. Each bar represents the completion time of job A when job B is colocated. For example, CPU+IO denotes the completion time of the CPU-bound job when the I/O-bound job is co-located.

This paper is organized as follows: Section 2 provides the motivation of this work. In Section 3, we present the underlying theoretical concepts for developing the D-factor model. Section 4 describes the experimental settings of this study. The validation results with both synthetic and realistic test samples are presented in Section 5, followed by an example application to a scheduling algorithm in Section 6. Related work is presented in Section 7, followed by concluding remarks in Section 8.

2. MOTIVATION

Interference due to sharing resources results in slow-down of workloads [9], depending on the characteristics of co-located workloads [13, 17]. Prior findings have suggested non-linear dilation factors when we consider multiple system resources; co-located applications on multi-core processors [5, 11], co-located processes in an operating system [9], co-located virtual machines in a physical machine [6], and co-located workloads in a data center [10].

Figure 1 highlights the non-linearity in slow-downs of workloads due to multiple resource contention. Here, we created two different types of jobs – a CPU-bound job, which consists of arithmetic operations and an I/O-bound job, which randomly reads two 2GB files. Both jobs take 100*sec* without the presence of other jobs. Experiments are done in a

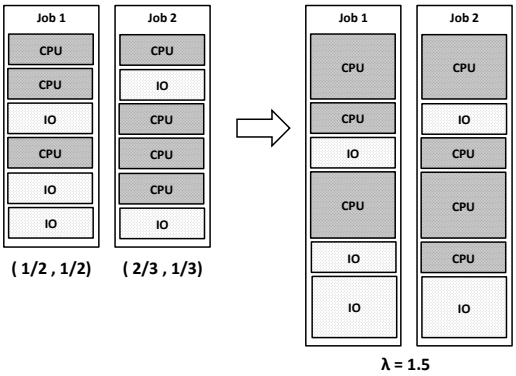


Figure 2: Dilation factor model describes slow-down of each job when multiple jobs are contending for multiple resources. (Left) Job-slices in their neutral states, where loading vectors are (1/2, 1/2) and (2/3, 1/3), respectively for CPU and I/O. (Right) The processing times of job slices will be dilated when they request the same resource at the same time. As described in Property 2 in Section 3, when two jobs are competing, their dilation factors, λ, are identical.

machine with two single-core CPUs that run Linux. However, we turned off one CPU to ensure that both jobs are accessing the same CPU. More details of the experimental platform are described in Section 4. We measured completion time of each job with another job in order to understand the interference between jobs. When the same type of jobs coexist in the system, we observe a linear relationship between total completion time and individual completion times. However, for different types of jobs in the system, a non-linear relationship is observed. This example suggests that we may reduce the performance interference among co-located jobs by considering multiple resources in a system. An empirical study on a large scale data center has shown that multiplexing workloads leads to higher efficiency since each workload utilizes different system resources [10].

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?*

The previous simple example implies that, in a realistic environment, a machine or a system consists of multiple resources and hosts jobs that can request any of the resources, as shown in Figure 2. Recall, when we assume that a job accesses a single resource such as a processing unit, we may characterize a job by its completion time to estimate interference as shown in Figure 1. For multiple resources, on the contrary, a 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 to the following question:

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

In this work, we consider a job as a scheduling unit such as a process. We model a job as a sequence of hypothetical *job slices*, each of which is devoted to a single resource in order to approximate the behavior of real workloads. 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*. Note that the loading vector represents the portion of time to access each resource during the execution of a job, which should be dependent on the hardware that a job runs.

As for approximating a real job with job slices and characterizing a real job with a loading vector, three major issues need to be addressed: simultaneous access of resources; dynamically varying resource access patterns; and multiple resources of the same type such as multi-core CPUs. Since loading vector represents the statistical characteristics to access each resource – average intensity to access each resource, simultaneous resource accesses can be approximated. Specifically, we show that our model reasonably captures a mixture of I/O workloads and CPU workloads in Section 5. We may approximate a dynamic workload as a series of static workloads, each of which can be represented by a loading vector. In this study, we will focus on static workloads since treating time-varying parameters requires additioal effort. As for n-core processors, we may reserve one element for each core in a loading vector. More details will be discussed in Section 3.

Finally, in order to formulate the *model* 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 waiting time for a resource-queue. 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* – referred to as slow-down of job completion time under the presence of other jobs.

3. MATHEMATICAL MODELING

As illustrated by the simple example in Section 2, if jobs on a single machine compete over multiple shared resources, the dilation of their completion times can exhibit 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 any more if they compete over multiple resoueces; jobs can even have different dilation factors. To clearly describe this phenomenon, we need the modeling of machines with multiple resources.

3.1 Modeling: machines and jobs

A machine with multiple resources can be viewed as a collection of multiple resource-queues; each resource corresponds to a queue and requests from jobs running on the machine compete for the resources. Let a machine have 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, we introduce the concept of *job-slices* – a hypothetical

atomic unit of work which is characterized by the following assumptions; (i) a job-slice requests for and can be processed by only a single resource, thus, it is also an atomic unit of request for the resource queues. (ii) It takes 1 (hypothetical) time-unit for a job-slice to be processed by the resource in which it is queued, independent of the type of resource.

A job is considered as a sequence of job-slices each of which requests a single resource. If a job consists of ℓ job slices, its *neutral* (which means undisturbed by other jobs) completion time is ℓ time-units. A request is viewed as the submission of a 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. For n concurrent jobs on the machine, we make the following two assumptions: (iii) there are at most n slices (including the one in service) in any one of the m queues. (iv) At any time, there are total n slices in all the queues. Assumption (iii) implies that the time for a slice to be completed is at most n time-units.

Remark 1. *Notice that the actual amount of work during a unit time such as the amount (in MB) of file processed during the time depends on the system configuration; for example, if the operating system utilizes I/O buffer cache, some of the I/O requests turn effectively into memory requests. Thus, it might not be feasible to generally estimate the actual amount of work for multiple-resource workloads by investigating the program source and hardware specification. Characteristics of workloads will be obtained by probe processes or by regression analysis of their completion times, which will be illustrated in this paper.*

Based on assumption (ii), we are actually proposing a unified measure of work done by different resources; Instead of counting the amount of work done by resources by their native measure such as the number of instructions (for CPU) and the file size in MB (for I/O), we represent the amount of work by the the time spent by any resource for the task. Thus, one unit of work is the amount of work that can be done by any resource in unit time. This allows us to compare the workloads of different kinds using a common unit.

Recall the only required information for the linear sum model for single-resource machines is the neutral completion time of the job. For a multiple-resource model, additional information is required: the request–resource-queue correspondence. The amount of information increases with the size of the system, the number of resource types to be considered, and the number of workloads if we want to obtain the complete description as with queuing model approaches. Hence, we take a statistical approach.

3.2 The loading vector

In this paper, we characterize a job by the statistical pattern of its resource requests. In general, a deterministic resource-access pattern in time is difficult to obtain except for very specific applications of known structures. Instead, we assume a job has, for each resource, a unique probability of a slice to be queued for the resource; that is, we view that the number of job-slices queued for each resource divided by the number of total slices of the job is a statistical invariant of a job. Thus, in an m-resource model, the workload of a job is characterized by an m-dimensional vector, which we call the *loading vector* $\boldsymbol{p}$ of the job.

Remark 2 (Multicore systems). *In order to avoid possible confusion, we present the theory and experimental results for single-core systems since there exists a variety of multi-core processor architectures and process scheduling policies. However, the application of the presented model to multicore systems would be possible. For example, as many operating systems treat, a dual-core system can be considered as a machine with 2 CPU-resources independent of each other. Thus, if we consider CPU and I/O resources in the model, the resulting loading vector can be 3 dimensional – namely, CPU1, CPU2, and I/O. Then, we can apply the presented model to identify the 3 dimensional loading vector of a job.*

Thus, without any other competing job (in our terms, in the neutral condition), p_i are given by

$$p_i = \frac{\text{the time spent at the resource-}i}{\text{the total completion time}} \geq 0. \tag{1}$$

Also, p_i can be interpreted as the probability of a job-slice being queued for queue-i. Obviously,

$$\|\boldsymbol{p}\|_1 = \sum_{i=1}^{m} p_i \leq 1. \tag{2}$$

Notice the inequality; we allow a job to have slices that do not request any resource. Such an *idling* slice simply spends a time-unit without populating any of the m queues. We define two classes of jobs: a job is

1. *idle* if $\sum_{i=1}^{m} p_i < 1$ and
2. *busy* if $\sum_{i=1}^{m} p_i = 1$.

Given a set of n jobs, we denote $\boldsymbol{p}_j$ as the loading vector of job-j and p_{ij} as the ith component of $\boldsymbol{p}_j$. 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.

3.3 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 at queue-i. Then, the expected waiting time of the job-slice (of job-j) is the expected number of job-slices (from other jobs) in the queue. Hence, the conditional expectation of the dilated completion time of the single slice of job-j, on the condition that it is in queue-i, is given by

$$1 + \sum_{\substack{k=1 \\ k\neq j}}^{n} p_{ik} = 1 + \sum_{k=1}^{n} p_{ik} - p_{ij}\,, \tag{3}$$

where the additional 1 time-unit is the service time.

Since p_{ij} is the probability of queuing a slice of job-j at queue-i, the expectation of the dilated completion time T_j of job-j is given by

$$\begin{aligned} T_j &= \tau_j \left(\left(1 - \sum_{i=1}^{m} p_{ij} \right) + \sum_{i=1}^{m} p_{ij} \left(1 + \sum_{k=1}^{n} p_{ik} - p_{ij} \right) \right) \\ &= \tau_j \left(1 + \sum_{i=1}^{m} p_{ij} \sum_{k=1}^{n} p_{ik} - \sum_{i=1}^{m} p_{ij}^2 \right) \end{aligned} \tag{4}$$

where τ_j is the neutral completion time of job-j. Notice, the term $(1 - \sum_{i=1}^{m} p_{ij})$ represents the probability of idling that

causes no dilation. Let us define the total loading vector $\overline{\boldsymbol{p}}$ by $\overline{\boldsymbol{p}} = \sum_{j=1}^{n} \boldsymbol{p}_j$. The above relation can be rewritten in vector notation by $T_j = \tau_j \left(1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j\right)$. Thus, a job j is slowed down by the factor of T_j/τ_j due to resource contention. We summarize the result by introducing the dilation factor $\lambda_j = T_j/\tau_j$ as follows:

Definition 1. *Given a job set on a machine characterized by the loading vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, n)$, the dilation factors $\lambda_j = T_j/\tau_j$ $(j = 1, \ldots, n)$ are given by*

$$\lambda_j = 1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j \,. \tag{5}$$

Remark 3. *The above formula distinguishes our model from other applications of queuing theory. Combined with the representation of loading statistics by the loading matrix presented above, we can describe the average slow-down of a job in a closed formula. Also, the formula can be viewed as a statistical description of the interaction between jobs running on the same machine in terms of mutual interference.*

For identical jobs, the formula can be simplified as follows:

Property 1. *If all of n jobs are identical ($\boldsymbol{p}_j = \boldsymbol{p}$), the dilation factors are identical ($\lambda_j = \lambda$) and are given by $\lambda = 1 + (n-1)\,\boldsymbol{p} \cdot \boldsymbol{p}$.*

The above property can be utilized 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 $\boldsymbol{p}$.

The dilation factors are also identical in another situation. By applying $n = 2$ and $\overline{\boldsymbol{p}} = \boldsymbol{p}_1 + \boldsymbol{p}_2$ to Equation (5),

Property 2. *If there are 2 jobs, the dilation factors are identical ($\lambda_1 = \lambda_2 = \lambda$) and given by $\lambda = 1 + \boldsymbol{p}_1 \cdot \boldsymbol{p}_2$.*

If we create a synthetic process which requests only a single resource, this reference (or *probe*) process can be utilized to compute the loading vector of a job. The primary benefit of the probe processes is that we can identify the loading vector of a job when we cannot control the execution of a job in the system.

Property 3. *Suppose there are two jobs – one that we want to identify its loading vector p and the other that uses only resource-i, a probe process. By Property 2, they share the same λ and the ith component of p is given by $p_i = \lambda - 1$.*

3.4 Special case: 1-resource-busy jobs (the linear sum model)

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

Lemma 1. *Assume 1-resource-busy jobs: $p_{kj} = 0$ for any $j = 1, \ldots, n$ and for every $k = 1, \ldots, m$ except an index $1 \le i \le m$, and $\|\boldsymbol{p}_j\|_1 = 1$. Then, the dilation factors are identical ($\lambda_j = \lambda$) for all jobs and given by*

$$\lambda = n \tag{6}$$

Proof. Since all the jobs are busy, $p_{ij} = 1$ and $p_{kj} = 0$ for any $k \neq i$. Hence, all jobs are identically given by $\boldsymbol{p}_j = \boldsymbol{p}$. Then, $\boldsymbol{p} \cdot \boldsymbol{p} = 1$ and, by Property 1 in the previous section, the dilation factors are identical and given by $\lambda = 1 + (n-1)\,\boldsymbol{p} \cdot \boldsymbol{p} = 1 + (n-1) = n$ □

Let τ_j denote the neutral completion time of job-j in a job set of size n. We define the total completion 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. For 1-resource-busy jobs, $T = \sum_{j=1}^{n} \tau_j$.

Theorem 1. *Suppose during the total completion time, there is no idling gap, i.e. the machine is always populated by at least one job. Then, the total completion time is the sum of individual neutral completion times independent of the starting and ending times of the jobs, that is,*

$$T = \sum_{j=1}^{n} \tau_j \,. \tag{7}$$

Proof. Please refer to [19]. □

Note that Lemma 1 and Theorem 1 assume that jobs are fully overlapped for estimating completion times.

Remark 4. *In general cases, such a simple formula for total completion time does not exist. Consider a 2-job system with $\boldsymbol{p}_1 = (1, 0)$ and $\boldsymbol{p}_2 = (0, 1)$. Then, $\lambda_1 = \lambda_2 = 1$, that is, there is no dilation of completion 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 completion time will be 2 minutes. Depending on the overlap, the total completion time can vary from 1 minute to 2 minutes. But, still the linear model estimate becomes the upper bound of the total completion time in any case.*

Remark 5. *(1-resource-idling jobs) Even if there's only one requested resource, the linear 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 $p < 1$. Then, $\lambda = 1 + (n-1)\,p^2 = (1-p^2)\,1 + p^2\,n$, that is, λ is a linear interpolation of 1 and n with respect to p^2, since $p < 1$ and $\lambda < n$.*

3.5 Special case: 2-resource-busy jobs

The usefulness of this model comes from the fact that each loading vector $\boldsymbol{p}_j$ can be represented by a single scalar parameter p_j. Without loss of generality, we can remove non-requested resources from considerations and assume $\boldsymbol{p}_j = (p_j, 1 - p_j)$. Then, $\overline{\boldsymbol{p}} = (\overline{p}, n - \overline{p})$ where $\overline{p} = \sum_{j=1}^{n} p_j$, and

$$\begin{aligned} \lambda_j &= 1 + p_j \overline{p} + (1 - p_j)(n - \overline{p}) - p_j^2 - (1 - p_j)^2 \\ &= 1 + n - \overline{p} - n\,p_j + 2\,p_j\,\overline{p} - p_j^2 - (1 - p_j)^2 \,. \end{aligned} \tag{8}$$

The result can be further simplified if the jobs are identical, i.e. $p_j = p$. Then, $\overline{p} = n\,p$ and, for any $j = 1, \ldots, n$,

$$\lambda_j = 1 + n - 2\,n\,p + 2\,n\,p^2 - p^2 - (1 - p)^2 \tag{9}$$

$$= 1 + n(1 - 2\,p + p^2) + (n-1)\,p^2 - (1 - p)^2 \tag{10}$$

$$= 1 + (n-1)\,\left(p^2 + (1-p)^2\right) \,. \tag{11}$$

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

Theorem 2. *Assume 2-resource-busy identical jobs with the loading vector given by $(p, 1-p)$. Then, the dilation factors are identically given by*

$$\lambda = 1 + (n-1)\,\left(p^2 + (1-p)^2\right) \tag{12}$$

In other words, if the dilation factor is known, we can obtain the loading vector by the formula:

$$p = \frac{1}{2}\left(1 \pm \sqrt{1 - 2\frac{n-\lambda}{n-1}}\right) \tag{13}$$

PROOF. The derivation is given above. □

Notice that there are two possible solutions for p and the model does not distinguish them.

The above relation suggests a simple experimental strategy to obtain the loading vector using Algorithm 1. Notice that we can characterize jobs only from their completion times, which does not require resource monitoring or kernel instrumentation. However, by utilizing system resource monitor, we can enhance the accuracy of the model.

Algorithm 1 Constructing the loading vector of a job in 2-resource model.

1: Measure τ by running job j alone.
2: Measure T, the dilated completion time of job j when n instances of job j are running concurrently in the system.
3: Evaluate the dilation factor $\lambda_j = T/\tau$.
4: From Equation 13, obtain $p_i = \frac{1}{2}\left(1 \pm \sqrt{1 - 2\frac{n-\lambda}{n-1}}\right)$
5: Obtain loading vector $\boldsymbol{p}_j = (p_i, 1 - p_i)$

3.6 The total dilation factor

One of the potential benefits of D-factor model is the capability to extend existing schedulers for single-resource machines into schedulers for multi-resource machines. Towards this end, we define the total dilation factor $\overline{\lambda}$ by the sum of λ_j for all jobs. Then, by the formula given in Definition 1,

$$\overline{\lambda} = \sum_{j=1}^{n} \lambda_j = n + \|\overline{\boldsymbol{p}}\|_2^2 - \sum_{j=1}^{n} \|\boldsymbol{p}_j\|_2^2 . \tag{14}$$

Notice, this simple formula involves only the L^2 norms of the loading vectors; informally the absolute values of the loading vectors. For details, refer to [19]. With the total loading vector, we can state a new objective function for a certain class of scheduling problems as follows:

Definition 2. Given a new job with the loading vector $\boldsymbol{p}'$, find an assignment to machine μ which minimizes

$$\boldsymbol{p}' \cdot \overline{\boldsymbol{p}}_\mu \tag{15}$$

where $\overline{\boldsymbol{p}}_\mu = \sum_{\forall j \in \mu} \boldsymbol{p}_j$ is the current total loading vector of machine-μ.

4. EXPERIMENTAL ENVIRONMENT

This section describes the experimental settings designed to validate the proposed D-factor model.

4.1 Target system overview

We experimented with clusters in two environments – native Linux and Xen-based virtualized environments [4]. Table 1 shows the detailed specifications of experimental settings for the native Linux and virtualized environment. In the native Linux environment, each node is running only one Linux operating system (OS). On the contrary, in the VM environment, each node can host multiple OS instances. The overview of the VM environment is shown in Figure 3.

Table 1: Specifications for experimental environment

CPU	Two single-core 64-bit AMD 2.4Ghz
RAM	4GB
Shared Storage	NFS (disk images for Xen)
Local Storage	Ultra320 SCSI
Network	1Gbps Ethernet, 10Gbps Infiniband
vCPU (Dom0)	Runs on both CPUs
vCPU (VMs)	Runs on one CPU
RAM/VM	256MB
I/O (VM)	Tap:aio (bypasses buffer cache of Dom-0)
Kernel	Linux 2.6.18
Hypervisor	Xen 3.4.2

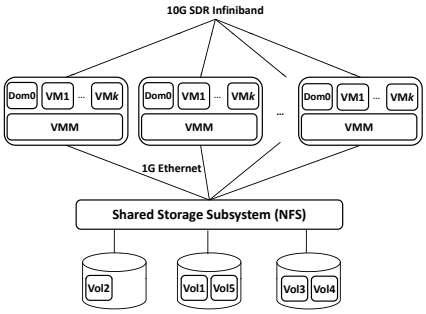


Figure 3: Virtualized experimental environment

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 physical resources, especially I/O devices. The Virtual Machine Monitor (VMM) manages the shared resources such as processors, memory, I/O subsystems and network devices. All the access requests to the hardware resources from applications running on a virtual machine flow into the VMM and then Dom-0, if necessary. Each node in the native Linux environment accesses local SCSI disks whereas the virtual hard disks of guest virtual machines are located in a Network File System (NFS) partition. Thus, I/O requests from guest machines may invoke network traffic. We employed varying number of applications on each node in the native Linux environment. In the VM environment, we run one application per virtual machine, but we varied the number of virtual machines.

4.2 Description of workloads

We used a mixture of synthetic and realistic workloads to validate the proposed D-factor model on a variety of workload environments. For synthetic workloads, we employed *standard jobs* (the details of which will be described later) and file compressions that use both CPU and I/O resources. For realistic workloads, we used I/O-bound workloads (FileBench [2]) and MapReduce workloads. Table 2 summarizes resource usage profile of workloads in this study.

Synthetic workloads – standard jobs and file compression. We have created standard jobs for two purposes. The first is to demonstrate that the proposed model can capture completion times of synthetic workloads using only one system resource. The second is to illustrate a method to profile workloads. *Standard job* is a workload that uses only one resource without any idle period. The completion time of a

Table 2: Resource Usage Profile of Workloads

Type	Name	CPU	I/O
Synthetic	`STD-CPU`	High	Low
Synthetic	`STD-IO`	Low	High
Synthetic	`FileComp`	High	Med
FileBench	`FileServer`	Low	High
FileBench	`VarMail`	High	Med
MapReduce	`PiEstimator`	High	Low
MapReduce	`Sort`	Med	Med
MapReduce	`Grep`	Med	Med

standard job should be consistent, provided that the standard job is running alone in the system. For example, the completion time of a standard job for disk I/O should not depend on the current size of free memory or the current status of buffer cache. In this study, we created two standard jobs – a standard job for CPU (`STD-CPU`) and a standard job for disk I/O (`STD-IO`). `STD-CPU` is a workload that repeats integer arithmetic calculations. `STD-IO` is a workload that opens two 4GB files on the local disk in the synchronous mode and, then, reads 1MB from random positions of both files, while avoiding to access the buffer cache using the POSIX library, specifically `posix_fadvise`.

We have created a workload, `FileComp` to show that we can profile a CPU-I/O workload with standard jobs. `FileComp` compresses 2,560 files, each of 256KB size. The neutral completion time of `FileComp` τ is $78.08sec$. With `STD-CPU` of the duration of $200sec$, the completion time of `FileComp` T is $123.67sec$. If we assume that a system consists of two resources – CPU and I/O, we may consider `FileComp` as a two-resource-busy job. Since `STD-CPU` is a CPU-only job, the loading vector is $(CPU, IO) = (1, 0)$. Let the loading vector of `FileComp` be $\boldsymbol{p} = (p, 1-p)$. According to Property 2 in Section 3, the dilation factor $\lambda = 1 + p$. λ can be calculated by $T/\tau = 123.67/78.08 = 1.58$, which yields $p = 0.58$. Hence, the loading vector of `FileComp`, $\boldsymbol{p} = (0.58, 0.42)$. We used `FileComp` for evaluating our model in the native Linux environment. In this example, we profiled a job using standard jobs. However, it is non-trivial to implement these standard jobs due to their dependence on the system configuration. Thus, with FileBench workloads, we demonstrate an alternative way to profile a job – Algorithm 1.

Realistic workloads – FileBench and MapReduce. We experimented with FileBench in a virtualized environment. We employed two predefined workloads in FileBench, `fileserver` and `varmail`. FileBench is an application benchmark that mimics the typical behavior of workloads. We employed one FileBench workload per VM and varied the number of VMs. According to Algorithm 1, both workloads are profiled by $\boldsymbol{p}_{file} = (0.02, 0.98)$ and $\boldsymbol{p}_{mail} = (0.10, 0.90)$.

We experimented with three MapReduce workloads : `Sort`, `Grep`, and `PiEstimator` in the native Linux environment. MapReduce job consists of multiple tasks that are hosted in parallel on different cluster nodes. We run these MapReduce workloads on a dedicated 17-node native Linux cluster with Hadoop 0.20.1 [1]. Each workload generates two map-and-reduce tasks on each node. Each task accesses the local disk and exhibits insignificant communication between each other. The completion time of an application is determined by the completion time of the slowest task. For these experiments, we confirmed that the variance of completion times between tasks for one application is insignificant. Since our experimental platform consists of two single-core CPUs, increasing the number of instances of an application creates resource contention for each CPU. We conducted two experiments; (i)increasing the number of instances of each application from 1 to 4 and (ii) co-locating three different MapReduce workloads in the system. Applications are initiated at the same times and we used the Hadoop fair scheduler [1] to allocate system resources to these applications.

In order to estimate the completion times with the D-factor model, we profiled each workload with `STD-CPU`, similar to the `FileComp` workload. The loading vector, $\boldsymbol{p} = (CPU, \textit{other resources})$, and the neutral completion time, τ, of each application are $\boldsymbol{p}_{Sort} = (0.9, 0.1)$, $\tau_{Sort} = 56.0sec$, $\boldsymbol{p}_{Grep} = (0.5, 0.5)$, $\tau_{Grep} = 95.0sec$, $\boldsymbol{p}_{Pi} = (0.5, 0.5)$, $\tau_{Pi} = 90.0sec$. Then, the loading matrix is given by

$$P = \begin{bmatrix} 0.9 & 0.5 & 0.5 \\ 0.1 & 0.5 & 0.5 \end{bmatrix}.$$

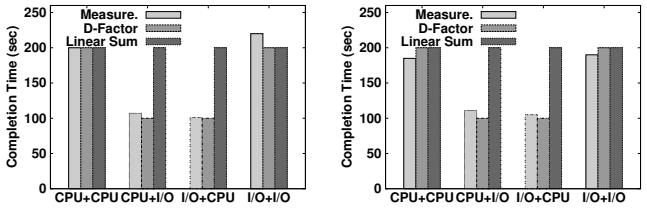
Note that the loading vector represents the probability of accessing each resource instead of the utilization. Thus, resource usage does not necessarily match with the loading vector. For example, regardless of high CPU usage, some other factors may dominate the execution time of `PiEstimator`, which is captured in the loading vector representation.

Methodologies. We compare the estimated total completion time from the D-factor model and the linear sum model with actual measurements. We compare D-factor with linear sum since both methods do not require information of request arrival rate and service rate as in queuing analysis. For two jobs i and j when $\tau_i > \tau_j$, we can calculate the actual completion times of two jobs, T_i and T_j, from the dilation factors, λ_i and λ_j, obtained by $T_i = \lambda_j \tau_j + \left(1 - \frac{\lambda_j \tau_j}{\lambda_i \tau_i}\right) \tau_i$, $T_j = \lambda_j \tau_j$. Since job j finishes at $\lambda_j \tau_j < \lambda_i \tau_i$, $1 - \frac{\lambda_j \tau_j}{\lambda_i \tau_i}$ of job i will be executed without interference with job j. Similarly, we can calculate the actual completion times of more than two jobs. Note that in Remark 4, we have already discussed the difficulty of calculating completion time of jobs when jobs are partially overlapped during their executions. We believe that an efficient method for such a general case deserves a seperate effort.

What our model computes is the dilation factor, λ_j for job j, which is given by $\lambda_j = 1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j$, where $\boldsymbol{p}_j$ is the jth column vector of the loading matrix and $\overline{\boldsymbol{p}} = \boldsymbol{p}_1 + \boldsymbol{p}_2$ in the two-resource-busy model. To compute the dilation factor, we obtain the loading vector or loading matrix using either Algorithm 1 or Property 2 (Refer to Section 3).

5. MODEL VALIDATION

In this section, we validate the proposed D-factor model while illustrating the procedure to profile a job and to estimate the completion time of a job when it is co-hosted with other jobs. Experimental results in this section indicate that the D-factor model captures the behavior of each job in shared systems. The D-factor model provides (1) more accurate estimation of the completion times of co-hosted jobs than the linear sum model, (2) more efficient utilization of the system resources and (3) better predictable performance with existing scheduling algorithms than with the linear sum. Note that all numbers are averaged over 40 runs since dilation factor model actually aims to predict average completion time of jobs, not to predict variance of completion time of jobs. Numbers in parenthsis represent



(a) Native Linux (b) Virtualized Environment

Figure 4: Experiments with standard jobs show the average errors from dilation factor are 7% for the native Linux and 10.3% for the virtualized environment. CPU represents `STD-CPU` and I/O represent `STD-IO`. We hosted two processes in (a) and two VMs in (b).

the relative error of the estimation given by $|(\hat{T} - T)/\hat{T}|$, where $\hat{T}$ is from measurement and T is from estimation.

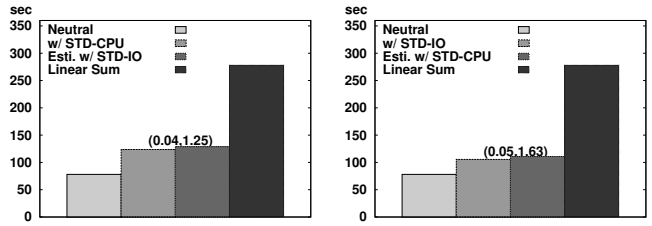
5.1 Synthetic workloads

Standard jobs. As shown in Figure 4, the experiments with both `STD-CPU` and `STD-IO` confirm that the proposed model estimates the slow-downs of completion times of jobs due to multiple resource contention in shared systems. Since each standard job uses only one resource, we may consider `STD-CPU` characterized by $\boldsymbol{p} = (1, 0)$ and `STD-IO` by $\boldsymbol{p} = (0, 1)$. With the given loading vectors, we obtain $\lambda = 1$ for two different standard jobs and $\lambda = 2$ for two identical standard jobs according to Theorem 1. When we set the duration of each standard job to $100sec$, the total completion time of two different standard jobs will be $100sec$ and that of two identical standard jobs will be $200sec$. Hence, we may observe that the completion time of a job in a shared multi-resource system is not linearly proportional to the completion times of individual jobs.

The gist of the dilation factor theorem is that multiple resource contention results in a non-linear waiting time, proportional to their probabilities of accessing the system resources. In contrast, the linear sum model estimates the total completion time as the linear sum of completion times of individual jobs. The linear sum fits when the system owns one resource or serves each job after completing the previously assigned jobs. Experiments from standard jobs indicate the average errors from D-factor model are 7% for the native Linux and 10.3% for the virtualized environment. However, the linear model shows a 98% error in the worst case.

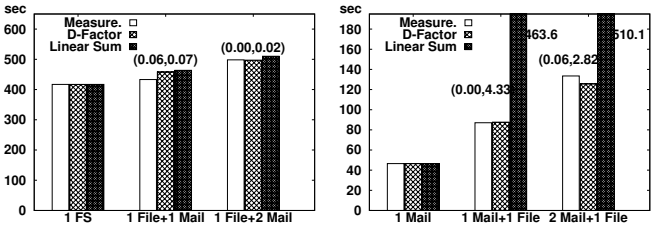
Figure 4a shows completion time of each co-located workload in the native Linux (two processes) and Figure 4b for the virtualized environment (two VMs). These experiments confirm that D-factor model accurately captures the completion times of jobs with multiple shared resources in both native Linux and virtualized environments. In addition, we show that we can construct a standard job to profile workloads according to the proposed model in actual systems. However, we observe a significant error for two identical `STD-IO`, which is attributed to variance in disk access latencies to perform the same operations according to the sequence of access requests [31].

File compression. Now, we demonstrate that the proposed model can estimate the completion time of a workload with both CPU and disk I/O accesses, `FileComp` (refer to Figure 5). In these experiments, we set the duration of `STD-CPU` and `STD-IO` to $200sec$. For the loading vector of `FileComp`, $\boldsymbol{p} = (0.58, 0.42)$, the estimated completion time will be $78.1 \times (1 + 0.42) = 110.9sec$ since we assume that the loading vector of the I/O standard is $(0, 1)$. From actual measurements for `FileComp` with the I/O standard, we obtain the completion time as $105.50sec$, resulting in a 5.1% error. We have shown the accuracy of estimation from D-factor model with synthetic workloads. Next, we show the validation results with realistic workloads.



(a) `FileComp` with `STD-CPU`. (b) `FileComp` with `STD-IO`.

Figure 5: Estimated completion times of `FileComp` (a) using the loading vector of `FileComp` from measurements with `STD-CPU`; and (b) using the loading vector to predict the completion time with `STD-IO`. The numbers in parenthesis represent relative errors from D-factor and linear sum, respectively.



(a) `fileserver` (b) `varmail`

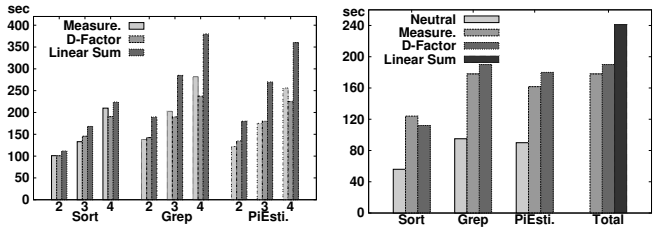
Figure 6: Completion times for various combinations of FileBench workloads on one physical machine with up to 3VMs. The loading vectors are obtained as to Algorithm 1.

5.2 Realistic workloads

FileBench in a virtualized environment. Experimental results with FileBench are shown in Figure 6. We confirm that the D-factor model well explains the dilated completion times for collocated I/O workloads in a virtualized environment as well.

With the provided loading vectors for both workloads, we estimate the completion time as $458.1sec$ when we co-locate one `varmail` and one `fileserver` in the same physical machine, compared to $433.1sec$ with actual measurements. In this situation, we estimate the running time of `varmail` to be $87.62sec$, which shows an error of less than 1% with actual measurements. However, the linear model cannot estimate the completion time of each workload. With the linear model, if we consider the total completion time of all the workloads in the system as the completion time of each job, the completion time of `varmail` becomes $463.6sec$.

From Figure 6, we can observe that for one instance of `fileserver` and two instances of `varmail`, the respective average running times are $133.5sec$ and $498.3sec$. The corre-



(a) Identical MapReduce (b) Heterogeneous MapReduce

Figure 7: (a) Total completion time of each MapReduce application against the number of instances. The maximal errors are 9.4% for `Sort`, 15.78% for `Grep`, and 12.1% for `PiEstimator`. (b) With the loading vectors of each of three applications, we can estimate the completion time of each hetorogeneous MapReduce application. Both experiments used a 17-node cluster.

sponding average running times from estimation are 133.48*sec* and 497.2*sec*. From the experimental results with FileBench, the estimation from the D-factor model shows 6.0% error. Next, we show more complex examples of identical MapReduce applications and heterogeneous MapReduce applications in a system.

Identical MapReduce applications. Experiments with identical mapreduce applications on a 17-node cluster demonstrate the accuracy of the D-factor model as the number of instances increases up to four. In Figure 7a, the estimation from dilation factor theorem shows 9.4% error for `Sort`, 15.78% for `Grep`, and 12.1% for `PiEstimator` in the worst case. Compared with the D-factor model, the linear model shows larger margin of errors for all the considered workloads, 26%, 38%, and 54% for `Sort`, `Grep`, and `PiEstimator`, repsectively. We observe that applications with substantial I/O accesses tend to show larger errors as we increase the number of instances in the system. We speculate that this trend stems from the variance in disk latencies similar to what we observed for `STD-IO` cases. We plan to model this behavior in future work.

Heterogeneous MapReduce applications. Figure 7b shows that the D-factor model estimates the completion times of three co-hosted MapReduce applications within 11% error. Loading vector and neutral completion time of each application remain the same as identical MapReduce experiments.

This experiment contrasts the D-factor model with the linear model in terms of the ability of estimating completion times of individual workloads when they are hosted in a shared system. The linear model accounts for the total completion time without considering the completion times of individual jobs. However, the D-factor model obtains the total completion time based upon the completion times of individual workloads. The ability to estimate completion time of each workload is practically useful in shared systems. In our experiment, we observe that `Sort` completes in about 120*sec*. Using the linear model, a new job will be scheduled to the system after all the previous workloads are completed; 241*sec* later since we cannot estimate that `Sort` will be completed before that (refer to Figure 7b). However, the D-factor model can estimate that `Sort` will be completed in 110*sec*. Thus, we may schedule another workload at 110*sec* after initiating all the applications.

We notice that the measured completion times of `Grep` and `PiEstimator` are smaller than the estimation. In this analysis, we assumed two-resource-busy jobs, which means there is no idle period in processing the given workloads and consumes only two resources. Since `Grep` and `PiEstimator` have substantial I/O activities, we may have idle wait times for completing I/O accesses. When we multiplex I/O workloads with other types of workloads, operating systems may utilize idle wait times. Thus, the completion time of `Grep` and `PiEstimator` could be smaller than the estimation when they are co-hosted. We have briefly discussed how to handle jobs with idle periods in Section 3.4 (refer to Remark 5).

To summarize, through experiments, we demonstrated that the D-factor model can estimate the total completion time of jobs when they share multiple system resources. In this section, we experimented with synthetic workloads – standard jobs and file compression as well as realistic workloads – MapReduce and FileBench. We demonstrated that the D-factor model well describes the behavior of both the native Linux and virtualized systems since the model is a high-level statistical abstraction of system behaviors. We showed that the D-factor model does not only estimate the total completion time, but also estimates the completion times of individual jobs. We can observe that the D-factor model can quantitize the performance of multiplxed workloads, given the workload profiles. Also, we showed the process to use the D-factor model to estimate the completion times of given workloads: construct standard jobs, probe the target workloads using the standard jobs, and estimates completion times of given workloads.

Experiments in this section suggest that we may predict I/O performance for cloud services [3] using the D-factor model. In the following section, we demonstrate an example for exploiting the D-factor model to enhance actual system performance with a scheduling algorithm.

6. EXTENDING SCHEDULERS FOR SINGLE RESOURCE SYSTEMS

This section describes practical benefits from the proposed D-factor model by applying it to an existing scheduling algorithm. We selected the on-line parallel machine scheduling problem [30] that addresses the problem of scheduling jobs on parallel machines without the knowledge of the sequence and the size of future jobs. This problem can be translated into the on-line bin packing problem [26], which is most widely considered in assigning virtual machines to physical machines [23,32,34]. We acknowledge that it would be challenging to apply the D-factor model to other classes of scheduling problems that consider flow control of jobs or machines such as open-shop, flow-shop, and job-shop [29] since the model assumes independent jobs.

One of the initial studies on on-line parallel machine scheduling problems is the list scheduler [14], an on-line greedy scheduler, which decides schedule S of a new job j with size p_j by

$$min\left\{p_j + \sum_{i \in J_\mu} p_i\right\}, \forall \mu \in M\,, \tag{16}$$

where J_μ is the set of jobs that is currently running in the machine μ and M is the set of machines. For breaking a tie, the lowest indexed machine is chosen. Although this algorithm is simple, competitive ratio of list scheduler is 2, while

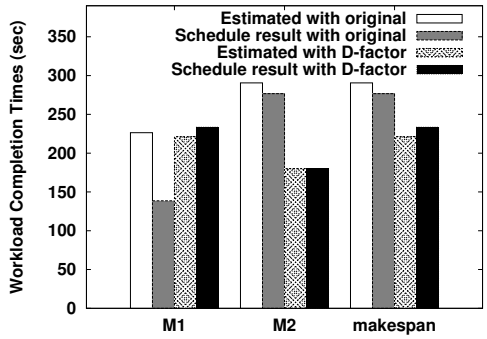


Figure 8: Adopting D-factor as objective function in Graham's on-line scheduler [14] shortens the makespan by 20.5%, compared with the original algorithm.

Table 3: Job profiles for the scheduling example.

workload	completion time	loading vector
W_1	110.56 sec	(1,0)
W_2	115.83 sec	(0,1)
W_3	180.0 sec	(1,0)
W_4	110.56 sec	(1,0)

the best result known to date is 1.9201-competitive algorithm for deterministic algorithms [12] and 1.58-competitive algorithm for randomized algorithms [7]. Thus, we selected this algorithm to demonstrate the performance enhancement by adopting the D-factor model in scheduling algorithms, though we do not claim any theoretical improvements in scheduling problems.

Here, we created four jobs using synthetic workloads, `STD-CPU` and `STD-IO`, which run on two machines on the same experimental platform described in Section 4. We consider a situation that three jobs (W_1, W_2, W_3) are already in the system and a new job, W_4, arrives at the system $60sec$ later. The size, completion time and loading vector, of each job are shown in Table 3. Experimental results shown in Figure 8 demonstrate that the makespan with D-factor is 20.5% shorter than that with the original list scheduler, which is a significant enhancement since the theoretical lower bound of deterministic on-line parallel machine scheduling problem is 1.88-competitive [25], which is a 12% enhancement (1.88 compared to 2). However, with the multi-resource contention model, performance enhancement may exceed the theoretical bound for single-resource machines.

Let us explain the rationale behind this performance improvement. The original list scheduler schedules the first three jobs as follows:

$$\begin{array}{llll} Machine1 & : W_1 & W_2 & (226.4sec) \\ Machine2 & : W_3 & & (180.0sec)\text{'} \end{array}$$

where the numbers in the parenthesis represent estimated completion times of jobs in each machine. When the fourth job, W4 with job size of $p_4 = 110.56sec$, arrives $60sec$ later, list scheduler schedules the fourth job to Machine 2, expecting $180.0 + 110.56sec$ of total completion time. Thus,

$$\begin{array}{llll} Machine1 & : W_1 & W_2 & (226.39sec) \\ Machine2 & : W_3 & W_4 & (290.56sec)\text{·} \end{array}$$

By augmenting the objective function (Equation 16) to Equation 15, we can use vector-valued size, loading vectors. Then, the list scheduler with loading vectors schedules the first three jobs as follows:

$$\begin{array}{llll} Machine1 & : W_1 & W_2 & (\overline{\boldsymbol{p}} = (1,1), 115.83sec) \\ Machine2 & : W_3 & & (\overline{\boldsymbol{p}} = (1,0), 180.0sec)\text{ '} \end{array}$$

where $\overline{\boldsymbol{p}}$ is the total loading vector and the numbers are estimated total completion times based upon D-factor model. Notice that, for the same configuration of machine 1, the estimated completion time with D-factor model is $115.83sec$, compared to $226.4sec$ with the original list scheduler based upon linear sum. Actual measurement of this schedule was $138.65sec$, which confirms that the D-factor model provides better estimation of total completion time. This will lead to a contrasting schedule when W_4 arrives $60sec$ later. For W_4 with $\boldsymbol{p}_4 = (1,0)$, the schedule becomes

$$\begin{array}{llll} Machine1 & : W_1 & W_2 \quad W_4 & (\overline{\boldsymbol{p}} = (2,1), 221.2sec) \\ Machine2 & : W_3 & & (\overline{\boldsymbol{p}} = (1,0), 180.0sec)\text{'} \end{array}$$

which results in 20.5% enhancement in makespan.

In this example, we demonstrated that the same scheduling algorithm may show significant performance gain by evaluating the total completion time of candidate schedules with the D-factor model. In general, we can expect better scheduling results with the D-factor model when workloads access multiple resources and they are independent since the D-factor model considers multiple resources contention among independent jobs. In order to consider multiple resources in a system, we may consider other types of scheduling problems such as open-shop, flow-shop, or job-shop scheduling problems. Instead, the D-factor model can augment existing on-line parallel machine scheduling algorithms that assume a machine as one resource, to consider multiple resources in a machine. Hence, when a system uses on-line parallel machine scheduler or on-line bin packing algorithms, we could augment the system without significant changes in software/hardware to reduce the makespan of workloads.

7. RELATED WORK

Despite the importance of estimating dilation factor of each co-located job for providing predictable performance in shared environments, there is no established generic approach to estimate dilation factor of workloads in shared envionments [13, 20]. The main difficulty to estimate the dilation factor of each job is that it depends on the behavior of co-located jobs, which is often described as non-deterministic behavior [17] and a critical problem in Cloud platforms [3].

As an attempt to estimate dilation factor, Govindan *et al.* [13] and Mars *et al.* [20] proposed an empirical method for sharing memory subsystems. Especially, Mars *et al.* demonstrated that they could predict dilation factors up to three co-located Google's workloads in Google's infrastructure. Both studies created a probe program to measure the sensitivity of each workload for sharing memory/cache. Based upon the empirical sensitivity analysis, they estimated slow-down of workloads with different co-located workloads. With our model, both works can be considered as an example of two-resource busy model to express the slow-down of co-located applications. Their concepts of sensitivity of each application and pressure on shared resources are captured in our statistical job characteristics – loading vector. In addition, we suggested probe jobs for other resources such as CPU and disk I/O. We believe that loading vector can be constructed using probe jobs in [13, 20] in order to consider spatial characteristics of memory systems.

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 [8]. 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 multi-bin packing, we model bins and the sizes of items as vectors, but it does not consider the contention across resources when we place items to bins. Thus, it cannot express the non-linearity of dilation factors of workloads in shared environments. Since most of prior virtual machine placement methods are based upon on-line bin packing algorithms [23, 32, 34] or similar linear relations [15], they are inherently difficult to reflect non-linear slow-down of a job due to multiple resource contention.

Unlike the prior work, we provide a model that can predict the performance of applications in shared environments. With mixed workloads in data centers, we can achieve more graceful performance degradation [10, 22]. In addition, the issue raised from I/O-bound work in [3] implies that we need to consider the importance of multiple resource requests in processing a job. Therefore, we proposed a performance model that estimates the performance impact on a job by other collocated jobs running together in a server.

8. CONCLUSIONS

In this study, we derived a novel completion time model of jobs for shared service systems. We estimate the completion time of jobs on a system by considering that multiple shared resources may be involved to process a job. The resource usage of a job is represented by a loading vector, which in turn is used to estimate the dilation in individual job completion times. Based upon the proposed model, we profiled a job and estimate the overall completion time of jobs in shared service systems. In contrast to queuing theory based models that require parameters of each resource such as service rates and request arrival rates, the D-factor model only needs to the completion times to profile a job, from which the model can estimate the completion times of co-located jobs. We validate the proposed D-factor model with experiments using synthetic and real workloads. From the validation results using synthetic workloads, the average relative errors are 7% in the native Linux environment and 10.3% in a VM environment. With realistic workloads, the D-factor model also predicts the completion time of co-existing workloads within a 16% error.Also, we presented an example to extend a scheduler based upon single-resource model, which reduces the makespan by 20.5%.

Calculating the overall completion times of the assigned jobs is the fundamental operation in designing or evaluating a scheduling algorithm. We can use the D-factor model to estimate the completion time more accurately or conveniently than previous models. The required overhead is to find the loading vector of a job to obtain the expected total completion time. As described in section 3, we can obtain the loading vectors by comparing the total completion time of multiple instances of identical jobs with the neutral completion time of the job. One of the assumptions made in this study is that the resource access pattern of each job is independent to each other. Otherwise, we cannot easily calculate the probability of the resource contention by the inner product of loading vectors.

Since sharing multiple resources among jobs is a common practice in computing systems, we can find a plenty of applications for the proposed D-factor model. Some of possible applications are as follows: Estimating the total completion time of jobs when we have heterogeneous processors such as CPU and GPU might be possible. CPUs and GPUs can be treated as different types of resources to be shared among jobs and the total completion 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 completion time of a job, we can apply the D-factor model.

Although we successfully demonstrated the potential of D-factor model, several diretions are possible to further enhance the D-factor model. These include:

- Extending the model for the space-shared resources like memory systems: the working set behavior of each co-located job will modify the loading vector of each job. In order to use the model for the space-shared resources, we need to establish a model that identifies the modifier of the loading vector of each job due to the working set behavior. The authors believe such an extention can consider in-processor cache and shared system memory, which will make the D-factor model more useful in modern many-core environments.
- An algorithm to compute the dilation factor for n partially overlapped jobs: this can be discussed in the context of job scheduling since the initialization and completion of jobs are dependent on job schedulers. As breifly mentioned in the section 6, developing a job scheduler with the D-factor model is one of the eventual goals of this study.
- Overcoming the dependency of the loading vector of a job on the measured platform: since the loading vector of a job is obtained from measurements, it is dependent on a specific hardware-operating system combination. However, the D-factor model requires relatively smaller effort than prior analytical models. One possible direction to work around this would be a method to convert the loading vector of a job in one system to other systems.

In a sense, although the proposed D-factor model is not the most perfect model to explain shared resource contention, the model provides an insight to understand contention for multiple shared resources. In addition, the D-factor model allows to extend many existing scheduling algorithms for single resources into scheduling algorithms for multiple resources.

9. ACKNOWLEDGMENTS

This research was supported in part by NSF grants CNS-0721479, CNS-1152449, and CCF-1147388. This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. This research used resources of the Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

10. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org.

[2] FileBench. http://www.solarisinternals.com/wiki/index.php/FileBench.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. of the ACM*, 53(4), 2010.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP*, 2003.

[5] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*, 2008.

[6] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[7] Bo Chen, AndrÃl' van Vliet, and Gerhard J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5), 1994.

[8] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms for bin packing: a survey*. PWS Publishing Co., Boston, MA, USA, 1997.

[9] Peter J. Denning. The working set model for program behavior. *Commun. of the ACM*, 11(5):323–333, 1968.

[10] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barrose. Power provisioning for a warehouse-sized computer. In *ISCA '07*.

[11] Alexandra Fedorova, Sergey Blagodurov, and Sergey Zhuravlev. Managing contention for shared resources on multicore processors. *Commun. of the ACM*, 2010.

[12] Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms - ESA 2000*. 2000.

[13] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated vitual machines. In *ACM SOCC*, 2011.

[14] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[15] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *VEE*, 2009.

[16] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckout, Andy Georges, Lizy K. John, and Koen De Bosschere. Performance prediction based on inherent program similarity. In *PACT*, 2006.

[17] Ravi Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Don Newell, Yan Solihin, Lisa Hsu, and Steve Reinhardt. Qos policies and architecture for cache/memory in cmp platforms. In *ACM SIGMETRICS*, 2007.

[18] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *ISPASS*, 2007.

[19] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, Galen M. Shipman, and Chita R. Das. A quantitative analysis of performance of shared service systems with multiple resource contention. Technical Report CSE-10-010, The Pennsylvania State University, 2010.

[20] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Micro*, 2011.

[21] D.A. Menasce. Two-level iterative queuing modeling of software contention. In *10th IEEE MASCOTS*, 2002.

[22] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *ICAC*, 2010.

[23] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM*, 2010.

[24] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Eurosys*, 2010.

[25] III Rudin, John F., and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM J. Comput.*, 32:717–735, March 2003.

[26] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2), 2009.

[27] Akbar Sharifi, Shekhar Srikantaiah, Asit K. Mishra, Mahmut Kandemir, and Chita R. Das. Mete: meeting end-to-end qos in multicores through system-wide resource management. In *ACM SIGMETRICS*, 2011.

[28] Bikash Sharma, Victor Chudnovsky, Joseph Hellerstein Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in google computer clusters. In *ACM SOCC*, 2011.

[29] D. B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *The second annual ACM-SIAM symposium on discrete algorithms*, 1991.

[30] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. *IEEE Symposium on Foundations of Computer Science*, 1991.

[31] Elizabeth Shriver, Arif Merchant, and John Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *ACM SIGMETRICS*, 1998.

[32] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *SC '08*.

[33] ENERGY STAR Program U.S. Environmental Protection Agency. EPA report to congress on server and data center energy efficiency, August 2007.

[34] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *USENIX Annual Technical Conference*, 2009.

[35] Matthew Wachs, Lianghong Xu, Arkady Kanevsky, and Gregory R. Ganger. Exertion-based billing for cloud storage access. In *HotCloud*, 2011.