Performance Evaluation 108 (2017) 32–47

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

A quantitative model of application slow-down in multi-resource shared systems☆

Seung-Hwan Lim [a], Youngjae Kim [b,*]

[a] Computational Data Analytics Group, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

[b] Department of Computer Science and Engineering, Sogang University, 35 Baekbeom-ro, Mapo-gu, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Received 20 August 2015
Received in revised form 9 May 2016
Accepted 24 October 2016
Available online 26 December 2016

Keywords:
Modeling technique
Performance of systems
Measurement

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 extended the D-factor model to capture the slow-down of applications when multiple identical resources exist such as multi-core environments and multi-disks environments. Validation results of the extended D-factor model with HPC checkpoint applications on the parallel file systems show that D-factor accurately captures the slow down of concurrent applications in such environments.

1. Introduction

Sharing a system with multiple workloads enhances system utilization, thereby lowering the economic [1] and environmental dent [2]. Recent advents in server virtualization bolster this sharing policy since it allows multiple operating system instances to share a physical machine [3]. However, sharing a system comes at a price—contention for system

☆ 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. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

* Corresponding author.
E-mail addresses: lims1@ornl.gov (S.-H. Lim), youkim@sogang.ac.kr (Y. Kim).

http://dx.doi.org/10.1016/j.peva.2016.10.004
0166-5316/

resources. Contention for system resources leads to high variance in performance, which would deter hosting latency sensitive applications that require tight performance guarantees [4]. The high variance in performance is often considered as unpredictable performance [4], due to the deficient understanding of performance variation of a workload from contention for multiple resources [5,6]. 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 [7,8] or measurements based solutions [9,10,6,11]. 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 [7], while control theory based approaches use the relationship between allocated amount of resources and workload behavior [8]. However, resource access behavior of a job depends on co-located jobs and specific hardwares, especially when both jobs and machines are heterogeneous [5,12,13]. For instance, co-located jobs can alter cache hit ratio of a job in multi-core environments [5,14] or disk access latencies in shared storage systems [12], 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 [9,10,6,11]. Koh et al. [10] predicted the performance degradation of co-located workloads using recorded similar workloads in terms of their resource usage statistics, similar to program similarity analysis [9]. Recently, Govindan et al. [6] and Mars et al. [11] 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 [15]. 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 [5,16,17]. For example, completing two co-hosted jobs – one 100 s CPU-job with one 100 s I/O-job – will take less than 200 s, 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 [6,11], 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 [6]. 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 [3], with synthetic workloads, FileBench [18], 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 also evaluated the extended D-factor model for multiple instance resource environment. In our experiments, we used HPC checkpoint applications on the Parallel File Systems (PFS), and multiple checkpoint applications shared on the PFS. Our results show that the extended D-factor model can predict I/O performance on the shared storage, which can be useful for I/O performance prediction in the cloud storage environment.

Road-map. 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. Related work is presented in Section 6, followed by concluding remarks in Section 7.

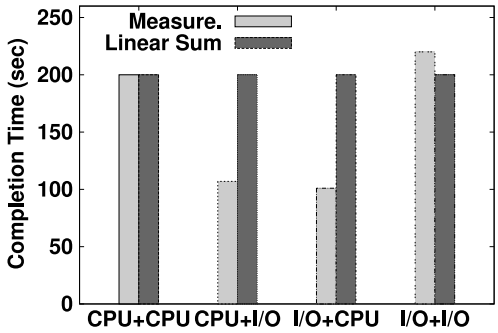


Fig. 1. Non-linearity in slow-down of a job in a multi-resource system. CPU represents a CPU-bound job and IO represents an I/O-bound job. Each bar represents the completion time of job A when job B is colocated. For example, CPU+IO denotes the completion time of the CPU-bound job when the I/O-bound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

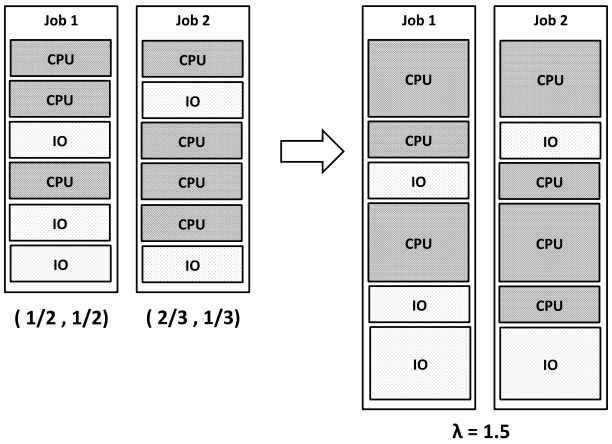


Fig. 2. Dilation factor model describes slow-down of each job when multiple jobs are contending for multiple resources. (Left) Job-slices in their neutral states, where loading vectors are (1/2, 1/2) and (2/3, 1/3), respectively for CPU and I/O. (Right) The processing times of job slices will be dilated 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.

2. Motivation

Interference due to sharing resources results in slow-down of workloads [19], depending on the characteristics of co-located workloads [5,6]. Prior findings have suggested non-linear dilation factors when we consider multiple system resources; co-located applications on multi-core processors [16,17], co-located processes in an operating system [19], co-located virtual machines in a physical machine [20], and co-located workloads in a data center [1].

Fig. 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 2 GB files. Both jobs take 100 s without the presence of other jobs. Experiments are done in a 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 [1].

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 Fig. 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

Fig. 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 5.1. 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 additional 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 resources; 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 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.

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

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

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

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

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

$$\lambda_j = 1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j. \tag{3}$$

Remark 2. 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 Eq. (3),

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*

$$\lambda = n. \tag{4}$$

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,*

$$T = \sum_{j=1}^{n} \tau_j. \tag{5}$$

Proof. Please refer to [21]. □

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

Remark 3. 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 min. 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 min. Depending on the overlap, the total completion time can vary from 1 min to 2 min. But, still the linear model estimate becomes the upper bound of the total completion time in any case.

Remark 4 (*1-Resource-Idling Jobs*)**.** Even if there is 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 $\lambda < n$.

Remark 5 (*Practical Use Case for 1-Resource-Busy Jobs*)**.** This seemingly artificial 1-resource-busy jobs can be found in practice even when the job uses multiple physical resources. For instance, for a job that accesses CPU cores and system memory, the job can be effectively considered as a 1-resource-busy job if the working set entirely fits to the system memory. Workloads that can fit into such a scenario can be (1) iterative algorithms in-memory computing platforms such as Apache Spark [22] and (2) matrix operations when the matrix is entirely populated in system memory.

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

$$\begin{aligned}\lambda_j &= 1 + p_j\overline{p} + (1 - p_j)(n - \overline{p}) - p_j^2 - (1 - p_j)^2 \\ &= 1 + n - \overline{p} - n\,p_j + 2\,p_j\,\overline{p} - p_j^2 - (1 - p_j)^2. \end{aligned} \tag{6}$$

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

$$\lambda_j = 1 + n - 2\,n\,p + 2\,n\,p^2 - p^2 - (1 - p)^2 \tag{7}$$

$$= 1 + n(1 - 2\,p + p^2) + (n - 1)\,p^2 - (1 - p)^2 \tag{8}$$

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

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*

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

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

$$p = \frac{1}{2}\left(1 \pm \sqrt{1 - 2\frac{n - \lambda}{n - 1}}\right). \tag{11}$$

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 11, 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,

$$\overline{\lambda} = \sum_{j=1}^{n} \lambda_j = n + \|\overline{\boldsymbol{p}}\|_2^2 - \sum_{j=1}^{n} \|\boldsymbol{p}_j\|_2^2. \tag{12}$$

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 [21]. 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*

$$\boldsymbol{p}' \cdot \overline{\boldsymbol{p}}_\mu \tag{13}$$

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

3.7. Special case: extended dilation factor model for multiple instances resource environment

The preceding theorems present the theory based upon the assumption that all the computational resources have only a single instance. Although the model works decently in actual systems, this section relaxes the assumption of a single

Table 1
Specifications for experimental environment.

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

instance for each resource in order to achieve higher accuracy in typical real systems. We extend the model with systems, where a computation resource contains multiple identical instances.

3.7.1. Theory formulation

When we have multiple identical instances for a resource, we can consider an n-core processor as a system with n CPU-resources. Similarly, we can treat the m-disk array as a system with m I/O-resources. Suppose each identical instance has independent resources from each other. If we consider CPU and I/O resources in the model, the resulting loading vector for a system with n-core processors and m-disk array can be $n + m$ dimensional—namely, $(CPU_1, \ldots, CPU_n, Disk_1, \ldots, Disk_m)$. Then, we can apply the D-factor model to identify the $n + m$ dimensional loading vector of the job.

Consider a job with loading vector $\boldsymbol{p} = (p, 1-p)$ in 2-resource-busy model. When one computational resource comprises k identical and independent instances, each of k instances will get uniformly distributed loads. Hence, the loading vector can be written by a $k + 1$ dimensional vector of $\boldsymbol{p} = (p/k, p/k, \ldots, p/k, 1 - p)$.

Corollary 1. *Assume n identical jobs and k identical instance of a computational resource. dilation factors are given by:*

$$\lambda = 1 + (n - 1)\left(k * (p/k)^2 + (1 - p)^2\right). \tag{14}$$

Proof. Derivation can be obtained by using Property 1 for $\boldsymbol{p} = (p/k, p/k, \ldots, p/k, 1 - p)$. □

Remark 6. For simplicity, we assumed identical resources are independent from each other and the uniform load of these independent instances. When complex and dynamic interactions exist between those identical resources, this assumption may not hold. In such a case, analysis might require more careful treatment because the system can exhibit serious load imbalances and dependency due to complex and dynamic interactions. However, in many production systems, such load imbalances and dependency problems among resource instances should be typically and reasonably managed with simple scheduling policies such as Round-Robin and FIFO. Thus, the extended D-factor model based upon an assumption that identical instances of a resource are independent will be useful to capture the behavior of many existing systems.

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 [3]. 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. In the VM 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.

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

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 [18]) 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 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 4 GB files on the local disk in the synchronous mode and, then, reads 1 MB 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 2560 files, each of 256 kB size. The neutral completion time of `FileComp` τ is 78.08 s. With `STD-CPU` of the duration of 200 s, the completion time of `FileComp` T is 123.67 s. 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.

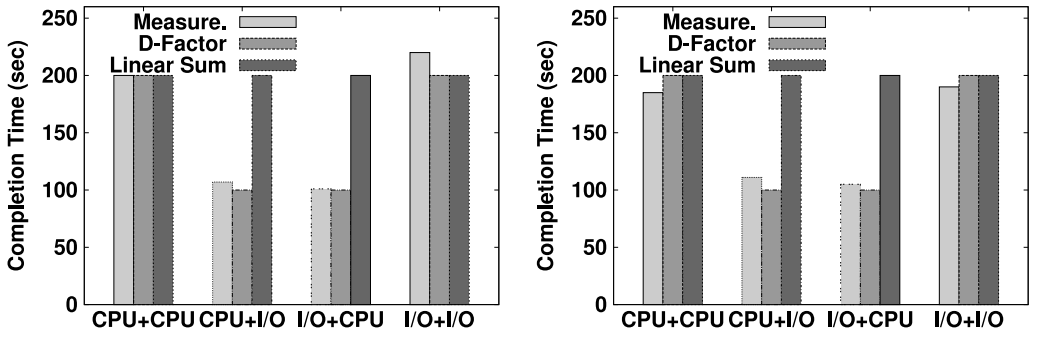
Realistic workloads—FileBench and MapReduce. We experimented with FileBench in a virtualized environment. We employed two predefined workloads in FileBench, `fileserver` and `varmail`. FileBench is an application benchmark that mimics the typical behavior of workloads. We employed one FileBench workload per VM and varied the number of VMs. According to Algorithm 1, both workloads are profiled by

$$\boldsymbol{p}_{file} = (0.02, 0.98) \quad \text{and} \quad \boldsymbol{p}_{mail} = (0.10, 0.90).$$

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 [23]. 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 [23] 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 $\boldsymbol{p}_{Sort} = (0.9, 0.1)$, $\tau_{Sort} = 56.0$ s, $\boldsymbol{p}_{Grep} = (0.5, 0.5)$, $\tau_{Grep} = 95.0$ s, $\boldsymbol{p}_{Pi} = (0.5, 0.5)$, $\tau_{Pi} = 90.0$ s. Then, the loading matrix is given by

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



(a) Native Linux. (b) Virtualized environment.

Fig. 3. 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 represents STD-IO. We hosted two processes in (a) and two VMs in (b).

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 3, 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 separate 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

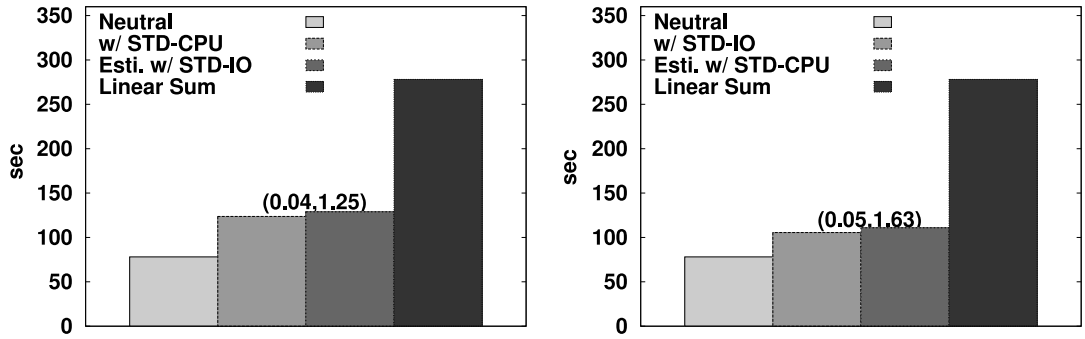
5.1. Evaluating D-factor model for single instance resource environment

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

Standard jobs. As shown in Fig. 3, 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 100 s, the total completion time of two different standard jobs will be 100 s and that of two identical standard jobs will be 200 s. 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.

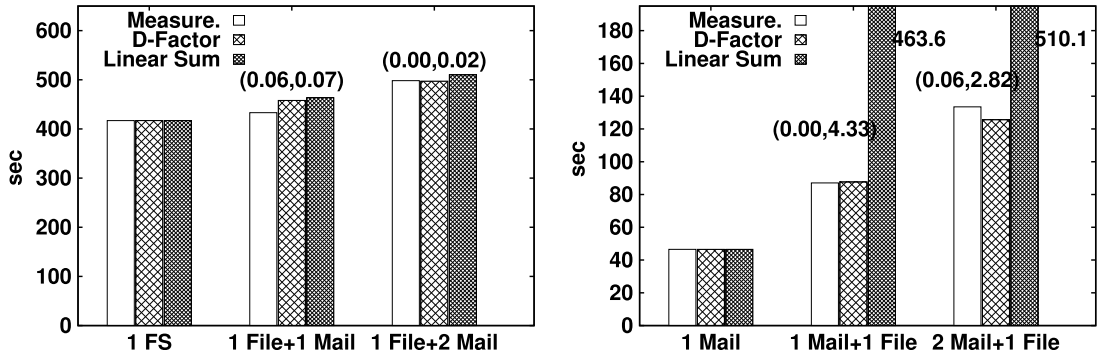
The gist of the dilation factor theorem is that multiple resource contention results in a non-linear waiting time, proportional to their probabilities of accessing the system resources. In contrast, the linear sum model estimates the total completion time as the linear sum of completion times of individual 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.

Fig. 3(a) shows completion time of each co-located workload in the native Linux (two processes) and Fig. 3(b) for the virtualized environment (two VMs). These experiments confirm that D-factor model accurately captures the completion



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

Fig. 4. 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`.

Fig. 5. 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.

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 [24].

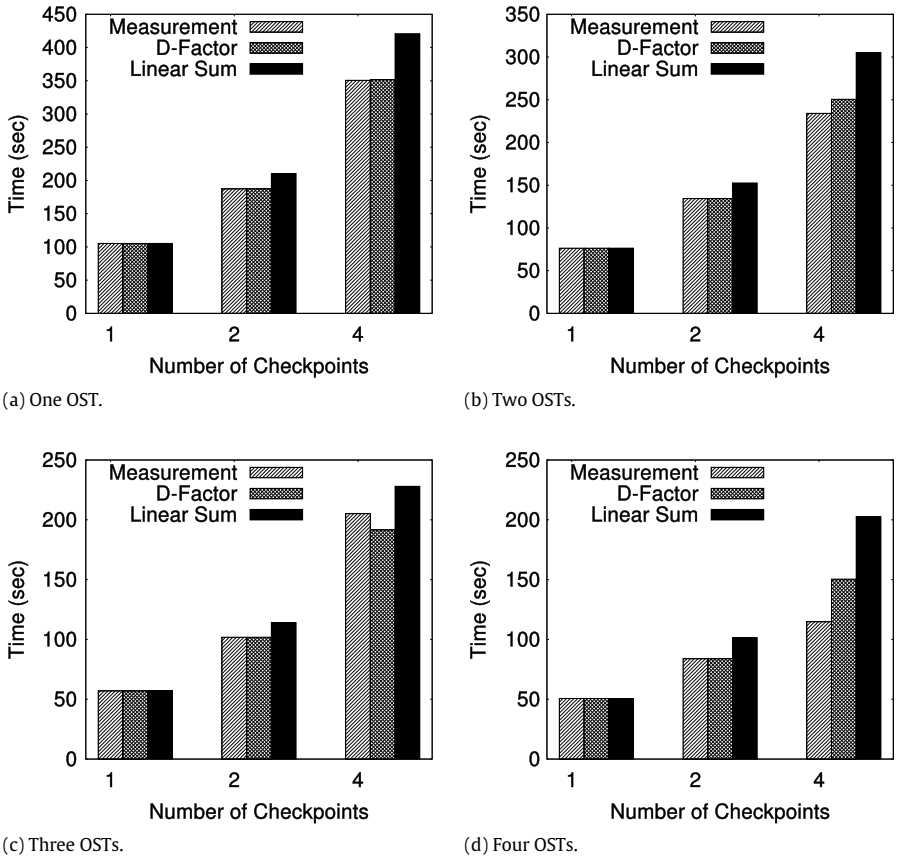
File compression. Now, we demonstrate that the proposed model can estimate the completion time of a workload with both CPU and disk I/O accesses, `FileComp` (refer to Fig. 4). In these experiments, we set the duration of `STD-CPU` and `STD-IO` to 200 s. For the loading vector of `FileComp`, $\boldsymbol{p} = (0.58, 0.42)$, the estimated completion time will be $78.1 \times (1 + 0.42) = 110.9$ s 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 105.50 s, 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.

FileBench in a virtualized environment. Experimental results with FileBench are shown in Fig. 5. 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 458.1 s when we co-locate one `varmail` and one `fileserver` in the same physical machine, compared to 433.1 s with actual measurements. In this situation, we estimate the running time of `varmail` to be 87.62 s, 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 463.6 s.

From Fig. 5, we can observe that for one instance of `fileserver` and two instances of `varmail`, the respective average running times are 133.5 s and 498.3 s. The corresponding average running times from estimation are 133.48 s and 497.2 s. 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.

We also validated the D-factor model by estimating the completion times of three co-hosted MapReduce applications. We found that the D-factor model had 11% error. The details of the co-hosted MapReduce experiments in virtualized environment can be found in [25].



(a) One OST. (b) Two OSTs.

(c) Three OSTs. (d) Four OSTs.

Fig. 6. Experiments with checkpoint applications by increasing the number of disk drives in Luster based file systems. OST denotes an object storage target, which is a logical disk volume. In our configuration, one OST equals one hard drive.

5.2. Evaluating the extended D-factor model for multiple instances resource environment

This section evaluates the extended D-factor model for multiple identical resource instances. We evaluate the model with both macro-level and micro-level settings. As for macro-level setting, we used the shared parallel file systems (PFS) environments. As for micro-level setting, we considered multiple shared channels in Solid State Devices (SSD).

PFS environments can be considered as environments with multiple identical resource instances in the sense that a typical PFS consists of multiple identical disks, represented as an Object Storage Target (OST) in Luster file systems. In other PFS systems, the terms can vary, but they are conceptually similar to Luster file system. SSD also can be treated as an environment with multiple identical resource instances. AN SSD is internally composed of multiple parallel channels, connected to flash packages, each serving as independent I/O storage instance.

5.2.1. Checkpointing applications on the PFS

Let us describe our test-bed for parallel file system experiments. We used a cluster of four compute nodes that communicate with each other and with the disk arrays via InfiniBand (IB) QDR (40 Gb/s). Each node used two 2.40 GHz Intel Xeon CPUs, which has eight processing cores. Each node had 256 GB of DRAM and ran with Linux kernel 2.6.32. As for parallel file system, we used Luster file system with one Object Storage Server (OSS), one Metadata Server (MDS), and 8 OSTs, mounted over 8 SAS 10 K RPM 1 TB drives on each. Each OST was on top of 8 logical volume drives on aforementioned 8 physical drives. Our major consideration in selecting workloads is to effectively validate the model for multiple identical resources. In PFS case, the identical resources are disks, which help us to choose checkpointing application that dominantly uses disks or system stacks related to I/O operations.

Checkpointing process itself is one of the major I/O operations in HPC systems. In HPC, an application is run with multiple MPI processes, and when they checkpoint, each process synchronizes its process status with the PFS, and as all MPI processes generally write to the PFS at barrier and at the same time, thus generating bursty in I/O write patterns onto the PFS. In order to service such bursty I/O patterns, the PFS is comprised of multiple disk drives, thus supporting high bandwidth with parallel I/Os. Internally in the PFS, it stripes file writes at a granularity of an object (1 MB in our setup) in a round robin fashion.

Table 3
Error rates of each estimate (%) with respect to actual measurement, when four checkpoint applications contend for the PFS with the number of disks shown in each case. The error rate is calculated by $\frac{|actual\ measurement - estimate|}{actual\ measurement} \times 100$.

Disk (#) in PFS	1	2	4	8
D-Factor	0.29	7.01	6.62	30.98
Linear sum	20.01	30.37	11.10	76.38

Fig. 6 shows that D-factor provides higher accuracy than linear sum in estimating application slow-down in our parallel file system experiments. Actual measurements were obtained from the average execution time of multiple-please put a number–runs for each test. As the variability was not serious, we did not show the variance of the measurements. We compared the estimated total completion time from the D-factor model and the linear sum model with actual measurements. In the D-factor model, the key is to find a loading vector, λ. The loading vector can be easily calculated using Algorithm 1. In Fig. 6(a), for instance, we calculate a loading vector, $(p_{CPU}, p_{I/O})$, following the procedures in Algorithm 1, which becomes (0.125, 0.875) in this example. Then, we calculate λ with the number of hosts ($n = 4$) using Eq. (10), which is 3.34. Then, we can project an estimated completion time when $n = 4$. Similarly, we can calculate other D-factor estimates for various storage setups in the PFS.

Remark 7. Linear sum will work when resource demand and supply are in equilibrium state. On the other hand, D-factor can work when the resource demand is less than supply. In a case with idle periods in resource usage, the d-factor model inherently works well since each element in a loading vector represents the probabilistic demand of accessing each abstracted resource during its execution. In other words, D-factor model represents the probability of interleavingly accessing multiple resources. Thus, D-factor model will work well in production systems, where resources are typically over-provisioned to prevent the system from excessive resource demand going beyond resource capacity.

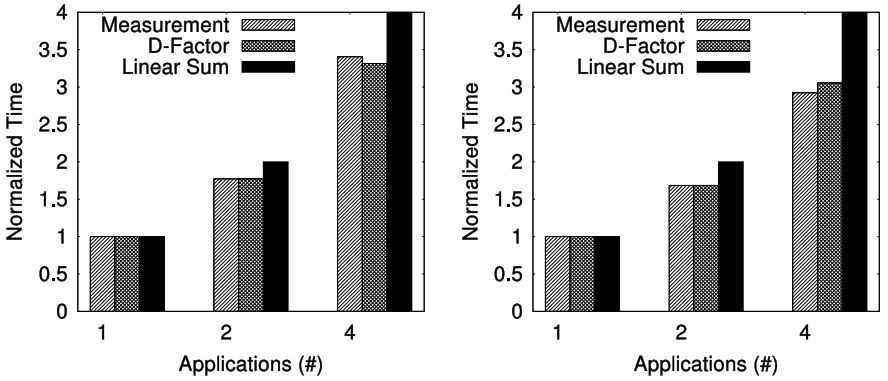
In order to see the impact on D-factor estimates for cases where the shared storage systems become faster with more spindles, we experimented by increasing the number of spindles in the shared storage systems. Then we compared errors calculated from both D-factor model and linear sum with respect to actual measurement Table 3 shows error rates in comparison for both models. We notice that D-factor normally shows lower error rates than linear sum. It can be explained that the D-factor model effectively captures the effect from resource-bubbles. It makes D-factor model more accurate than linear sum model. We might be able to do the same modeling with queuing models. However, D-factor can model this phenomenon with less labors in obtaining parameters. However, we also observe that the D-factor error rate tends to increase for larger number of disks. Current D-factor model only captures two resource models. It is limited that some part of each probability in a loading vector does not capture 100% of its resource utilization. We see this in part of our future work, to enhance the D-factor model for such multiple resources environment.

5.2.2. SSD for multiple identical applications

Unlike Hard Disk Drive, SSD is organized in multiple, internal flash components to appear as a single block of I/O device to the host computer. The SSD is composed of multiple flash memory *Planes* and the planes are implemented in multiple dies connected with multiple independent channels. Across the dies, data pages are striped at a fixed size of a stripe. [26] Thus, SSD can also be regarded as a resource with multiple instances. Moreover, SSD shows slower write times than reads, and random writes can become worse in performance than sequential writes, because internal garbage collection (GC) process can hinder foreground I/O service, thus, increasing I/O response times during GC [27]. With these unique characteristics of SSD, we evaluate the D-factor model for different applications' I/O patterns when they complete for an SSD. We used a modern SATA-III (6 Gb/s for interface) SSD with 550 MB/s and 370 MB/s for read and write [28] to evaluate co-located applications' run times on the SSD for actual measurement and mode-based estimates.

For Fig. 7, we used write-intensive applications with two different types of workloads and showed normalized run-times while we increase the number of co-located applications on an SSD. For an application's I/O characteristics, we evaluated both write and read dominant workloads. In HPC, such 80% write is common in applications because of checkpoints [29], whereas in some HPC applications such visualization applications are read dominant [30]. For both workloads, we observe that D-factor model shows close estimate to actual measurement with a more accurate estimate than linear sum model. Comparing error rates for D-factor and linear sum models with respect to actual measurement, we can see that the D-factor model's error rate is less than 5% error whereas the linear sum model can reach at about 20% in error at the worst.

To summarize, through experiments, we demonstrated that the D-factor model estimates the execution time of individual jobs when they share multiple system resources. We also demonstrated that the D-factor model effectively describes the behavior of resources with multiple identical instances since the model is a high-level statistical abstraction of system behaviors. Experiments in this section suggest that we may predict I/O performance for cloud services [4] using the D-factor model. We have further studied the application of the D-factor model for building a scheduling algorithm to enhance overall system performance. The details of this study can be found in our prior work [25].



(a) 80% write and 20% read.

(b) 20% write and 80% read.

Fig. 7. Experiments with I/O dominant applications on SSD.

6. 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 environments [11,6]. 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 [5] and a critical problem in Cloud platforms [4].

As an attempt to estimate dilation factor, Govindan et al. [6] and Mars et al. [11] 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 [11,6] 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 [31]. 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 [32–34] or similar linear relations [35], 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 [1,36]. In addition, the issue raised from I/O-bound work in [4] 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.

7. 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 in processing 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. the D-factor model only needs 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 of 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. We extended D-factor model presented in [25] for the cases where a resource has multiple instances, which can be useful for multi-core environments or disk-arrays. We demonstrated

that the D-factor model can better estimate the slow-down of applications than linear sum model for the extended cases.

Acknowledgments

We would like to thank the anonymous reviewers for their detailed comments, which helped us improve the quality of this paper. This work was supported in part by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea Government (MSIP) (No. R0190-15-2012) and by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MISP) (No. 2015R1C1A1A0152105). The work was also supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the US DOE (under the contract No. DE-AC05-00OR22725).

References

[1] X. Fan, W.-D. Weber, L.A. Barrose, Power provisioning for a warehouse-sized computer, in: ISCA'07, 2007.
[2] US Environmental Protection Agency, E. P. EPA Report to Congress on Server and Data Center Energy Efficiency., August 2007.
[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: SOSP, 2003.
[4] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010).
[5] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, S. Reinhardt, Qos policies and architecture for cache/memory in cmp platforms, in: ACM SIGMETRICS, 2007.
[6] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam, Cuanta: Quantifying effects of shared on-chip resource interference for consolidated vitual machines, in: ACM SOCC, 2011.
[7] D. Menasce, Two-level iterative queuing modeling of software contention, in: 10th IEEE MASCOTS, 2002.
[8] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for qos-aware clouds, in: Eurosys, 2010.
[9] K. Hoste, A. Phansalkar, L. Eeckout, A. Georges, L.K. John, K. De Bosschere, Performance prediction based on inherent program similarity, in: PACT, 2006.
[10] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, C. Pu, An analysis of performance interference effects in virtual environments, in: ISPASS, 2007.
[11] J. Mars, L. Tang, R. Hundt, K. Skadron, M.L. Soffa, Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations, in: Micro, 2011.
[12] M. Wachs, L. Xu, A. Kanevsky, G.R. Ganger, Exertion-based billing for cloud storage access, in: HotCloud, 2011.
[13] B. Sharma, V. Chudnovsky, J.H. Rasekh Rifaat, C.R. Das, Modeling and synthesizing task placement constraints in google computer clusters, in: ACM SOCC, 2011.
[14] A. Sharifi, S. Srikantaiah, A.K. Mishra, M. Kandemir, C.R. Das, Mete: meeting end-to-end qos in multicores through system-wide resource management, in: ACM SIGMETRICS, 2011.
[15] D. Shmoys, J. Wein, D. Williamson, Scheduling parallel machines on-line, in: IEEE Symposium on Foundations of Computer Science, 1991.
[16] R. Bitirgen, E. Ipek, J.F. Martinez, Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach, in: MICRO, 2008.
[17] A. Fedorova, S. Blagodurov, S. Zhuravlev, Managing contention for shared resources on multicore processors, Commun. ACM (2010).
[18] Filebench, Filebench, 2014. http://filebench.sourceforge.net/wiki/index.php/Main_Page.
[19] P.J. Denning, The working set model for program behavior, Commun. ACM 11 (5) (1968) 323–333.
[20] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing SLA violations, in: 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007.
[21] S.-H. Lim, J.-S. Huh, Y. Kim, G.M. Shipman, C.R. Das, A quantitative analysis of performance of shared service systems with multiple resource contention. Tech. Rep. CSE-10-010, The Pennsylvania State University, 2010.
[22] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA, 2010, 7 pages. URL http://dl.acm.org/citation.cfm?id=1863103.1863113.
[23] The Apache Software Foundation, Apache Hadoop, 2015. http://hadoop.apache.org.
[24] E. Shriver, A. Merchant, J. Wilkes, An analytic behavior model for disk drives with readahead caches and request reordering, in: ACM SIGMETRICS,1998.
[25] S.-H. Lim, J.-S. Huh, Y. Kim, G.M. Shipman, C.R. Das, D-factor: A quantitative model of application slow-down in multi-resource shared systems, in: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS'12, ACM, New York, NY, USA, 2012, pp. 271–282. URL http://doi.acm.org/10.1145/2254756.2254790.
[26] L.-P. Chang, T.-Y. Chou, L.-C. Huang, An adaptive, low-cost wear-leveling algorithm for multichannel solid-state disks, ACM Trans. Embedded Comput. Syst. 13 (3) (2013) 55:1–55:26. URL http://doi.acm.org/10.1145/2539036.2539051.
[27] Y. Kim, J. Lee, S. Oral, D. Dillow, F. Wang, G.M. Shipman, Coordinating garbage collectionfor arrays of solid-state drives, IEEE Trans. Comput. 63 (4) (2014) 888–901. URL http://doi.ieeecomputersociety.org/10.1109/TC.2012.256.
[28] Sandisk, 2014. Sandisk Optimus SSD. http://www.sandisk.com/enterprise/sas-ssd/optimus-ssd/.
[29] Y. Kim, R. Gunasakaren, Understanding i/o characteristics of a peta-scale storage system, J. Supercomput. 71 (3) (2015) 761–780.
[30] P.H. Carns, R. Latham, R.B. Ross, K. Iskra, S. Lang, K. Riley, 24/7 characterization of petascale I/O workloads, in: Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31–September 4, 2009, IEEE Computer Society, New Orleans, Louisiana, USA, 2009, pp. 1–10. http://dx.doi.org/10.1109/CLUSTR.2009.5289150.
[31] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation Algorithms for Bin Packing: A Survey, PWS Publishing Co., Boston, MA, USA, 1997.
[32] A. Singh, M. Korupolu, D. Mohapatra, Server-storage virtualization: integration and load balancing in data centers, in: SC'08, 2008.
[33] A. Verma, G. Dasgupta, T.K. Nayak, P. De, R. Kothari, Server workload analysis for power minimization using consolidation, in: USENIX Annual Technical Conference, 2009.
[34] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: INFOCOM, 2010.
[35] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: VEE, 2009.
[36] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, D. Pendarakis, Efficient resource provisioning in compute clouds via vm multiplexing, in: ICAC, 2010.

Seung-Hwan Lim received a Ph.D. in Computer Science and Engineering from the Pennsylvania State University in 2012. Since 2012, He has been affiliated with Oak Ridge National Laboratory as a research staff in computational data analytics group. His research topic focuses on parallel and distributed systems. He has worked on optimizing the performance of data analytic platforms like Hadoop, Spark, relational databases, and NoSQL databases with the emphasis on machine learning and graph analysis.

Youngjae Kim received the B.S. degree in computer science from Sogang University, Korea in 2001, the M.S. degree from KAIST in 2003 and the Ph.D. degree in computer science and engineering from Pennsylvania State University in 2009. He worked as a research staff member in 2009–2015 at the Oak Ridge National Laboratory and as an assistant professor in the Department of Software and Computer Engineering at Ajou University in 2015–2016. Since 2016, he is with the Department of Computer Science and Engineering at Sogang University as an assistant professor. His research interests include distributed file and storage, parallel I/O, operating systems, emerging storage technologies, and performance evaluation.