TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments

Ron C. Chiang H. Howie Huang
Department of Electrical and Computer Engineering
George Washington University
{rclc, howie}@gwu.edu

ABSTRACT

Large-scale data centers leverage virtualization technology to achieve excellent resource utilization, scalability, and high availability. Ideally, the performance of an application running inside a virtual machine (VM) shall be independent of co-located applications and VMs that share the physical machine. However, adverse interference effects exist and are especially severe for data-intensive applications in such virtualized environments. In this work, we present TRACON, a novel Task and Resource Allocation CONtrol framework that mitigates the interference effects from concurrent data-intensive applications and greatly improves the overall system performance. TRACON utilizes modeling and control techniques from statistical machine learning and consists of three major components: the interference prediction model that infers application performance from resource consumption observed from different VMs, the interference-aware scheduler that is designed to utilize the model for effective resource management, and the task and resource monitor that collects application characteristics at the runtime for model adaption. We simulate TRACON with a wide variety of data-intensive applications including bioinformatics, data mining, video processing, email and web servers, etc. The evaluation results show that TRACON can achieve up to 50% improvement on application runtime, and up to 80% on I/O throughput for data-intensive applications in virtualized data centers.

1. INTRODUCTION

Cloud computing has achieved tremendous success in offering Infrastructure/Platform/Software as a Service, in an on-demand fashion, to a large number of clients. This is evident in the popularity of cloud software services, e.g., Gmail and Facebook, and the rapid development of cloud platforms, e.g., Amazon EC2. The key enabling factor for cloud computing is the virtualization technology, e.g., Xen [4], that provides an abstraction layer on top of the underlying physical resources and allows multiple operating systems and applications to simultaneously run on the same hardware. As virtual machine monitors (VMM) encapsulate different applications into each separate guest virtual machine (VM), a cloud provider can leverage VM consolidation and migration to achieve excellent resource utilization and high availability in large data centers.

Ideally, an application running inside a VM shall achieve the performance as it would own a portion of the machine to itself, that is, independent of co-located applications and VMs that share the same physical resource. Although extensive work has been done to achieve this so-called performance isolation [21, 25, 29], including various techniques to ensure CPU fair sharing [20], little attention has been paid to data-intensive applications that perform complex analytics tasks on a large amount of data, which have become increasingly common in this environment [13]. Traditionally assuming the exclusive ownership of the physical resources, these applications are optimized for the hard drive based storage systems by issuing large sequential reads and writes. However, this assumption breaks down in a shared, virtualized environment, and subsequently the previously optimized I/O requests are no longer advantageous. To the contrary, multiple data-intensive applications will be in competition for the limited bandwidth and throughput to network and storage systems, which very likely leads to high I/O interference and low performance. In this case, the combined effects from concurrent applications, when deployed on shared resources, are largely difficult to predict, and the interference as a result of competing I/Os remains problematic to achieve high-performance computing in a virtualized environment.

In this work, we study the performance effects of co-located data-intensive applications, and develop TRACON[1], a novel Task and Resource Allocation CONtrol framework that mitigates the interference from concurrent applications. TRACON leverages modeling and control techniques from statistical machine learning and acts as the core management scheme for a virtualized environment. The evaluation shows that TRACON can achieve up to 50% improvement on application runtime and up to 80% on I/O throughput for data-intensive applications.

[1]In aviation, TRACON stands for Terminal Radar Approach Control facilities, "for example, Potomac TRACON handles air traffic going into and out of all the airports around Washington D.C., Baltimore, MD, and Richmond, VA." Source: http://www.faa.gov.

The main contributions of our work are two-fold:

- We characterize the I/O interference from multiple concurrent data-intensive applications in a virtualized environment, and build interference prediction models that reason about application performance in the event of varied levels of I/O interference. While prior work has extensively studied the VM interference on CPUs, caches, and main memory [21, 22, 29], we address the new challenges that arise when modeling data-intensive applications in virtualized data centers. Our statistical models profile the performance of a target application, when running against a set of benchmarks, to infer both the runtime and I/O throughput of the application. We propose to employ nonlinear models that are critical to capture the bursty I/O patterns in data-intensive applications. Our models can adapt in the runtime when it is detected that they no longer accurately model the application's performance. This design utilizes the application characteristics, observed from the VMs, and maintains a low system overhead.
- We develop a management system TRACON for a virtualized data center to mitigate the interference effects from co-located data-intensive applications. To achieve this goal, we incorporate the proposed nonlinear interference prediction models into TRACON and by doing so, the system can make optimized scheduling decisions that lead to significant improvements in both application performance and resource utilization. We conduct a comprehensive set of experiments on different combinations of data-intensive applications.

The rest of the paper is organized as follows. The next section introduces the virtualization technology and illustrates the performance degradation caused by I/O interference. Section 3 introduces TRACON system architecture, and presents the modeling and scheduling techniques for interference reduction. The evaluation setup and results are presented in Section 4. We conclude in Section 5.

2. BACKGROUND

Virtualized data centers are the most common cloud computing platforms. In this work, we focus on Xen and its notable paravirtualization technique, where the Xen VMM works as a hardware abstraction layer to guest operating systems with the modified kernels. Note that Xen also supports hardware-assisted full virtualization that emulates the host hardware for unmodified operating systems. In paravirtualization, the VMM is in charge of resource control and management, including CPU time scheduling, routing hardware interrupt events, allocating memory space, etc. In addition, a driver domain (*Dom*0) that has the native drivers of host hardware performs the I/O operations on behalf of guest domains (*DomU*). Fig. 1 depicts a typical Xen I/O architecture, where each guest domain uses a virtual device driver (frontend) in cooperation with a corresponding module (backend) and the native driver in the driver domain to accomplish I/O operations.

2.1 VM Interference

When multiple VMs are running on the same physical machine, severals factors contribute to the degraded application performance, including virtualization overheads and the imperfect performance isolation between VMs. As cloud applications become increasingly data centric, we shall address additional challenges of the I/O interference when running data-intensive applications in such virtualized environments.

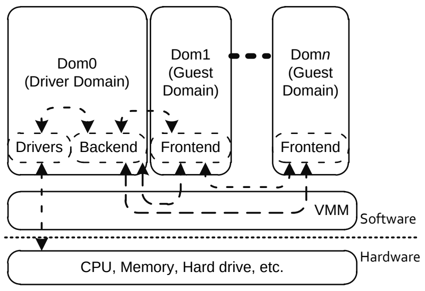


Figure 1: Xen I/O architecture. Solid lines represent the I/O channels for data transmission. Host hardware interrupts are controlled and routed through VMM and depicted as dashed lines

In the following example, we illustrate this problem using the data collected on a machine managed by Xen, where two applications, *App*1 and *App*2, run in two separate VMs, $VM1$ and $VM2$, respectively. The evaluation environment will be described in detail in Section 4. In the first scenario (the first row in Table 1), App1 is a CPU-intensive program (Calc) that performs algorithmic calculations in VM1 and its runtimes are measured when program in VM2 is a CPU-intensive, data-intensive, both CPU- and data-intensive, or CPU- and data-moderate program. We normalize App1 runtimes to that of App1 running alone, that is, normalize to without interference. In the second scenario (the second row in Table 1), App1 is a data-intensive program (SeqRead) that sequentially reads a large file.

Table 1: Normalized App1 runtime in VM1 while running various App2 in VM2

App2 / App1	CPU High	I/O High	CPU & I/O Medium	CPU & I/O High
Calc	1.96	1.26	1.77	2.52
SeqRead	1.03	10.23	1.78	16.11

As illustrated in Table 1 , while two CPU-intensive applications show performance slowdowns, the result is not unexpected. In this case, because both VMs are multiplexed on the same CPU, the runtime of App1 is simply doubled due to Xen's fair credit scheduling. However, for a data-intensive program, performance interference is much more severe and unpredictable. For example, while App1(SeqRead) experiences little change in performance when App2 has a heavy CPU consumption, it slows down by 10 times when App2 is performing a similar task and competing for I/O devices. Furthermore, because the driver domain handles I/O operations on behalf of unprivileged guest domains [25, 31], the interference can become even more severe when App2 demands both CPU and I/O resources - App1 is 16 times slower in this case. Clearly, a deep understanding of I/O interference is crucial for virtualized data centers to improve customer experiences and increase resource utilization. In this work, we build interference prediction models that automatically infer the effects when there exist varied levels of concurrent I/O operations, and propose novel scheduling algorithms to minimize the negative impacts of co-located applications.

2.2 The Challenges with Existing Techniques

Traditional performance modeling [38, 34, 28] and scheduling techniques [7, 33, 36] focus on the computation-intensive applications and model CPU utilization and performance. Extensive work has been done to characterize and predict the I/O performance, mostly in a non-virtualized environment [26, 32, 16].

For VM performance modeling, Wood et al. [37] measure and use application characteristics to model the virtualization overheads, and Kundu et al. [22] propose an iterative model training technique based on artificial neural networks to build models for predicting application performance in virtualized environments. Similarly, Mei et al. [25] and Pu et al. [31] study the network traffic interference in virtualized cloud environments. However, it is unclear how the above work can be utilized to mitigate the I/O interference for data-intensive applications.

Our work is closely related to Q-Clouds [29] and *pSciMapper* [39]. Q-Clouds utilizes online feedback to build a multi-input multi-output (MIMO) model to capture the performance interference, and to tune resource allocations to mitigate the performance interference. However, Q-Clouds focuses on the CPU bound workload. *pSciMapper* is a power-aware consolidation framework for scientific workflows and builds the models to relate the resource requirements to performance and power consumption. Note that energy-aware scheduling becomes increasingly important in data centers [27], as the total energy used by the servers is estimated to approach 3% of US electricity consumption [35]. In comparison, TRACON goes further by focusing on the performance improvement of the whole system. Moreover, our work investigates data-intensive scientific workflows and demonstrates the ability to be used in a large scale system under heavy workloads.

3. TRACON SYSTEM ARCHITECTURE

Most cloud service providers utilize a hierarchical management scheme to administer the large quantity of machines [2]. In this environment, each application server has a similar virtualized environment as shown in Fig. 1, where VMs are dynamically allocated to run the applications from the clients. A manager server is responsible for supervising a group of the application servers, which report their status in a time interval to the manager server. The manager servers can form a tree-like hierarchy for high scalability.

Given a virtualized environment that consists of a large number of physical machines and different applications, we utilize statistical machine learning techniques, in particular statistical modeling for reasoning about the application's performance under interference. We share the same philosophy as in [6, 23] that the statistical machine learning will play an important role in the application and resource management in large-scale data centers.

As the core management scheme for a virtualized environment, TRACON, our Task and Resource Allocation CONtrol framework consists of three major components: 1) the interference prediction model infers the application performance from the resource consumption observed from multiple VMs; 2) the interference-aware scheduler is designed to utilize the model and generate optimized assignments of tasks and physical resources; and 3) the task and resource monitor that collects application characteristics at the runtime and feed to both the model and scheduler. Fig. 2 presents the TRACON architecture and the interactions between different components.

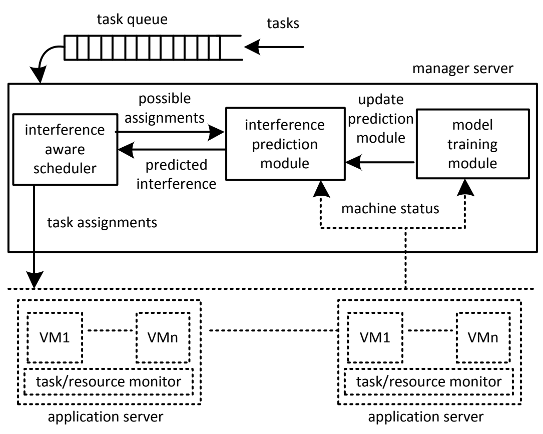


Figure 2: TRACON system architecture

Upon the arrival of the tasks, the scheduler will generate a number of possible assignments based on the incoming tasks and the list of available VMs, which will then be communicated to the interference prediction module. This module uses the constructed models, the application profiles, and the machine status to predict the interference effects for the given assignments. Finally, depending on the predictions, the scheduler makes the scheduling decision and assigns the tasks to different servers.

On the application servers, the task and resource monitors will manage the on-going tasks that is assigned to each VM and collect application characteristics and the performance interference. In a simple example where two VMs are to share the host hardware, the monitor measures the resource utilization of both VMs via existing system tools (e.g., *xentop* and *iostat*) while the assigned tasks are running. Xentop is used to monitor and record the physical CPU utilization of each domain, because a top command in each domain, including the driver domain, can only get the domain's virtual CPU utilization. In addition, because Dom0 performs the I/O operations on behalf of each guest domain, we use iostat in Dom0 to monitor and record the resource utilization of physical storage devices. Note that the resource utilization monitoring in Dom0 is preferable because system administrators generally do not have access to each guest domain for security and privacy reasons.

3.1 Interference Prediction Model

On the high level, the interference can be perceived as the changes in the performance, including the total runtime as used in prior work [38, 28], and I/O throughput that we have shown is critical to data-intensive applications. In this work, we construct the interference prediction models in order to extrapolate the application performance as a function of the resource consumption by virtual machines. We apply several regression analysis techniques that are commonly used for modeling the relationship between an observed response and controlled variables [9].

For the data-intensive applications that consume a significant amount of I/O bandwidth and CPU cycles, we choose to characterize each application by four key parameters (controlled variables), listed in Table 2: the read throughput, the write throughput, the local CPU utilization in the current guest VM domain (DomU), and the global CPU utilization in the virtual machine monitor (Dom0). The first two parameters measure the I/O workload from the target application in terms of the number of requests per second, and the third is used to model the CPU consumption from data processing of the application. While a model with these three parameters is a straightforward approach to reason about an application, such a model is not sufficient to achieve high accuracy for a virtualized environment, which is the reason that we introduce the fourth parameter, the global CPU utilization in the virtual machine monitor. Intuitively, because all the requests from guest VMs are routed through Dom0, it is crucial to properly account for the CPU consumption from the I/O handling tasks that are performed in DomU, as well as in Dom0 that acts on physical devices on behalf of DomU. If such I/O overheads on CPU utilization were ignored, the prediction models would produce significantly larger errors, as we will show in Section 4.

Table 2: Application Characteristics

CPU	*I/O*
Local utilization in DomU	Read requests per second
Global utilization in Dom0	Write requests per second

In our model, all four parameters can be easily collected with the help of the TRACON task and resource monitor. This approach leverages low-overhead system tools and reduces unnecessary interference from system monitoring. We propose this simple approach to avoid additional changes to a state-of-art virtualized environment (OS and virtual machine kernels), which we believe can lead to a wide adoption in current production systems.

In the following, we present three types of models, the weighted mean method (WMM), the linear model (LM), and the non-linear model (NLM), for two different responses - application runtime and I/O throughput. For simplicity, we assume that each physical machine supports two virtual machines (VM1 and VM2), each can be assigned with one application. For each model, each of these responses relates to four key parameters for each VM, that is, eight variables in total, which are the application characteristics in both virtual machines. For each model, Y (runtime or IOPS) is the response variable, while $X_{VM1,i}$ and $X_{VM2,i}$ where $i \in \{1, 2, 3, 4\}$, the application characteristics on VM1 and VM2, are the controlled variables.

Weighted mean method is based on the principal component analysis (PCA) [18] that utilizes applied linear algebra to the data set in order to produce a set of uncorrelated variables, i.e., the principal components that capture most important dynamics in the data. The assumption is that because the principal components account for most data variances, chances are that they are a good representation of the data while other variables with smaller variances mostly contribute to the noise and redundancy. As such, the PCA is commonly used to deal with complex data sets, as well as to reduce the high dimensionality in the modeling process.

Our WMM model similar to [21] calculates Euclidean distances between the data points in the space spanned by the first four principle components. Next, it chooses three nearest data points and uses the reciprocal of their distances as the weights to get the predicted response. In this work, we use WMM as the baseline when evaluating the linear and non-linear models.

Linear models assume there is a linear relationship between the response variable and controlled variables, which can be formally presented as

$$\hat{Y} = c + \sum_{i=1}^{4} \alpha_i \cdot X_{VM1,i} + \sum_{i=1}^{4} \beta_i \cdot X_{VM2,i} \qquad (1)$$

where α_i and β_i are coefficients and c is a constant. The error is defined as the difference between the real and expected value, i.e., $\left|Y - \hat{Y}\right|$, and the sum of squared errors (SSE) is calculated as $\sum(Y - \hat{Y})^2$.

To obtain a linear model with high prediction accuracy, one needs to search for a good combination of the constant c and coefficients α_i and β_i such that SSE is minimized. However, in our case, a prediction model that consists of all eight parameters for two VMs may not necessarily have the best fit to the observed data. In other words, a model with fewer inputs may have a higher or equivalent prediction accuracy than a more complex one. To this end, we use a stepwise algorithm [14] that adds or removes possible variables one at a time into the model fitting process. After re-fitting the new model, the algorithm will evaluate the model's goodness of fit. The process continues iteratively and in the end outputs the best model among all the candidates. For comparison, the algorithm needs a good metric to measure a model's goodness of fit. Although the goal of a modeling process is to minimize SSE, it is insufficient to simply use SSEs because of the trade-off between accuracy and flexibility [9].

To this end, we utilize the concept of Akaike information criterion (AIC) [1] that is based on the information theory to provide scores for evaluating such a trade-off. By definition, AIC can be described as $(-2) \times \log_e$ (*maximum likelihood*) + 2 × (*number of parameters*), which describes the quality of a model with regard to the parameters that are selected by the maximum likelihood method. Note that a lower value of AIC indicates a better model. In this work, we use the stepwise algorithm with AIC as the scoring function to help select a linear model.

Nonlinear models: Our assessment of the linear models reveals that while their prediction accuracy is mostly in par with the weighted mean method, these models cannot be considered as a good fit of the observed data. As we focus on data-intensive applications in this work, the bursty I/O patterns in such applications [15, 10] tend to make the linearity no longer hold and lead to large prediction errors. The need for an alternative model to both the weighted mean method and linear models leads us to explore nonlinear models, in particular with the degree of two, i.e., quadratic models in our study.

By expanding the controlled variables $X_{VM1,i}$ and $X_{VM2,i}$ to all the terms in the expansion of the degree-2 polynomial $(1 + \sum_{i=1}^{4} X_{VM1,i} + \sum_{i=1}^{4} X_{VM2,i})^2$, we can construct an initial non-linear function of the controlled variables for the regression as equation (2).

In the non-linear modeling process, we use the Gauss-Newton method [11] to find the coefficients such that SSE is minimized. The Gauss-Newton method is an iterative process that gradually updates the parameters to obtain the optimal solution. Similarly, we use a stepwise algorithm to choose a non-linear model with the best AIC value. In general, we find that nonlinear models have the best prediction accuracy compared to the other two methods in predicting either the runtime or IOPS, as will be shown in Section 4.

$$\begin{aligned}\hat{Y} \;=\; & c + \sum_{i=1}^{4} \alpha_i^{(1)} \cdot X_{VM1,i} + \sum_{i=1}^{4} \alpha_i^{(2)} \cdot X_{VM2,i} + \\ & \sum_{i=1}^{4}\sum_{j=1}^{4} \beta_{i,j}^{(1)} \cdot X_{VM1,i} \cdot X_{VM2,j} + \\ & \sum_{i=1}^{4}\sum_{j=1}^{i-1} \beta_{i,j}^{(2)} \cdot X_{VM1,i} \cdot X_{VM1,j} + \\ & \sum_{i=1}^{4}\sum_{j=1}^{i-1} \beta_{i,j}^{(3)} \cdot X_{VM2,i} \cdot X_{VM2,j} + \\ & \sum_{i=1}^{4} \gamma_i^{(1)} \cdot X_{VM1,i}^2 + \sum_{i=1}^{4} \gamma_i^{(2)} \cdot X_{VM2,i}^2 \qquad (2)\end{aligned}$$

Model training and learning: For a given application, we generate its interference profile by running it on VM1 while varying the workloads on VM2, for which we develop a workload generator to exercise both CPU and I/O devices with different intensities. By doing so, we obtain a collection of data on interference effects under different background workloads. For the CPU utilization, the workload generator executes a set of arithmetic operations in a loop with varied idle intervals between each iteration so as to the CPU utilization in a guest domain can be controlled in five different intensities ranging from 0%, 25%, 50%, 75% to 100%. In the meantime, a storage device is tasked with either read or write requests. In both cases, the workload generator reads from or writes to a file. The file is much larger than the allocated memory size of the guest domain to avoid OS caching. Similarly, the read requests per second and write requests per second can also be controlled in five different intensities ranging from 0% to 100% by adjusting the length of sleep interval between each iteration. For the purpose of creating more realistic scenarios, we create in total 125 of different workloads that serve as the background applications in profiling the interference. Note that we also include the performance for each application without interference, that is, the application runs in one VM while the other VM is idle.

For a cloud platform, our approach can be simply automated when a new application comes in. Further, this approach supports online learning of the interference prediction model, that is, the model shall be dynamically monitored and modified when it cannot accurately capture the interference relationships among different applications. The causes may come from the changes in applications, virtual machines, operating systems, and cloud infrastructures. To this end, TRACON collects statistics on applications and virtual machines and keeps tracks of the prediction errors of the models. Upon the occurrence of some predefined events (e.g., a significant shift of the mean or a large surge in the variance), TRACON will start to build a new model with the latest data.

3.2 Interference-Aware Scheduling

Interference prediction completes one side of the story - with the help of these models, TRACON can now schedule the incoming tasks to different virtual machines in a way that minimizes the interference effects from co-located applications. In general, optimally mapping tasks to machines in parallel and distributed computing environments has been shown to be an NP-complete problem [12]. In this work, we explore a number of heuristic techniques to find a good solution for the scheduling problem. Specifically, TRACON aims to reduce the runtime and improve the I/O throughput for data-intensive applications in a virtualized environment. Given a set of tasks T and each task $t \in T$ has the runtime of RT_t and the I/O throughput of $IOPS_t$, we define the total runtime RT_{total} for this set of tasks as

$$RT_{total} = \sum_{\forall t \in T} RT_t \qquad (3)$$

and the combined throughput $IOPS_{total}$ as

$$IOPS_{total} = \sum_{\forall t \in T} IOPS_t \qquad (4)$$

In this work, we explore three different scheduling strategies: online scheduling that reduces the queueing time for each incoming task by quickly dispatching them to various virtual machines, batch scheduling that pairs the incoming tasks based on the predicted interference, and mixed scheduling that aims to balance between both batch and online scheduling. For comparison, we use an FIFO scheduler as the baseline where the incoming tasks are allocated to virtual machines in a first-in, first-out order.

Minimum interference online scheduler (MIOS) is designed to make a quick scheduling decision that becomes necessary when the tasks arrive at a rapid speed. In such a scenario, the tasks will arrive at the queue at arbitrary times and the scheduler will dispatch an incoming task immediately without waiting for later tasks. We design MIOS based on the concept of the minimum completion time (MCT) heuristic [8]. With the goal of minimizing the sum of execution times of all tasks, MCT maps each incoming task to the machine that completes the task in the shortest time. When a task t arrives, MIOS will predict t's performance on each available VM, and assign t to a VM with the best predicted performance. The advantage of MIOS is the ability to dispatch a task in a short time. On the downside, the task assignment may not be better than a batch scheduler that considers more possible assignments. The MIOS algorithm is presented in Algorithm 1.

Algorithm 1: MIOS

```
Data: Task $t_i$;
      $Pool$ consists of $VM_{j,k}$, where $j \in 1,\ldots,m$, and
      $k \in 1,\ldots,n$ ;
      $Model$ is the interference prediction model.
Result: $t$ and $VM_{j,k}$ assignments.
begin
    for each $VM_{j,k}$ in the $Pool$ do
        $score_i$ = Predict($t$, $VM_{j,k}$, $Model$);
    end
    $VM_{candidate}$ = Min($score_i$);
    Assign($t$, $VM_{candidate}$);
end
```

Minimum Interference Batch Scheduler (MIBS) is a batch scheduling algorithm based on the concept of the Min-Min heuristic [17]. In a batch scheduling scenario, the scheduling process takes place when the queue that holds the incoming tasks is full. In the first step, the Min-Min heuristic finds a machine with the minimum score (e.g., completion time) for each task on the queue (the first "Min"). In the second step, among all task-machine pairs, Min-Min finds the pair with the minimum score (the second "Min"), and assigns the selected task to its corresponding machine. This procedure repeats until the queue is empty.

In TRACON, assume we have a queue of incoming tasks t_i, where $i \in \{1, 2, \ldots, l\}$ and l is the total number of available tasks, and virtual machines are denoted as $VM_{j,k}$, where $j \in \{1, 2, \ldots, m\}$ and m is the number of VMs per physical machine; $k \in \{1, 2, \ldots, n\}$ and n is the number of physical machines. First, MIBS takes the first task *candidate*$_1$ in the queue as the input to run MIOS. Second, MIBS chooses another task *candidate*$_2$ from the rest of the queued tasks that has the least interference with *candidate*$_1$. Then, MIBS takes *candidate*$_2$ as the input to run MIOS. On one hand, MIBS needs to calculate the interference between the incoming tasks, which may lead to a longer waiting time in the queue. However, as MIBS considers the pairing of all incoming tasks, it has good chances of improved performance when the models accurately predict the interference between different tasks. The algorithm of MIBS scheduling is listed in Algorithm 2.

Algorithm 2: MIBS

```
Data: $Queue$ is a task batch of $t_i$, where $i \in 1, \ldots, l$ ;
      $VM_{j,k}$ are VMs on available machines, where
      $j \in 1, \ldots, m$, and $k \in 1, \ldots, n$ ;
      $Model$ is the interference prediction model.
Result: $t_i$ and $VM_{j,k}$ assignments.
begin
    while $Queue$ is not empty do
        $candidate_1 = t_1$;
        MIOS($candidate_1$, $VM_{j,k}$, $Model$);
        for each task $t_i$ in the Queue, $i \neq 1$ do
            $score_i$ = Predict($t_i$, $t_1$, $Model$);
        end
        // the first "Min"
        $candidate_2$ = Min($score_i$);
        // the second "Min"
        MIOS($candidate_2$, $VM_{j,k}$, $Model$);
        RemoveFromQueue($candidate_1$, $candidate_2$);
    end
end
```

Minimum Interference miXed scheduler (MIX) intends to combine two algorithms and possibly improve the performance. The scheduler will not dispatch an assignment of MIBS immediately. Instead, MIX gives every job a chance to be the first job in the queue when executing MIBS, and hopes that future assignments would possibly offer new opportunities for better scheduling decisions. The obvious drawback here is that for each task, the delay may be increased, although the overall performance could potentially be improved. The algorithm of MIX scheduling is listed in Algorithm 3.

Algorithm 3: MIX

```
Data: $Queue$ is a task batch of $t_i$, where $i \in 1, \ldots, l$ ;
      $VM_{j,k}$ are VMs on available machines, where
      $j \in 1, \ldots, m$, and $k \in 1, \ldots, n$ ;
      $Model$ is the interference prediction model.
Result: $t_i$ and $VM_{j,k}$ assignments.
begin
    while $Queue$ is not empty do
        for each task $t_i$ in the Queue do
            Mark $t_i$ as the first task in $Queue$;
            $Assignment_i$ = MIBS($Queue$, $VM_{j,k}$,
            $Model$);
            if $Assignment_i$ is better than
            $Assignment_{MIX}$ then
                keep $Assignment_i$ as $Assignment_{MIX}$;
            end
        end
        Execute $Assignment_{MIX}$;
        RemoveFromQueue($Assignment_{MIX}$);
    end
end
```

In summary, three scheduling strategies have different advantages and drawbacks. MIOS has the lowest scheduling overhead, MIX has the potential to achieve the best performance while incurring the highest possible overheads, and MIBS stands in between which we will show shortly can lead to a good balance between the scheduling performance and overhead.

4. EVALUATION

4.1 Data-intensive Benchmarks

As we mostly focus on the I/O performance interference in a virtualized environment, we select eight data-intensive benchmarks in this work, covering different applications from bioinformatics, software development, system administration, data mining, multimedia processing, and server applications. Table 3 summarizes the benchmarks used in this work. For the I/O intensity, a larger number indicates a higher IOPS and throughput requirement.

Bioinformatics: It is a crucial task to find similar DNA or protein sequences for the bioinformatics research. Basic Local Alignment Search Tool (BLAST) [3] is one of the most widely used algorithms for identifying local similarity between different biological sequences. This is done by comparing sequences to databases and identifying sequence regions with statistically significant scores. BLAST can be used for multiple purposes, and we choose two NIH BLAST algorithms *blastn* and *blastp*, which are used to answer the nucleotide and protein queries, respectively [30]. For the inputs, the nucleotide and protein databases used in this work are NCBI's (National Center for Biotechnology Information) NT (12GB) and NR (11GB) databases that contain the full-set of non-redundant DNA and protein sequences.

Software development: Source code compilation is a commonly used benchmark for storage systems. During the compilation process, the compiler reads a number of the source code files at different time points and writes the object files to disks. Here we *compile* the Linux kernel of 2.6.18.

Table 3: Data-Intensive Applications and Benchmarks

Name	Category	Description	Data size	File count	I/O Intensity
blastn	Bioinformatics	DNA sequence similarity searching	12 GB	101	6
blastp	Bioinformatics	Protein sequence similarity searching	11 GB	61	3
compile	Software development	Linux kernel compilation	2.1 GB	1,358	4
dedup	System administration	Data compression and deduplication	672 MB	1	7
email	Server application	Email server workload benchmark	1.6 GB	249,825	1
freqmine	Data mining	Frequent itemset mining	206 MB	1	5
video	Multimedia processing	H.264 video encoding	1.5 GB	1	8
web	Server application	Web server workload benchmark	160 MB	10,000	2

System administration: As digital data grows exponentially, deduplication becomes an important task for system administrators to remove data redundancy and reduce the cost of storage systems. We use *dedup* from the Parsec benchmark suite [5]. Dedup applies various data compressions to a data stream in a pipelined manner and writes an output file with the compressed data. In the test, dedup uses an input file of 672 MB. Note that we also choose two other data-intensive benchmarks from Parsec.

Data mining: For data-mining applications, we pick *freqmine* from Parsec, which mines frequent itemsets from a 206MB input file.

Media processing: Again, we choose a Parsec benchmark called *video*, which is used to encode an H.264 video file of 1.5GB. Note that video has the highest IOPS among all the benchmarks.

Server application: We choose to benchmark two typical enterprise servers, *email* and *web* servers. For email server workload, we use a popular benchmark, postmark [19], which performs a large number of file operations (create, read, write, or delete) on small files.

For web server workload, we use the web server profile in the FileBench [24]. For web benchmark, we evaluate the IOPS only and do not evaluate runtime because FileBench takes the runtime as an input. This benchmark simulates a mix of open/read/close operations of 10,000 files in about 20 directories, and performs append to a file for every ten reads/writes to simulate the proxy log. In total 100 threads are used and the average file size is 16KB.

Mixed I/O workload: We utilize eight benchmarks to generate the workloads with different I/O intensities. In particular, we create three types of workloads, namely the light, medium, heavy I/O, which represent a mixture of workloads with low, medium, and high I/O requirements, respectively. To this end, we sort eight benchmarks based on their IOPS, which are shown in Table 3. Each number represents the rank of an application in terms of the I/O intensity. For example, number 1 represents *email* with the lowest IOPS and number 8 means *video* with the highest IOPS. We generate light, median, and heavy I/O workloads by following the Gaussian distributions with the means of 2.5, 4, and 5.5, respectively.

4.2 Simulation Settings

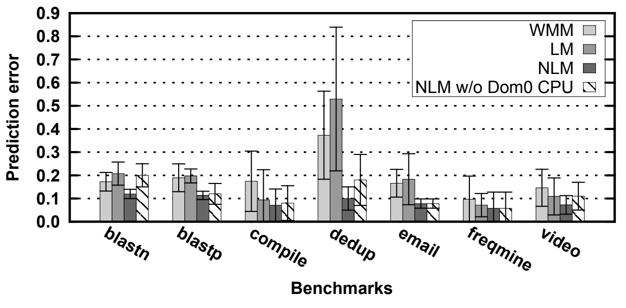
We implement a simulator to emulate the TRACON's performance in large-scale data centers. The simulator can evaluate two different scenarios with static and dynamic workloads. In the first case, we assume that there is a list of applications that are waiting to be processed. The number of applications equals to the total number of available virtual machines. Upon the arrival of such workload, the simulator queries the interference prediction module for expected workload interference and generates a schedule based on the predicted results. The simulator calculates the performance by using the actual statistics that have been measured in the real systems.

In dynamic workload scenario, we assume that the workload arrival rate follows a given distribution, and each task can be scheduled as soon as possible. When a scheduling event is triggered, the simulator takes all the tasks in the queue and current statuses on all VMs as the input, and queries the prediction module. Next, the scheduler generates an assignment based on the predicted results and the emulator estimates the actual time and system status by using previously measured data. Since the workloads are arriving randomly in time, they may be scheduled in between of executions of their co-located tasks. To address this, we calculate the new runtime by using the remaining portion of the work. For example, task A in VM1 and task B in VM2 are running on the same physical machine from the beginning. But, task B finishes earlier than task A and a scheduler puts task C as next task onto VM2. Suppose task A has already finished 80% of its workload, the remaining runtime of task A is estimated by assuming that 20% of its workload runs concurrently with task C.

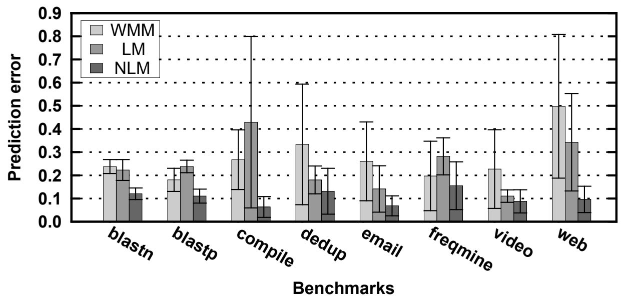
We run the simulations for a data center with 8 to 1,024 machines, and scale up to 10,000 machines. We measure the real effects of interference and use the measured data for simulation. All evaluations and measurements are conducted on Dell machines with 2.93 GHz Intel Core2 Duo E7500 processor, 4GB RAM, and a 1TB Samsung SATA hard drive, with Linux kernel 2.6.18 and Xen 3.1.2. Each VM is allocated with one virtual CPU, 512MB RAM, and 200GB disk space. For simplicity, we assume that the machines are homogeneous in the data center. For all the experiments, we report the average value of three runs. The emulation results are compared to the First-In-First-Out (FIFO) scheduler, which is served as a baseline in the following experiments.

4.3 Performance of Prediction Models

We profile and model the eight benchmarks with the methods described in Section 3.1. The *prediction error* is defined as $\mid$ *predicted value* $-$ *actual value* $\mid$ / *actual value*. Fig. 3(a) and 3(b) show the prediction errors of LM, NLM, and WMM on the runtime and IOPS with respect to differ-



(a) Runtime models



(b) IOPS models

Figure 3: Model prediction errors. The column heights represent the average prediction errors, and the error bars represent the standard deviations

ent benchmarks.

For both models, NLM's prediction errors stay relatively stable across different benchmarks. For the benchmarks with many random I/O operations, like *compile* or *web*, LM and WMM have higher prediction errors than those with mainly sequential I/O operations, like *video*. The differences between linear and non-linear models are mostly contributed by the bursty I/O patterns from the applications, which makes the linearity hard to hold in such cases. We find that adding degree-2 terms into a model significantly reduces the prediction error. In general, NLM has 10% prediction errors compared to 20% or more for LM and WMM. The improvement is bigger when applying the models on the IOPS. Note that NLM is also more stable with small standard deviations, shown as the error bars in the figure.

We also want to point out that, as shown in Fig. 3(a), the fourth parameter, global CPU utilization, is very important for the model to achieve high accuracy. Without it, NLM would have much larger prediction errors, e.g., twice as much for blastn.

4.4 Task Scheduling with Different Models

We evaluate the effectiveness of different prediction models when used in the scheduler. In each run, we generate a batch of tasks by randomly uniformly sampling tasks from the eight applications. For a batch of tasks, three schedules are generated by MIBS with WMM, LM, and NLM respectively. The performance numbers are normalized to those from the FIFO scheduler. Let the total runtime of tasks scheduled by a scheduler S be RT_S and the one by FIFO be RT_{FIFO}. The runtime improvement, speedup, is defined as

$$Speedup = \frac{RT_{FIFO}}{RT_S}. \tag{5}$$

Similarly, let the total IOPS of tasks scheduled by a scheduler S be $IOPS_S$ and the one by FIFO be $IOPS_{FIFO}$. The throughput improvement, IOBoost, is defined as

$$IOBoost = \frac{IOPS_S}{IOPS_{FIFO}}. \tag{6}$$

Let the MIBS$_{RT}$ be the MIBS with the goal to minimize the total runtime and MIBS$_{IO}$ be the MIBS with the goal to maximize the total IOPS. Fig. 4 shows *Speedup* and *IOBoost* by MIBS$_{RT}$ and MIBS$_{IO}$ when using WMM, LM, and NLM respectively. NLM not only has lower prediction errors than WMM and LM, but also has better performance in assisting the scheduler to minimize the runtime and increase the I/O throughput. In this experiment, a batch of 32 tasks are scheduled on to a cluster with 16 machines and each machine has two VMs.

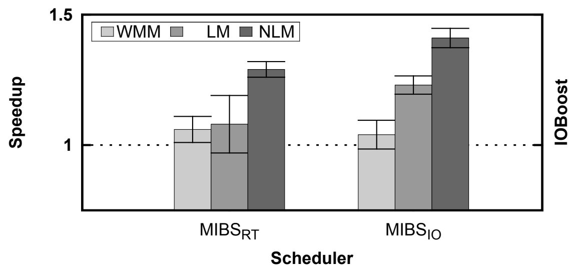


Figure 4: Task scheduling with different models. The column heights represent the average runtime and IOPS improvements, and the error bars represent the standard deviations

4.5 NLM Prediction Accuracy

In this section, we analyze NLM's ability in determining the minimum runtime and maximum throughput. Fig. 5 shows the predicted minimum, measured (real) minimum, average, and maximum runtimes of each application when it runs concurrently with other applications. One can see that NLM is able to closely predict a benchmark's minimum runtime, and the predicted minimum never goes beyond the measured average or maximum runtimes. Similarly, NLM performs well on the IOPS predictions as shown in Fig. 6. The predicted maximum IOPS is always within a small distance from the measured maximum throughput.

4.6 Model Adaption

Our models can make dynamic adjustments to improve the prediction accuracy during the runtime. In this experiment, we build an initial interference model of blastn with application statistics (a total of 500 data points) collected on a machine with local storage devices. Then we use this model to predict the blastn's runtime and IOPS on the machine with identical software and hardware setting, but instead using remote storage devices via the iSCSI (Internet SCSI) interface.

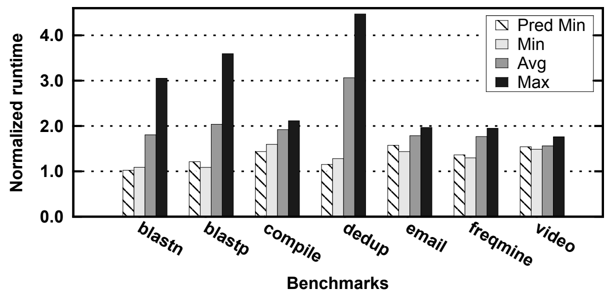


Figure 5: Predicted minimum runtime of each application compared to measured minimum, average, and maximum runtimes

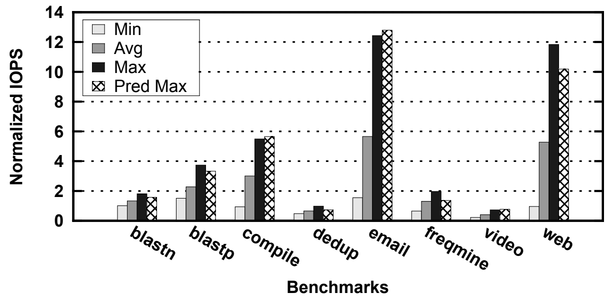


Figure 6: Predicted maximum IOPS of each application compared to measured minimum, average, and maximum IOPS

In Fig. 7, we can see that different storage devices can result in dramatic drops in prediction accuracy for our blastn models - the prediction error of IOPS model increases from 12% to 83%, and the prediction error of runtime increases from 12% to 160%. In this case, TRACON continues to collect the application statistics from the runtime environment, and gradually replaces the old training data with the newly collected data. We rebuild the models when every 160 new data points are collected. As shown in Fig. 7, TRACON is able to reduce the prediction accuracy quickly to the same level of around 10% as it was before. If the environment remains unchanged, that is, the local storage is used throughout the experiment, the model can be slightly improved although the difference between the old and new models is too small to be distinguishable in Fig. 7.

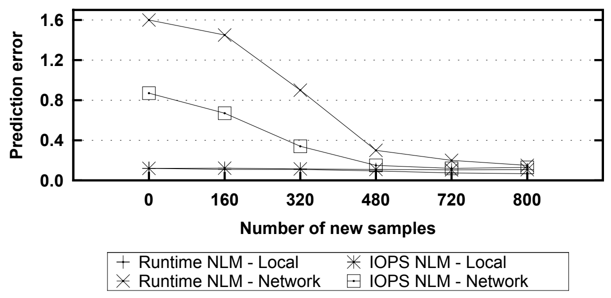


Figure 7: Online model learning

In summary, NLM has a lower prediction error and better performance than both LM and WMM. In addition, it is able to dynamically adjust its prediction accuracy when adapting to a different environment. Therefore, we will use NLM as the prediction module in the following experiments.

4.7 Performance of Scheduling Algorithms

Static workload: In a static workload scenario, we use workloads with different I/O intensities to examine the improvements when we schedule it using MIBS$_{RT}$ and MIBS$_{IO}$. Fig. 8 demonstrates the speedups from MIBS$_{RT}$ and MIBS$_{IO}$ with respect to different numbers of machines and I/O workloads. For the heavy I/O workload, both MIBS$_{RT}$ and MIBS$_{IO}$ get limited speedups because there is no much room to reduce the interference - almost all combinations in this workload likely severely interfere with each other. MIBS$_{RT}$ outperforms MIBS$_{IO}$ in this case because the I/O bandwidth has been saturated. When the workload has the light I/O intensity, both MIBS$_{RT}$ and MIBS$_{IO}$ achieve significantly better performance, with 30% speedups, than with the heavy I/O workload. Intuitively, in this case, even FIFO could have a good chance to encounter less interference.

The best performance is achieved for the medium I/O workload, where both MIBS$_{RT}$ and MIBS$_{IO}$ obtain more than 40% improvement over FIFO. In addition, MIBS$_{IO}$ beats FIFO by 1.5 times when there are 1,024 machines. Note that MIBS$_{IO}$ outperforms MIBS$_{RT}$ in this case because it can effectively increase the I/O utilization without utilizing all the bandwidth.

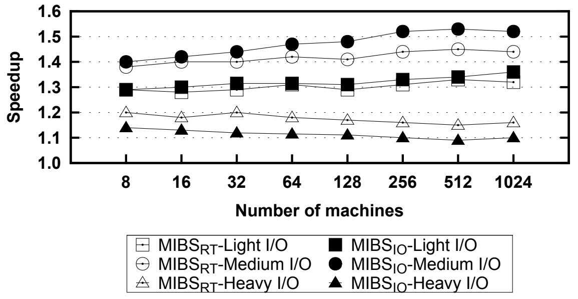


Figure 8: Speedup by MIBS$_{RT}$ and MIBS$_{IO}$

Dynamic workload: Most data centers are dealing with the tasks that arrive dynamically and need to schedule them in a real-time fashion. In this section, we assume that task arrival rate follows a Poisson process with an average rate of λ tasks per minute. The throughput T_S is defined as the number of tasks completed on a system with the scheduler S in a time period. The normalized throughput is defined as T_S / T_{FIFO}.

Suppose that the schedulers are MIBS$_8$, MIOS, and MIX$_8$ where the subscripts here represent a queue length of eight tasks and there are ten hours to process tasks in a data center with 64 machines. We present in Fig. 9 the normalized throughputs of MIBS$_8$, MIOS and MIX$_8$ when the workloads have light, medium, or heavy I/O intensities, respectively. When λ is small, three schedulers have similar throughputs because the data center is idle for most of the time. That is, a scheduler can always find an idle machine for an incoming task. As λ goes up, the machines are gradually occupied, and the advantages of a scheduling algorithm

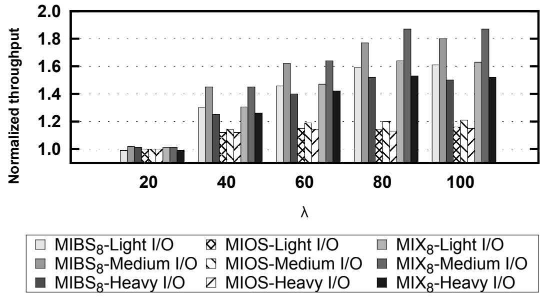


Figure 9: Normalized throughputs of MIBS$_8$, MIOS and MIX$_8$ at λ tasks per minute

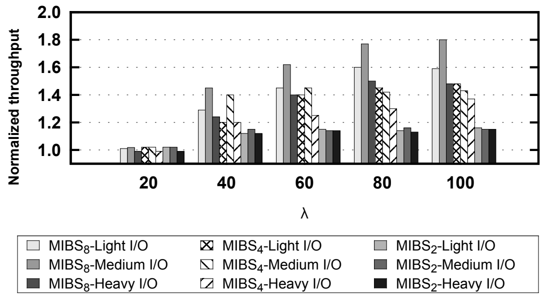


Figure 10: Normalized throughputs of MIBS for different λ and queue lengths

become more obvious. In this case, although MIX$_8$ has the best performance, MIBS$_8$'s performance is very close with a smaller overhead, which makes it more suitable for dynamic workloads. Similar to the previous results for static workload, three schedulers can achieve higher throughputs for the medium I/O workload than the light and heavy I/O workloads.

Fig. 10 shows these normalized throughputs of MIBS are improved as λ increases. We vary the queue length of MIBS from two, four, to eight. The trend remains that for different queue lengths, MIBS works best for medium I/O workload. The performance improves when the queue length increases, e.g., at λ of 100, MIBS$_8$ achieves about 10% higher throughput than MIBS$_4$ and MIBS$_2$.

4.8 Scalability

We explore the performance of different schedulers when using 8 to 1,024 machines with $\lambda = 1,000$. Fig. 11 shows MIBS$_8$'s throughput is close to MIX$_8$'s and the gap is reduced as the number of machines increases. In this case, MIBS$_8$ is a better solution because it has a smaller scheduling overhead while achieving an approximate throughput as MIX$_8$. In contrast, MIOS has the least performance improvement over FIFO. If we scale the data center to 10,000 machines and $\lambda = 10,000$, the normalized throughput of MIBS$_8$ with the medium I/O workload remains high with 40% improvement.

Fig. 12 demonstrates the changes in normalized throughputs of MIBS$_8$, MIBS$_4$, and MIBS$_2$ when the number of machines increases. Similar to varied λ in Fig. 10, MIBS with a longer queue has a higher I/O throughput than ones with a shorter queue.

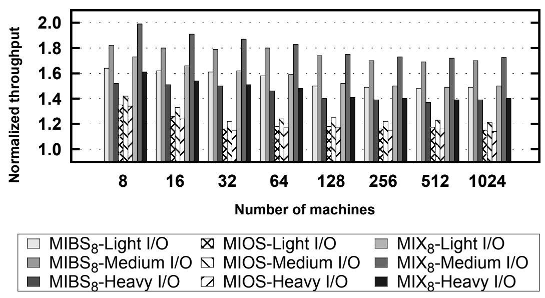


Figure 11: Normalized throughputs of MIBS$_8$, MIOS and MIX$_8$ for different number of machines

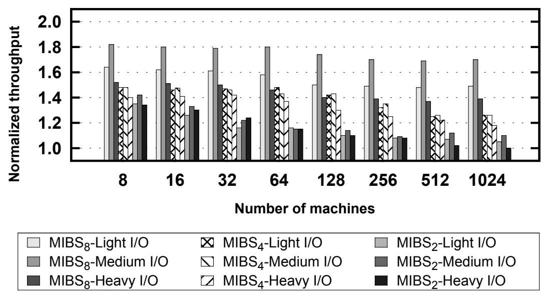


Figure 12: Normalized throughputs of MIBS for different number of machines and queue lengths

5. CONCLUSION

Virtualization has become the key for data centers to achieve excellent resource utilization, scalability, and high availability. In this work, we investigate the performance effects of co-located data-intensive applications in virtualized environments, and propose a management system TRACON that mitigates the interference effects from concurrent data-intensive applications and greatly improves the overall system performance. First, we study the use of statistical modeling techniques to build three different models of performance interference, and propose to use the non-linear models as the prediction module in TRACON. Second, we develop three scheduling algorithms that work with the prediction module to manage the task assignments in virtualized data centers.

In the future, we plan to extend this work with different modeling techniques to build a more accurate model and reduce the modeling and profiling overheads. We also plan to investigate new scheduling algorithms to further reduce scheduling overheads while maintaining a good system performance. Furthermore, we will explore I/O interference effects on various storage devices, e.g., RAID and solid-state drives (SSD), as well as network storage systems.

6. ACKNOWLEDGMENTS

This material is based on research sponsored by the National Science Foundation under grant OCI-0937875. The authors are grateful to the anonymous reviewers for their helpful comments and feedback.

7. REFERENCES

[1] Hirotugu Akaike. A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38:63–74, August 2008.

[3] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[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 *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177. ACM, 2003.

[5] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[6] Peter Bodík, Rean Griffith, Charles Sutton, O Fox, Michael Jordan, and David Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 12–12. USENIX Association, 2009.

[7] Carlos Boneti, Roberto Gioiosa, Francisco J. Cazorla, and Mateo Valero. A dynamic scheduler for balancing hpc applications. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 41:1–41:12. IEEE Press, 2008.

[8] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölóni, Albert I. Reuther, Mitchell D. Theys, Bin Yao, Richard F. Freund, Muthucumaru Maheswaran, James P. Robertson, and Debra Hensgen. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop (HCW)*, pages 15 –29, 1999.

[9] Kenneth P. Burnham and David R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2nd edition, July 2002.

[10] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. In *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pages 1 –14, May 2011.

[11] Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization)*. Wiley-Interscience, 3 edition, February 2008.

[12] Edward G. Coffman. *Computer and Job Shop Scheduling Theory*. John Wiley & Sons Inc, New York, 1976.

[13] Ewa Deelman and Ann Chervenak. Data management challenges of data-intensive scientific workflows. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 687 –692, May 2008.

[14] Norman Richard Draper and Harry Smith. *Applied Regression Analysis*. John Wiley and Sons, New York, 1981.

[15] H. Howie Huang and Andrew S. Grimshaw. Automated performance control in a virtual distributed storage system. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, GRID '08, pages 242–249. IEEE Computer Society, 2008.

[16] H. Howie Huang, Shan Li, Alex Szalay, and Andreas Terzis. Performance Modeling and Analysis of Flash-based Storage Devices. In *Proceedings of the 2011 IEEE Symposium on Massive Storage Systems and Technologies (MSST'11)*, 2011.

[17] Oscar H. Ibarra and Chul E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24:280–289, April 1977.

[18] Richard Arnold Johnson and Dean W. Wichern, editors. *Applied multivariate statistical analysis*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[19] Jeffrey Katcher. Postmark: a new file system benchmark. Network Appliance Tech Report TR3022, October 1997.

[20] Vahid Kazempour, Ali Kamali, and Alexandra Fedorova. Aash: an asymmetry-aware scheduler for hypervisors. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '10, pages 85–96. ACM, 2010.

[21] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2007.

[22] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –10, 2010.

[23] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[24] Richard McDougall and Jim Mauro. *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture*. Prentice Hall, 2006.

[25] Yiduo Mei, Ling Liu, Xing Pu, and S. Sivathanu. Performance measurements and analysis of network i/o applications in virtualized cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 59 –66, 2010.

[26] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Alice X. Zheng, and Gregory R. Ganger. Modeling the relative fitness of storage. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, pages 37–48. ACM, 2007.

[27] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5. USENIX Association, 2005.

[28] Farrukh Nadeem and Thomas Fahringer. Predicting the execution time of grid workflow applications through local learning. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 33:1–33:12. ACM, 2009.

[29] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 237–250. ACM, 2010.

[30] NIH. BLAST: Basic Local Alignment Search Tool. `http://www.ncbi.nlm.nih.gov/BLAST/`.

[31] Xing Pu, Ling Liu, Yiduo Mei, S. Sivathanu, Younggyun Koh, and C. Pu. Understanding performance interference of i/o workload in virtualized cloud environments. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 51 –58, 2010.

[32] Hongzhang Shan, Katie Antypas, and John Shalf. Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 42:1–42:12. IEEE Press, 2008.

[33] Fengguang Song, Asim YarKhan, and Jack Dongarra. Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 19:1–19:11. ACM, 2009.

[34] Ryutaro Susukita, Hisashige Ando, Mutsumi Aoyagi, Hiroaki Honda, Yuichi Inadomi, Koji Inoue, Shigeru Ishizuki, Yasunori Kimura, Hidemi Komatsu, Motoyoshi Kurokawa, Kazuaki J. Murakami, Hidetomo Shibamura, Shuji Yamamura, and Yunqing Yu. Performance prediction of large-scale parallell system and application using macro-level simulation. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 20:1–20:9. IEEE Press, 2008.

[35] US Environmental Protection Agency (EPA). Report to congress on server and data center energy efficiency: Public law 109-431. 2008.

[36] Xiaodan Wang, Eric Perlman, Randal Burns, Tanu Malik, Tamas Budavári, Charles Meneveau, and Alexander Szalay. Jaws: Job-aware workload scheduling for the exploration of turbulence simulations. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11. IEEE Computer Society, 2010.

[37] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 366–387. Springer-Verlag New York, Inc., 2008.

[38] Yuanyuan Zhang, Wei Sun, and Yasushi Inoguchi. Predicting running time of grid tasks based on cpu load predictions. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 286–292. IEEE Computer Society, 2006.

[39] Qian Zhu, Jiedan Zhu, and Gagan Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12. IEEE Computer Society, 2010.