Matrix: Achieving Predictable Virtual Machine Performance in the Clouds

Ron C. Chiang*, Jinho Hwang+, H. Howie Huang*, and Timothy Wood*
The George Washington University and IBM T.J. Watson Research Center*+*

Abstract

The success of cloud computing builds largely upon on-demand supply of virtual machines (VMs) that provide the abstraction of a physical machine on shared resources. Unfortunately, despite recent advances in virtualization technology, there still exists an unpredictable performance gap between the real and desired performance. The main contributing factors include contention to the shared physical resources among co-located VMs, limited control of VM allocation, as well as lack of knowledge on the performance of a specific VM out of tens of VM types offered by public cloud providers. In this work, we propose Matrix, a novel performance and resource management system that ensures the desired performance of an application achieved on a VM. To this end, Matrix utilizes machine learning methods - clustering models with probability estimates - to predict the performance of new workloads in a virtualized environment, choose a suitable VM type, and dynamically adjust the resource configuration of a virtual machine on the fly. The evaluations on a private cloud, and two public clouds (Rackspace and Amazon EC2) show that for an extensive set of cloud applications, Matrix is able to estimate application performance with average 90% accuracy. In addition, Matrix can deliver the target performance within 3% variance, and do so with the best cost-efficiency in most cases.

1 Introduction

In private and public clouds, the so-called Infrastructure as a Service (IaaS) model offers on-demand creation of virtual machines (VMs) for different users and applications, and enables dynamic management of VMs for maximizing resource utilization in the data centers. Ideally, a VM shall have three properties: 1) *efficiency*, where a significant portion of the program runs without any intervention from the hypervisor that manages the VMs; 2) *resource control* that prevents any program from gaining the full control of the system resources; and 3) *equivalence*, where any program running in a VM "performs in a manner indistinguishable" from an equivalent real machine [34]. Although virtualization technology has been improved greatly (its pervasive use in cloud computing is strong evidence), we have not yet achieved the vision of "an efficient, isolated duplicate of a real machine", that is, a VM shall be able to provide the performance close to the desired one.

Take a real world example, before buying a new tablet computer from an online retailer, one may shop a local store like BestBuy to test drive and compare various products. Nevertheless, any product that the customer eventually receives from the online retailer will be the same as what is presented locally. Unfortunately, when one purchases a VM in the cloud, little guarantee is provided to ensure an application hosted by the VM would keep the desired performance, not even mentioning to achieve the best cost-efficiency.

In this paper, we propose the concept of *Relative Performance* (RP) as the "equivalence" metric that measures the ratio between the desired performance and that of running in a VM. For a workload w, the RP can be formally defined as

$$RP_w = \frac{P_{VM}}{P_d}, \tag{1}$$

where P_{VM} is the performance of the workload w when running on a VM, and P_d is the desired performance. The performance is workload dependent and can be measured as the runtime (e.g., sequence alignment), throughput (e.g., video streaming), latency (e.g., webpage serving), etc. The RP that is equal to one means that the workload delivers the desired performance on the VM. The goal of Matrix is to deliver the desired performance while minimizing the resource cost.

In a cloud, many factors such as limited control of VM allocation and competition from co-located VMs to shared resources (e.g., CPU and I/O devices) contribute to hard-to-predict VM performance. To illustrate the problems on expected performance and operating cost, we run three benchmarks ranging from I/O intensive, memory intensive to CPU intensive workloads, both locally and on Amazon EC2. There are two local physical machines in this test: $PM1$ has a 2.93 GHz Intel Core2 Duo processor and 4 GB memory, and $PM2$ has a 3 GHz Intel Pentium4 processor with 2 GB memory. The desired performance P_{d1} and P_{d2} are the performance of running a given benchmark on $PM1$ and $PM2$ respectively. Fig. 1 shows the RPs (in runtime/latency for three benchmarks) for P_{d1} and P_{d2} on four EC2 instances[1]. Our tests show that the RP for these three benchmarks can vary dramatically from 18% of the target performance to more than three times. Clearly, it is challenging to know ahead of time for each application which VM instance provides a good tradeoff between the cost and

[1]For Amazon EC2 instances, $m1.small$ type equips with 1.7 GB memory and 1 EC2 Compute Unit priced at six cents per hour, $m1.medium$ 3.75 GB memory and 2 Compute Units at 12 cents per hour, $m1.large$ 7.5 GB memory and 4 Compute Units at 24 cents per hour, and the $t1.micro$ has the smallest amount of memory (613 MB) and CPU resource.

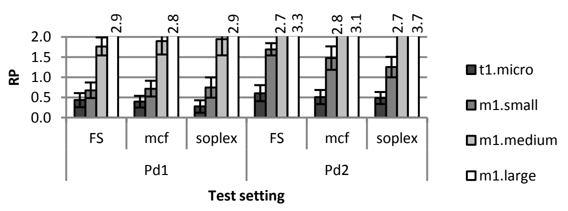


Figure 1: The performance for various EC2 instances ranges from 27% to 3.7 times of the desired performance P_{d1} and P_{d2}. Each column shows an average of ten runs

performance. Benchmarking an application in the cloud may alleviate the problem, but it becomes cumbersome as public cloud providers offer dozens of VM types.

In this work, we propose a performance and resource management system, *Matrix*, that targets at delivering predictable VM performance with the best cost-efficiency. To achieve this goal, Matrix utilizes clustering models with probability estimates to predict the performance of new workloads in a virtualized environment, chooses a suitable VM type, and dynamically adjusts the resource configuration of a VM on the fly.

The first contribution is that *Matrix can predict accurately how a new workload will perform on different cloud VM instances*. To this end, Matrix first constructs performance models of a set of representative workloads that define common application "genes". Given performance models for these applications, we leverage the support vector clustering (SVC) to quickly classify a new workload, using soft boundary probability estimates to infer its "gene" composition. A number of studies [28, 46, 22, 45] have worked on service-level agreement (SLA), performance prediction, and anomaly detection in virtualized environments. The major differences of Matrix lie in the understanding of the dynamic relationship between resource allocation and the new workload performance.

The second contribution is that *Matrix allocates VM resource to application in a way that minimizes the cost while achieving good performance*. To this end, Matrix applies an approximate optimization algorithm and makes use of the characteristics of the kernel functions of support vector machine (SVM) to find the optimized resource allocation. More specifically, the support vector regression (SVR) is used to develop our RP models. By exploiting gene composition knowledge, Matrix can do so without knowing a priori application information within guest VMs.

Third, *Matrix is able to handle different cloud environments and applications.* We conduct a large set of experiments with real cloud applications and ranging from a single machine, a local cluster, and a virtual cluster, to evaluate Matrix on both our private cloud and the public cloud of Amazon EC2 and Rackspace.

In this work, we present three use cases of Matrix:

- **Automatic VM configuration**. Matrix can adapt VM settings to the changes in workload, while maintaining a desired performance and achieving good cost-efficiency in the cloud.
- **VM instance recommendation**. With workload performance models, Matrix recommends the VM instance that is best suited for specific applications.
- **Cloud provider recommendation**. Given a new application, Matrix can also help users to choose an appropriate VM from different cloud providers.

2 Related Work

Performance Modeling and Analysis has been extensively studied, both in non-virtualized environments [29, 44], and virtualized environments [16, 36, 21, 54, 7]. There are also performance models which target specific applications or system components. For example, Li et. al [25] model the performance of parallel matrix multiplication in virtualized environments, and Watson et al. build probability distribution models of response time and CPU allocations in virtualized environments [50]. While we share the same idea on exploiting machine learning techniques, we further explore the ability of classification with probability estimates to model the performance of new workloads.

Automatic Resource Configuration is an important issue in parallel and distributed systems [24, 38, 15] and performance monitoring tools [23]. Similarly, various machine learning techniques have shown promising results for VM provision and configuration, e.g., clustering [35], classification [26], reinforcement learning [37]. Also, several works have focused on minimizing operation cost, for example, Niehörster et al. [31] applies fuzzy control at runtime, and Kingfisher [41] formulates the problem as an integer linear program (ILP) and implements a heuristic ILP solver. Most related to our work are several existing resource configuration frameworks such as DejaVu [48], JustRunIt [55], and [43]. The key differences of Matrix lie in a comprehensive framework to predict and maintain the desired performance of a new workload while minimizing the operating cost. While DejaVu also handles new applications and adapts resources to suit new demands, DejaVu uses dedicated sandbox machines to clone and profile VMs. In contrast, Matrix utilizes representative models to construct new workload's model in an online fashion. Many works aim to predict resource requirements for cloud applications, e.g., [13, 14, 18, 49, 51]. Most of them either focus on single application or ignore the cost-efficiency. On the other hand, Matrix is able to adapt to new applications and minimize the operating cost. Also, Matrix deals with the problem of multi-cloud resource management, which is shown to be critical in [5]. Performance interference in virtualized environments is another critical barrier to provide predictable performance. DeepDive [32] utilizes mathematical models and clustering techniques to detect interference. DeepDive requires comparing the performance from VM clones in dedicated machines. Similar to [7, 19, 30], Matrix removes this need by including the interference factors into the performance models.

3 Matrix Architecture

The goal of Matrix is to predict and configure VMs in an automatic manner so that the applications running within the VMs would achieve the performance with a close vicinity of a specific one. We present the architecture of Matrix in Fig. 2. On the left, Matrix builds both clustering and RP models of representative workloads and this task is done offline. There are three steps in this phase: 1) profiling the training set of representative workloads (presented in Sec. 3.1); 2) tuning the SVM parameters to find the best model configuration; and 3) training the classifier and the basic RP models, for later use of the online module (Sec. 3.2 and 3.3). This offline training stage builds RP models from our generic benchmarks, but it can be repeated periodically to include data from newly added workloads.

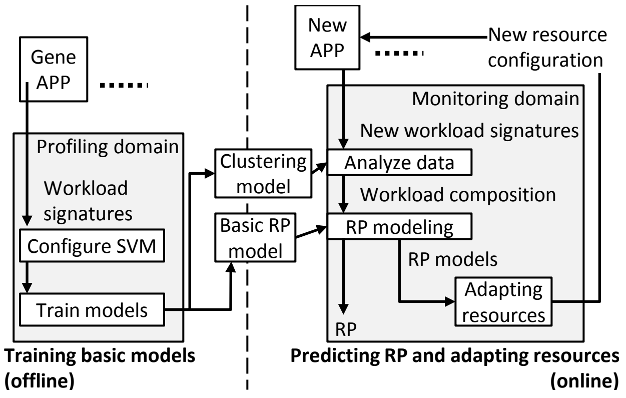


Figure 2: Matrix Architecture

When a new application is moved to the cloud, Matrix requires only the workload signature when running on its current infrastructure, which could be either physical or virtual machines. As shown in the right hand side of Fig. 2, Matrix can classify these workload signatures compared to the previously trained models. Then, the system calculates a runtime RP model based on adjusted performance estimates and outputs the predicted RP to the resource allocation module. Next, Matrix will search for the VM configurations with the minimum cost to maintain a desired performance (Sec. 3.4). To provide automatic resource management, we formulate an optimization problem with nonlinear inequality constraints. For fast response time, Matrix utilizes the Lagrange multipliers to provide an approximate solution and a bound to the minimum resource cost.

3.1 Workload Signatures

Matrix first must profile a set of workload "genes" that indicate how different types of applications will perform when moved into cloud platforms.

A group of representative applications are firstly selected as "genes" to construct an expert system. Our selection principle, similar to [2], is to have the reference workloads as diverse as possible - the resulting collection shall cover from CPU-intensive to data-intensive, and their problem sizes also shall vary from small to large data volumes. Table 1 summarizes the representative applications selected from a few widely used benchmark suites, e.g., FileBench [27], SysBench [20], SPEC2006 [10], PARSEC[1], and Cloud9 [4, 8]. Note that while this set of applications is not optimal by all means, they provide, as we will see in evaluations, a good basis for RP modeling. We leave the exploration of different gene applications as future work.

Table 1: Summary of representative applications

Name	Description
video server	serving a set of video files
web server	retrieving web contents and updating log files
file server	a mixture of various file I/O operations
OLTP	query and update database tables
mcf	running simplex algorithm
hmmer	pattern searching of gene database
soplex	linear program solver
canneal	evolutionary algorithm
DS01 to DS15	15 distributed data serving workloads
C01 to C15	15 parallel CPU-intensive workloads

For parallel application, we select a training set that consists of 15 data-intensive workloads (DS01 to DS15) and 15 CPU-intensive workloads (C01 to C15). The first five DS series workloads run Apache Cassandra, a distributed key-value store, with read/write ratios of 100/0, 75/25, 50/50, 25/75, and 0/100 where the record popularity is in uniform distribution. For DS6 to DS10, they access Cassandra with same read/write ratios but in the Zipfian distribution of record popularity. For the number 11 to 15 training workloads, they share the same pattern and order of read/write ratios in both the first and the second five groups, but the record popularity is in the latest distribution. The last 15 representative applications in the training set are CPU-intensive parallel workloads from Cloud9, a scalable parallel software testing service. The training set for CPU-intensive parallel workloads are randomly selected out of 98 different utility traces from the GNU CoreUtils 6.10 for running Cloud9.

For a basic signature, we take the arithmetic means of three system parameters - CPU utilization, the amount of data read and written per second. Since it is insufficient to use the mean alone to represent a workload when there is a large variability in the observed data, we choose the coefficient of variation (C.O.V) as part of the signatures to describe the variability. As prior work [17, 2] has already shown that the resource allocation of VMs greatly affects the observed system parameters, we include the number of VCPUs and the size of memory in the workload signatures because these two parameters are frequently used knobs for tuning VM performance. Furthermore, we also take into account the interference from co-located VMs. For simplicity, all workload signatures from other VMs are summed up as one background VM and included in the modeling process.

Dealing with applications running on multiple machines poses more challenges. The traffic in and out of each node is critical to data-intensive applications' performance. The number of nodes is also important for modeling workload concurrency. In other words, Matrix needs to scale resources horizontally (increasing and decreasing the number of nodes), as well as vertically (scaling up and down resources on each node). Thus, Matrix includes the amount of data flow of each node and the number of nodes in a cluster as additional parameters when modeling an application performance on a set of machines.

3.2 Clustering Method

Matrix needs a workload classifier to identify new workloads that are running in the guest VMs. Most of previous works use a "hard" classifier. That is, the classifier outputs a certain workload without ambiguity. That method, however, provides little help when dealing with new workload, which can be very different from any workload in the training set. To address this problem, we explore "soft" classifiers in this work, which have soft boundary and output probability estimates of being each component in the model. These probability estimates can be utilized as weights to infer the "gene" composition of new workloads. Specifically, we utilize a multiclass SVC with likelihoods provided by a pairwise coupling method [6]. We use a rigorous procedure to tune and train classifiers. Our classifiers are built as follows:

Data Scaling avoids the features in larger numeric ranges dominating those in smaller ranges. In addition, scaling data into a restricted range can avoid numerical difficulties during the kernel value calculation [6]. We scale each attribute in the range of $[0, 1]$.

Parameter Selection: Choosing the optimal parameter values is a critical step in the SVC design. The grid search method is a common practice in finding the best configuration of a SVC. That is, the parameter selection is usually done by varying parameters and comparing either estimates of generalization error or some other related performance measure [11]. When the search approaches a grid point, it calculates the value of ten-fold cross validation (CV). In order to save the searching time, the search firstly starts with a loose grid to identify regions with good CV values. Then, the search uses a finer grid to further approach the best configuration. We conduct the grid search on the following parameters: 1) SVC types: C-SVC [3] and ν-SVC [40, 39]. 2) Kernel functions: Polynomial, sigmoid, and Gaussian radial basis function (RBF). 3) Constraint violation cost C to avoid overfitting. $C \in \mathbb{R}^+$. 4) Kernel width coefficient γ, which affects the model smoothness. $\gamma \in \mathbb{R}^+$. 5) Variable ν in ν-SVC provides an upper bound on training errors $\nu \in (0, 1]$.

Training: Once the best parameter configuration is decided, the final classifier is trained by using the best configuration with the whole training data.

In terms of SVC types and kernel functions, the grid searching results suggest that ν-SVC with RBF kernel outperforms other classifiers and kernel functions. Therefore, Matrix uses ν-SVC with RBF kernel as the classifier. ν-SVC has been proved to provide an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.

3.3 Performance Modeling

The performance modeling has two main procedures: **1) Constructing the building block:** Matrix utilizes the SVR to construct the basic RP models of each training application. A popular version of SVR is ν-SVR [40, 39]. Matrix uses ν-SVR with the RBF kernel for the basic RP modeling because the grid searching results suggest it is better than others. **2) Generating the performance model:** The performance modeling of representative workloads completes one part of the story. Our goal is to capture new workloads' RP models in an online fashion.

Suppose there are n representative workloads w_i, $i \in \{1, \ldots, n\}$. The corresponding performance models are $f_i(R)$, where $r_j = \{x \in \mathbb{R} \mid 0 \le x \le 1\}$ and $R = \{r_1, \ldots, r_m\}$ are resource configurations and system statistics, $j \in \{1, \ldots, m\}$. Because all performance models are built by SVR, the performance models can be represented as: $f_i(R) = \underline{\theta} \cdot \phi(r_i) + \theta_0$, where $\phi(r_i)$ are kernel functions, θ_0 is the offset vector to the origin, and $\underline{\theta}$ is the separating vector.

Our classifier then analyzes a new workload w_{new} and generates an output $\{p_1, \ldots, p_n\}$, where p_i are the probability estimates of being workload w_i, $i \in \{1, \ldots, n\}$. The final performance model of workload w_{new} is

$$f_{new}(R) = \sum_{i=1}^{n} p_i \cdot f_i(R), \quad \text{where} \sum_{i=1}^{n} p_i = 1. \tag{2}$$

In other words, the likelihood p_i acts as a weight to control the fraction of f_i in the final model f_{new}.

3.4 Automatic Resource Configuration

Once we obtain a performance model of a new workload w_{new}, configuration module starts to find the minimum allocation for keeping the desired performance.

Let C_j be the cost of resource j, $j \in \{1, \dots, m\}$. Resources are, e.g., the memory size and the number of VCPUs and VMs. r_j is the ratio of resource j on a physical server that is allocated to the VM. We formulate the resource configuration problem as an optimization one with a nonlinear equality constraint:

$$\begin{aligned} \underset{R}{\text{minimize}} \quad & F_c(R) = \sum_{j=1}^{m} C_j \times r_j \\ \text{subject to} \quad & f_{new}(R) = \sum_{i=1}^{n} p_i \cdot f_i(R) = 1, \\ & \sum_{i=1}^{n} p_i = 1, \\ & r_j = \{x \in \mathbb{R} | 0 \leq x \leq 1\}, \\ & i \in \{1, \dots, n\}, j \in \{1, \dots, m\} \end{aligned}$$

Because both the objective and constraint function are continuously differentiable[2], we utilize the Lagrange algorithm for solving this problem.

Note that the above problem is formulated under the assumption that r_j is the ratio of resource j on a physical server that is allocated to the VM. However, real systems usually can not partition resources at an arbitrary granularity. For example, the memory allocation for VMs is usually done in the unit of one megabyte. If a system has 2 GB memory, the finest possible R_j values will be $\{1/2000, 2/2000, \dots, 2000/2000\}$. As a result, the system would not be able to use the optimal resource configuration R^*. Instead, the system needs to take $(\lceil r_1^* \rceil, \dots, \lceil r_m^* \rceil)$ as the resource configuration, where the ceiling operation of r_i here is defined as taking the smallest value r_i' in the finest possible granularity, such that $r_i' \geq r_i$. Let the granularity of resource i be d_i, $i \in \{1, \dots, m\}$. In other words, the miss allocation on resource i is at most d_i. Therefore, the upper bound on the extra resource allocation cost is $\sum_{i=1}^{m} C_i \times d_i$.

4 Implementation

We have implemented and tested Matrix on both a local private cloud and two public clouds, namely Amazon EC2 and Rackspace cloud servers. Fig. 3 summarizes the work flow of the prototype.

The preparing data block includes parsing, formatting, and scaling collected traces. The clustering model and RP models are previously built by the training set offline. The Matrix online module is controlled by a Linux bash shell script combined with a SVM module written in C and an optimization problem solver in MATLAB. The tasks of this online module are to 1) collect traces, 2) analyze workload compositions, and 3) predict current RP and suggest a configuration to obtain desired performance with less cost.

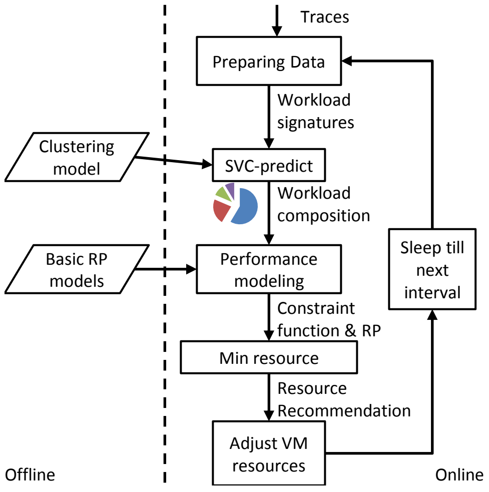


Figure 3: Matrix prototype

The online module is running as a background process in the host domain, which collects workload signatures of VMs every second by using *xentop*. At every minute, a parser will parse collected data and scale all values in the range of $[0, 1]$. The online module then feeds the scaled trace and clustering model to the *SVC-predict* module which outputs the workload composition in possibilities of representative workloads. These probability estimates along with the basic RP models become the running workload's performance model (Eq. 2). Then, Eq. 2 is served as the constraint function of the optimization problem in Sec. 3.4. Finally, the online module adjusts resource allocations and repeats the same procedure for the next interval.

There are three main differences between the two Matrix prototypes on private and public clouds. First, Matrix in public clouds can not use *xentop* to collect traces because we have no access to the host domain. Instead, we run *top* and *iostat* in every guest domain to collect traces. Second, Matrix can not arbitrarily adjust resources of an instance in the public cloud. And, instance types can only be changed when it is not running. To address this problem, we adapt Xen-blanket [53] nested virtualization for some tests.

Prototype performance. The measured running time from parsing collected trace to output the minimum resource recommendation is around 0.6 second where the optimization solver takes about 70% in the whole process. As future work, the running time of the online module can be further reduced by implementing the solver in

[2]One of the properties of the RBF kernel.

native system without using MATLAB. In addition, Matrix may also be integrated with Monalytics [23] to reduce overheads.

5 Evaluations

Testing scenarios. We evaluate Matrix in three scenarios: a single machine, a cluster of physical machines, and of VMs. Three virtualized environments are used in our experiments: local Xen virtualized servers, Amazon EC2 instances[3] and Rackspace cloud servers[4]. We label Rackspace cloud servers from smallest to the largest as RS1 to RS7. For example, RS1 has 1 VCPU and 512 MB memory and RS7 has 8 VCPUs and 30 GB memory. All tests on public clouds are conducted for at least 30 runs, with multiple batches that run at different times of day and on various weekdays and weekends.

In Sec. 5.1, we start the experiments with the single machine case, which aims to accommodate the testing applications in a VM such that the workloads perform closely to the desired one. We mainly use $PM1$, which is described in Sec. 1, as the target performance. We have two local servers for hosting VMs: $VS1$ and $VS2$ are two six-core Intel Xeon CPUs at 2.67 GHz and 2 GHz, and with 24 and 32 GB memory, respectively. Both machines are running Linux 2.6.32, Xen 4.0, and NFS over a Gigabit Ethernet.

In Sec. 5.2, Matrix aims to accommodate the testing applications in a set of VMs such that the workloads perform closely to the desired one. We use a four-node physical cluster (PC) as the target performance, each of which has a 1.80 GHz Intel Atom CPU D525 (two physical cores with hyper-threading) and four GB memory connected on a Gigabit Ethernet. In the local private cloud, we use the $VS2$ to host a virtualized cluster (VC). Similar to the single machine case in Sec. 5.1, the public VCs are hosted on the Amazon EC2 and Rackspace.

In Sec. 5.3, Matrix targets at accommodating the testing applications in a set of VMs in public clouds such that the workloads perform closely to the desired one in a local cloud. We use VCs of 32 and 64 VMs in a local cloud as the target performance, and study how to configure VCs in Amazon EC2 and Rackspace cloud servers to achieve similar performance. Each VM has one VCPU and 1.5 GB memory. This way, we examine the feasibility of migrating a VC from a private to public cloud while providing the desired performance with minimized cost.

Cloud applications that are used in this work consist of *Cloudstone*, a performance measurement framework for Web 2.0 [42]; *Wikipedia* with Database dumps from Wikimedia foundation [52] and real request traces from the Wikibench web site [47]; *Darwin*, an open source version of Apple's QuickTime video streaming server; *Cloud9* makes use of cloud resources to provide a high-quality on-demand software testing service; and *YCSB (Yahoo! Cloud Serving Benchmark)*, a performance measurement framework for cloud serving systems [9].

For YCSB, the experiments use two core workloads: YCSB1 and YCSB2, both send requests following a Zipfian distribution. The major difference between YCSB1 and YCSB2 is the read:write ratio: YCSB1 is an update heavy workload with the read:write ratio of 50:50, and YCSB2 reproduces a read mostly workload with the read:write ratio of 95:5. Note that after Sec. 5.2, YCSB1 and YCSB2 are served from multiple nodes. In addition, YCSB3, YCSB4, and YCSB5 will be added into the testing set as well. YCSB3 is a 100% read workload. 95% requests of YCSB4 are read operations and mostly work on the latest records. 95% requests of YCSB5 are also read operations but it scans within 100 records.

Evaluation metrics. We use three metrics to evaluate the performance of Matrix. To measure the accuracy of the models, we define the prediction accuracy as $1-(|predicted\,value - actual\,value|/actual\,value)$. That is, the closer to 1 the better.

The goal of Matrix is to achieve a desired VM performance with minimum cost. To this end, we define two additional metrics: the RP-Cost product (RPC) as $|RP-1| \cdot (VM\ Cost)$, and the Performance Per Cost (PPC) as $RP/VM\ Cost$. In this test, we measure the cost for purchasing instances on public clouds in dollars. For RPC, a smaller value is preferred as it indicates small performance difference and cost, and for PPC, a larger value is better because of indicating better performance for the same cost.

5.1 Single Machine Case

Model Composition. We first present how Matrix analyzes applications and composes performance models. Fig. 4 demonstrates the snapshots taken by Matrix while

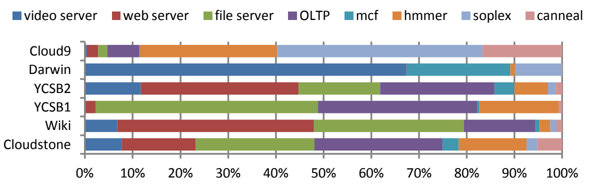


Figure 4: Application composition examples

applications are running. Let's take Darwin as an example. Darwin is about 67% like *video server*, 22% like *mcf*, 10% like *soplex*, and the possibilities to be others are very small. Although Darwin is a video streaming server, it is not 100% like the *video server* from the FileBench in the representatives. The reason is that the *video server* only emulates I/O operations and omits many CPU tasks on a video streaming server, which can be captured by

[3]A full list of Amazon EC2 instance types and prices can be found at http://www.ec2instances.info/

[4]A full list of Rackspace cloud servers can be found at http://www.rackspace.com/cloud/servers/.

Matrix with suggestion of including *mcf* and *soplex* as part of the Darwin's workload signature. Therefore, Darwin's estimated performance by the composition in Fig. 4 will be $0.67 \cdot f_{video\ server} + 0.22 \cdot f_{mcf} + 0.1 \cdot f_{soplex} + \ldots$ (Recall Eq. 2). Similarly, the sample composition of YCSB1 has a large portion of *file server*, *OLTP*, and *hmmer*. Note that these are just sample snapshots, and the composition ratio depends on the workload intensity and datasets, and may change over time.

Model Accuracy. We examine Matrix's accuracy on predicting new workloads' RP across different settings on our local VMs, the Amazon EC2 instances, and the Rackspace cloud servers. To train the RP models on the local VMs, we run the training set on $PM1$ and VMs for the RPs and training data. We collect 1,000 data points for each training workload's performance model. Each data point is generated by running the workload with a uniformly randomly configured thread (worker) count (2 to 32), working set size (5 to 32 GB), and resource allocation (1 to 8 VCPUs and 1 to 8 GB memory). Because hardware heterogeneity potentially affects performance of cloud applications [12, 33], Matrix also trains models for working on Amazon and Rackspace, instead of simply using those trained on the local VMs. The training process on the public clouds is almost identical to the one on local VMs, except the part of dynamically configuring resources. Because we can not arbitrarily adjust resources on the public clouds, the training data are collected from running them on each instance type for 100 times. Note that Matrix needs only a one time training process for modeling the gene workloads in the VMs.

For the tests on local VMs, we run each configuration for five times, five minutes per run. In Fig. 5, each column shows the average prediction accuracy and standard deviation of 30 runs (five runs for six testing applications). The same testing process is repeated on the Amazon and Rackspace at three different times and days. Thus, the public cloud results are averages of 90 runs.

Most of prediction accuracies are higher than 85% and the average value across all cases is 90.15%. The local VM tests on $VS2$ have a slightly higher accuracy (91.1%) than those on $VS1$ do (90%). On the Amazon EC2, t1.micro has the lowest prediction accuracy due to big variances on its performance. In general, larger instance types are more stable and usually lead to higher accuracies. The experiments on Rackspace also show that larger instances tend to have higher accuracy. Given the same instance type, HVM instances have lower accuracy than paravirtualized VMs, partly due to virtualization overheads. The average prediction accuracies across all Amazon and Rackspace instance types are 89.8% and 90.3% respectively.

All results pass the two-sample t-tests and are stable across all test environments. Note that we also conduct the same tests on the training set. The results show that the training applications can be identified correctly over 95% and their performance estimations have accuracy higher than 94% across all training applications.

Automatic Resource Configuration. Here, we assume a user wants to keep a desired performance of a YCSB VM with the minimum resources allocated. We run YCSB2 for one hour and change workload intensities every ten minutes. In the first ten minutes, two threads work on two millions records; The workload intensity is increased to four threads and eight millions records in the second period; eight threads and 16 millions records in the third period; Then, workload intensity is decreased to four threads and 16 millions records in the fourth period; two threads and 16 millions records in the fifth period; two threads and two millions records in the last ten minutes. Fig. 6a shows the corresponding resources and RPs as the workload intensity changes. Over the hour, the average resource savings are 37% on CPU and 55% on memory, when compared to a baseline VM which keeps using two VCPUs and four GB memory to imitate $PM1$'s setting. The average performance is 1.06 (closer to the target value) compared to 1.56 provided by the baseline VM.

In Amazon EC2, we can only change the type of an instance when it is not running. As a workaround, we use the Xen-blanket (nested virtualization) in an Amazon EC2 HVM instance (m3.2xlarge). In the one hour test, the average resource saving is about 5% on memory, compared to a baseline VM which keeps using one VCPUs and two GB memory. There is no resource saving numbers for CPU because the minimum VCPU number is one in this test. The average RP shown in Fig. 6b is about 0.95 compared to the one of 0.83 by the baseline VM. In other words, with the ability of adjusting resources to accommodate demands, Matrix can keep a desired performance with as few resources as possible.

Choosing instances among cloud providers. In this test, Matrix is used to recommend instances for running a certain workload as close to the desired performance as possible. The light, medium, and heavy workloads used here are defined as 4, 16, and 32 threads (or workers) with 8, 16, and 32 GB working set respectively. We conduct the same tests on Amazon EC2 and Rackspace cloud servers. Then, we list the most recommended instance types in Table 2 such that running certain workloads would be close to the desired performance with less cost. If the recommended instances on both sides have the same price, e.g., RS2 vs. m1.small, the one provides a higher RP will be selected. For the light workload intensity, RS3 is the most recommended type to use, which has the same price as m1.medium at $0.12 per hour. RS3 is chosen because it provides higher RP with the same price. The performance of YCSB work-

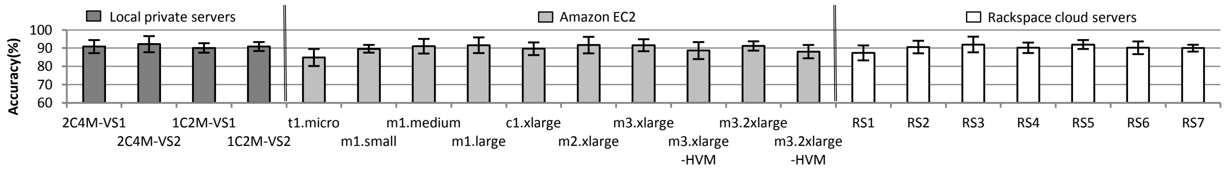
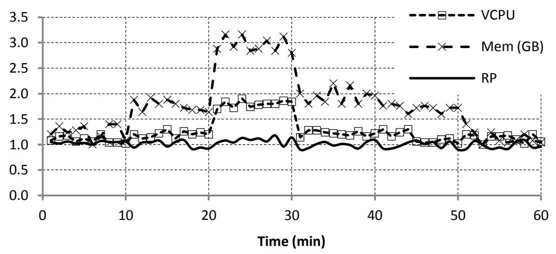
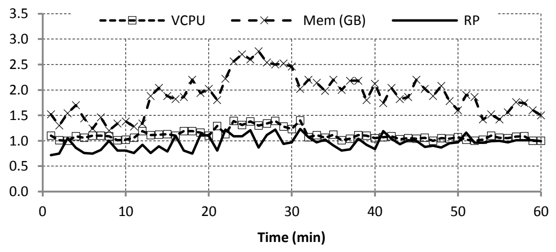


Figure 5: Accuracies on predicting performance. The labels *a*C*b*M-VS*c* on the leftmost four columns mean these tests are done on a VM with *a* VCPU and *b* GB memory hosted by our local machine VSc. The rightmost seven labels, RS1 to RS7, represent Rackspace instances from the smallest to the biggest one. Other labels represent Amazon instance types used



(a) Local



(b) Amazon EC2

Figure 6: RP changes as resources and workload intensity change. Intensities are changed every ten minutes

load is sensitive to the heap size because it affects the amount of cached contents and the frequency of flushing the cached requests in Cassandra. This effect would be more obvious if there are more write operations. Therefore, the recommendation for light YCSB1 is m1.small against RS2 because its memory space is larger.

Table 2: Most recommended instance types for running certain workloads with desired performance and less cost

Applications	Light	Medium	Heavy
Cloudstone	RS3	RS3	m1.large
Wiki	RS3	m1.medium	m1.large
YCSB1	m1.small	m1.medium	m1.medium
YCSB2	RS2	m1.medium	m1.medium
Darwin	RS3	RS3	m1.medium
Cloud9	RS3	RS3	RS3

For the medium workload intensity, the recommended Rackspace instances for YCSB1 and YCSB2 are both RS4, where the recommended Amazon instances are m1.medium. Although RS4 provides higher performance than m1.medium for these workloads, RS4 is more expensive and its RPs here are more than one than m1.medium. Therefore, the recommended instances for medium YCSB1 and YCSB2 are both m1.medium. For the rest of the applications with medium workload intensity, we mostly select the one with higher RP between RS3 and m1.medium.

For the heavy workload intensity, Cloudstone and Wiki choose m1.large against RS4 because of the higher performance with the same price. The situation for the heavy YCSB is the same as its medium case. The case of Darwin chooses m1.medium because Darwin does not need more CPU cores but more memory would be helpful. On the other hand, the heavy Cloud9 desires more CPU cores than memory. Thus, the heavy Cloud9 chooses RS3 over m1.medium.

Choosing the right instance types to minimize cost and optimize performance for a certain workload requires sophisticated analysis on application and platform characteristics. Such processes could be very time consuming without the help of Matrix.

5.2 Multi-Machine Case

Many cloud applications are designed to work on multiple computers and communicate via a network. In this section, we first start the tests on a local VC. For profiling the system under different resource configurations, the number of VMs in a VC ranges from one, two, four to eight; the VCPU numbers on one VM is varied from one to four; and the size of memory on one VM is also varied from one to four GB. In other words, we have 64 VC settings in terms of the VM numbers, VCPUs, and memory sizes. We assume all VMs in a cluster are identical and leave the heterogeneous or asymmetric clusters as future work. We collect required profiling statistics from five runs of each representative application on all 64 VC settings. In order to capture the dynamics of various workload intensities, training applications will be uniformly randomly configured with thread/worker numbers from 2 to 128 and working set sizes from 20 to 100 GB in each run, in total 9,600 data points.

For our tests on the public cloud, the instance types included as the Amazon VC instances for training and testing are *t1.micro*, *m1.small*, *m1.medium*, *m1.large*, *m1.xlarge*, and *m2.xlarge*. Similar to the local VC test, the number of VMs in an Amazon VC ranges from one, two, four to eight. Thus, we have 24 VC settings on EC2. The workload intensity is changed for profiling in the same way as it is in profiling local VCs. We also profiled VCs on Rackspace. The instance types used are RS1 to RS5. The rest processes and settings on Rackspace are similar to what we did on Amazon.

Prediction Accuracy. We first explore the accuracies on predicting RPs at clusters with different VM types and various numbers of VMs. Because of the space limit, we omit some figures. In general, the mean accuracy across all cases is 90.18% with a standard deviation of 2.55, where the mean accuracies on Amazon and Rackspace are 90.05% and 90.3% respectively.

From the accuracy tests, we found that Matrix has relatively good prediction accuracies on some applications, e.g., YCSB3 and YCSB4. Take the YCSB3, a read only testing workload in Zipfian distribution, as an example. Matrix effectively identifies this as an 100% read workload with an over 95% possibility. Among three possible distributions for pure read requests, Matrix recognizes this workload has an over 75% possibility to follow Zipfian distribution. This contributes to a relatively high accuracy for YCSB3.

To further analyze influences of representative applications in the training set, we remove five workloads at a time from the training set. Then, all models are rebuilt from the new training set. Next, we examine the accuracies on predicting RPs of applications in the testing set on a four-VM cluster whose VMs identically have four VCPUs and four GB memory. This test is repeated three times and the average accuracies are reported in Fig. 7. We remove CPU-intensive training applications first, the YCSB5 shows larger degradation than the others in the beginning because it consumes more CPU in scanning records when processing requests. When we start to remove data-intensive training workloads (the training set size is less than 15), all three testing applications drop dramatically. When we reduce the training size from ten to five, YCSB1 and YCSB5 both drop more than 20% because key genes (the 50/50 and 100/0 workload in the Zipfian distribution for YCSB1 and YCSB5 respectively) are removed. YCSB4 holds higher than the others at the training size of five because the 100/0 workload in the latest distribution, which represents most of the YCBS4, is still kept in the final five.

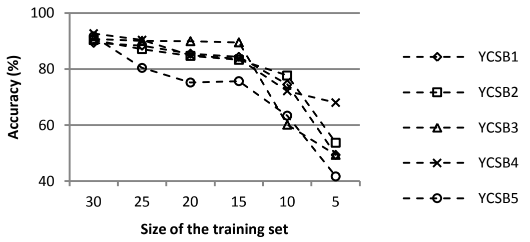


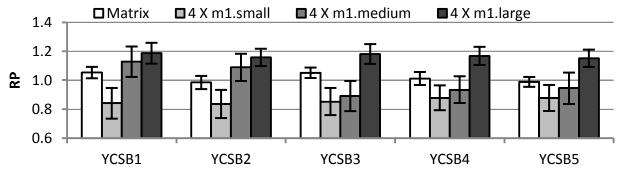
Figure 7: Accuracies on predicting RP decrease as the size of training set shrinks

VC in Private Cloud. We also verified automatically configuring a VC's resources to maintain the desired performance. The test is similar to the one in Fig. 6a, but has more dimensions in resources, e.g., number of machines, and workloads, e.g., changing workload types. Due to the space limit, we omit some figures of this test. In brief, Matrix tracks workload activities closely and is able to change VM configuration quickly and keep RPs on track.

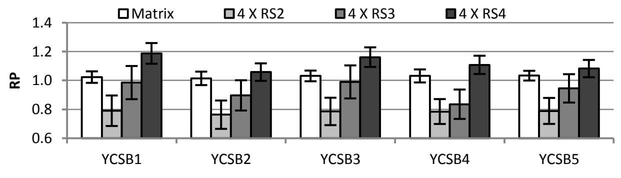
VC in Public Clouds. Here we will only change the type of instance. We did not use Xen-blanket here due to concern over the overhead of nested virtualization. In this case, we run the tests in three steps: 1) Each application in the testing set is executed for ten minutes on a VC with a randomly uniformly selected type and the number of VMs from one to eight. 2) Matrix collects required system statistics, and recommends a configuration. 3) The same application then runs on a cluster of instances closest to the recommended configuration. We repeat the above steps at the weekday daytime, the weekday nighttime, and the weekend. In addition, we change workload intensity from light, medium, and heavy for each testing application.

Fig. 8 shows the average RPs and standard deviations when we re-run testing cases with the recommended configurations as well as three fixed size VCs. Each column shows the average RP of 45 runs. Fig. 8a and Fig. 8b are results from Amazon and Rackspace respectively. All the RPs from Matrix spread between 0.88 and 1.16 with the mean of 1.02 across all cases. As it is shown in Fig. 8, using configurations suggested by Matrix makes the average RPs closer to one and smaller in variance than using the three static configurations. It leads to a low average RP value of 0.82 when using $4 \times m1.small$ or $4 \times RS2$ all the time because the medium and the heavy workloads are too intensive for it. In general, Matrix uses $4 \times m1.small$ or $4 \times RS2$ at light workloads but uses more powerful instances when workload is heavier. The average RP of all $4 \times m1.medium$ and $4 \times RS3$ cases is close to one but its standard deviation is 0.1, which is more than twice of the one of Matrix (0.04). The large variance in RPs of the $4 \times m1.medium$ and $4 \times RS2$ case comes from over-provisioning at the light workload, inadequacy at the heavy one, and the difference in the workload mix, even at the appropriate intensity. For example, Matrix uses $3 \times m1.medium$ for YCSB1 and $2 \times m1.large$ for YCSB5 at the medium workload which makes RPs closer to one than the $4 \times m1.medium$ does. When the workload is heavy, Matrix uses $2 \times m1.large$ or $2 \times RS4$ most of the time. Thus, although statically using $4 \times m1.large$ and $4 \times RS4$ has small variance values, the average RP in this case increases to 1.14.

Cost Efficiency. Here we examine the RPC and PPC values to see the cost-efficiency of each configuration.



(a) Amazon



(b) Rackspace

Figure 8: RPs when using Matrix and three static cluster settings on Amazon and Rackspace

To ease comparison, the RPC and PPC values in Table 3 are normalized to Matrix's . A smaller RPC means more cost-efficient while keeping desired RPs, and on the other hand, a higher PPC means more RP can be achieved for the same cost. In both the tests on Amazon and Rackspace, Matrix outperforms other static settings in both metrics.

Table 3: Cost efficiency (RPC and PPC) of Matrix and three static configurations on Amazon and Rackspace respectively

Amazon EC2

	Matrix	$4 \times m1.small$	$4 \times m1.medium$	$4 \times m1.large$
RPC	1.00	24.00	20.41	143.02
PPC	1.00	0.84	0.47	0.33

Rackspace cloud servers

	Matrix	$4 \times RS2$	$4 \times RS3$	$4 \times RS4$
RPC	1.00	25.33	18.67	90.54
PPC	1.00	0.78	0.68	0.52

5.3 Private to Public Cloud

In this case, we evaluate the case of migrating a virtual cluster from private to public cloud. We make the number of VMs per cluster larger than previous tests in order to test the scalability. Because our Rackspace account has a limitation on memory size at 64 GB, the results of public cloud here are all obtained from the Amazon EC2. We use 32- and 64-VM local VCs (VC32 and VC64) in this case. These local VCs are hosted on four $VS2$ servers. Each VM in one local VC has one VCPU and 1.5 GB memory, and each $VS2$ hosts 16 VMs. The training procedure on EC2 is almost the same as the previous one in Sec. 5.2, except that we extend the number of VMs to 32 and 64 in the procedure. We then verify the prediction accuracies in Amazon VCs. Because of the space limit, we omit some figures of this test. The average accuracy across different clusters is 0.89 with the standard deviation of 0.03.

We also make Matrix to recommend EC2 configurations comparable to the 32- and 64-VM local VCs for running the light, medium, and heavy testing workloads, which have 8, 32, and 64 threads and 80, 160, and 320 GB working set size respectively. We run each testing application and intensity for 30 times on a VC with $32 \times m1.xlarge$ instances for Matrix to find the matched configurations. In general, Matrix mostly uses $30 \times m1.medium$, $24 \times m1.large$, and $20 \times m1.xlarge$ instances for the VC32 at the light, medium, and heavy workloads respectively. When the cluster size increases from 32 to 64, Matrix makes the EC2 cluster to use more instances correspondingly. The configuration for the light workload is changed from $30 \times m1.medium$ to $64 \times m1.medium$. The configurations for the medium and heavy workloads become $44 \times m1.large$ and $36 \times m1.xlarge$ respectively. Using the suggested configurations gives average RPs of 1.02 with the standard deviations of 0.07 across different workload intensities. The RPs for all the cases spread between 0.88 and 1.16 with the mean of 1.03.

We also verified the PPC and RPC values of Matrix in this test. Due to the space limit, we omit a table here. According to the RPC values, Matrix costs much less than the static EC2 VC settings, especially at the VC64 tests. Further, Matrix demonstrates better PPC values than the static settings, which indicates a good performance-cost efficiency. The PPC values also show that using powerful instances may be not cost-efficient although they do provide better performance.

6 Conclusion

In this paper, we have presented Matrix, a performance prediction and resource management system. Matrix utilizes clustering methods with probability estimates to classify new cloud workloads and provide advice about what instance types will offer the desired performance level and the lowest cost. Matrix uses machine learning techniques and an approximation algorithm to build performance models, and uses them for managing on-line resources when applications are moved to the cloud. We demonstrated that our models have high accuracy, even when transitioning a distributed application from a cluster of physical machines to a set of cloud VMs. Matrix helps to keep a desired performance in the cloud while minimizing the operating cost.

As future work, Matrix may be extended to study the mapping from a local storage to a cloud one, such as the Amazon EBS. The cost model in Matrix could be more complete by including the charge on data usage. Also, we may expand the load balancing ability of Matrix to handle heterogeneous or asymmetric cluster machines and workload intensities.

References

[1] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[2] J. L. Bonebakker. Finding representative workloads for computer system design. Technical report, Mountain View, CA, USA, 2007.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[4] S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel symbolic execution for automated real-world software testing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 183–198, New York, NY, USA, 2011. ACM.

[5] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[6] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] R. C. Chiang and H. H. Huang. Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 47:1–47:12, New York, NY, USA, 2011. ACM.

[8] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea. Cloud9: a software testing service. *SIGOPS Oper. Syst. Rev.*, 43(4):5–10, Jan. 2010.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[10] S. P. E. Corporation. Spec cpu2006. `http://www.spec.org/cpu2006/`.

[11] K. Duan, S. Keerthi, and A. N. Poo. Evaluation of simple performance measures for tuning svm hyperparameters. *Neurocomputing*, 51(0):41 – 59, 2003.

[12] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. More for your money: exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 20:1–20:14, New York, NY, USA, 2012. ACM.

[13] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 99–112, 2012.

[14] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 18:1–18:14, 2011.

[15] Z. Hill, J. Rowanhill, A. Nguyen-Tuong, G. Wasson, J. Knight, J. Basney, and M. Humphrey. Meeting virtual organization performance goals through adaptive grid reconfiguration. In *Grid Computing, 2007 8th IEEE/ACM International Conference on*, pages 177–184, 2007.

[16] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, 2011.

[17] R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1 edition, Apr. 1991.

[18] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Bridging the tenant-provider gap in cloud services. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 10:1–10:14, 2012.

[19] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. 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.

[20] A. Kopytov. Sysbench. `http://sysbench.sourceforge.net/index.html`.

[21] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –10, 2010.

[22] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling virtualized applications using machine learning techniques. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, VEE '12, pages 3–14, New York, NY, USA, 2012. ACM.

[23] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf. Monalytics: online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 141–150, New York, NY, USA, 2010. ACM.

[24] S. Lacour, C. Perez, and T. Priol. Generic application description model: toward automatic deployment of applications on computational grids. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 4 pp.–, 2005.

[25] H. Li, G. Fox, and J. Qiu. Performance model for parallel matrix multiplication with dryad: Dataflow graph runtime. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 675–683, 2012.

[26] M. Maurer, I. Brandic, and R. Sakellariou. Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 368–375, 2012.

[27] R. McDougall, J. Crase, and S. Debnath. Filebench. `http://sourceforge.net/projects/filebench/`.

[28] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, VEE '05, pages 13–23, New York, NY, USA, 2005. ACM.

[29] Y. Nakajima, Y. Aida, M. Sato, and O. Tatebe. Performance evaluation of data management layer by data sharing patterns for grid rpc applications. In *Euro-Par 2008 Parallel Processing*, volume 5168 of *Lecture Notes in Computer Science*, pages 554–564. Springer Berlin Heidelberg, 2008.

[30] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 237–250, New York, NY, USA, 2010. ACM.

[31] O. Niehörster, A. Brinkmann, A. Keller, C. Kleineweber, J. Krüger, and J. Simon. Cost-aware and slo-fulfilling software as a service. *Journal of Grid Computing*, 10:553–577, 2012.

[32] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 219–230, 2013.

[33] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui. Exploiting hardware heterogeneity within the same instance type of amazon ec2. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 4–4, Berkeley, CA, USA, 2012. USENIX Association.

[34] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17:412–421, July 1974.

[35] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise grids and clouds. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 50–57, 2009.

[36] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, and N. J. Wright. Evaluating interconnect and virtualization performance for high performance computing. In *Proceedings of the second international workshop on Performance modeling, benchmarking and simulation of high performance computing systems*, PMBS '11, pages 1–2, New York, NY, USA, 2011. ACM.

[37] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 137–146, New York, NY, USA, 2009. ACM.

[38] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 5–14, 2006.

[39] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001.

[40] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Comput.*, 12(5):1207–1245, May 2000.

[41] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559 –570, june 2011.

[42] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *The first workshop on Cloud Computing and its Applications*, CCA '08, 2008.

[43] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 953–966, New York, NY, USA, 2008. ACM.

[44] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 31–44, New York, NY, USA, 2007. ACM.

[45] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 285 –294, june 2012.

[46] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell. Modeling virtual machine performance: challenges and approaches. *SIGMETRICS Perform. Eval. Rev.*, 37(3):55–60, Jan. 2010.

[47] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. `http://www.globule.org/publi/WWADH_comnet2009.html`.

[48] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 423–436, New York, NY, USA, 2012. ACM.

[49] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, 2011.

[50] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. Probabilistic performance modeling of virtualized resource allocation. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 99–108, New York, NY, USA, 2010. ACM.

[51] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the deployment of computations in the cloud with conductor. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation*, pages 367–381. USENIX, 2012.

[52] Wikimedia Foundation. Wikipedia:Database download. `http://en.wikipedia.org/wiki/Wikipedia:Database_download`.

[53] D. Williams, H. Jamjoom, and H. Weatherspoon. The xen-blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 113–126, New York, NY, USA, 2012.

[54] T. Wood, L. Cherkasova, K. Ozonat, and P. 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, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[55] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner. Justrunit: experiment-based management of virtualized data centers. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 18–18, Berkeley, CA, USA, 2009. USENIX Association.