uPredict: A User-Level Profiler-Based Predictive Framework in Multi-Tenant Clouds

Hamidreza Moradi, Wei Wang, Amanda Fernandez and Dakai Zhu
The University of Texas at San Antonio
{hamidreza.moradi, wei.wang, amanda.fernandez, dakai.zhu}@utsa.edu

***Abstract*—Accurate performance prediction for cloud applications is an essential component to support many cloud resource management and auto-scaling policies. However, most existing studies on performance prediction for cloud applications in multi-tenant clouds are at the system level and may require access to performance counters in hypervisors. In this work, we propose *uPredict*, a *user-level* profiler-based performance predictive framework for single-VM (virtual machine) applications in multi-tenant clouds. We designed three micro-benchmarks to assess the contention of CPUs, memory and disks in a VM, respectively. Based on the measured performance of an application and micro-benchmarks, the application and VM-specific predictive models are derived by exploiting various regression and neural network based techniques. These models can then be used to predict the application's performance using the in-situ profiled resource contention with the micro-benchmarks. We evaluated uPredict extensively with representative benchmarks from PARSEC, NAS Parallel Benchmarks and CloudSuite, on a private cloud and two public clouds. The results show that the average prediction errors are between 10.4% to 17% for various predictive models on the private cloud with high resource contention, while the errors are within 4% on public clouds. A smart load-balancing scheme powered by uPredict is presented and can effectively reduce the execution and turnaround times of the considered application by 19% and 10%, respectively.**

I. Introduction

Cloud computing has been adopted by many organizations as their main computing infrastructure due to its low cost of ownership and elasticity [32]. However, applications running on the clouds usually share hardware resources with other virtual machines (VMs) and applications from different cloud users/tenants. Such hardware resource sharing among multiple tenants causes resource contention, which in turn degrades the performance of applications running on clouds [23]. Moreover, the resource contention can vary due to changes in co-located VMs and their applications, which makes a target cloud application experience uncontrolled performance variations and fluctuations at runtime [22, 31].

However, to maximize the cost benefits of cloud deployments with optimal resource allocation [25, 26], or to satisfy the timeliness requirements of time-sensitive applications [7], cloud users may need to have an accurate knowledge of the performance of their applications. For such a purpose, cloud users need the capability to accurately predict the performance of their applications under various levels of resource contention at runtime. While there have been many studies proposed to predict an application's performance under hardware resource contention [11, 27, 28, 30, 33, 43], these studies usually relied on the access to and control over the underlying execution environment, which makes them not applicable for cloud users.

Cloud services are typically offered to users as black boxes, where a user cannot control the cloud execution environment to specify the set of VMs/applications that should be executed together to share hardware resources. As a result, it is difficult for a cloud user to obtain an isolated execution environment to profile an application's contention sensitivity on the cloud service's hardware as did in prior work [11, 27, 33]. Moreover, as cloud users cannot select the co-runners of their applications, they have to measure or estimate the severity of resource contention and the associated impacts on their applications' performance during execution. Given that cloud users generally have no direct access to the underlying hardware components and virtual machine hypervisors, they usually cannot utilize common execution inspection tools used by prior studies, such as hardware performance monitoring units (PMU), to obtain accurate estimations on the impact of the contention [28, 30, 43].

Therefore, it is imperative to design and develop performance prediction schemes for ordinary cloud users. Although some recent studies have addressed this problem, there are still some limitations. In [46], Yadwadkar et al. developed *PARIS*, which exploits resource profiling information to predict the performance of an application in a VM when it is deployed on different public cloud services. Similarly, Scheuner and Leitner employed micro-benchmarks to test and predict the performance of different types of VMs across public cloud services [36], where a large number of micro-benchmarks have been deployed. Although these studies can predict an application's *average* performance on various VMs and/or different cloud services, they cannot be utilized to predict the *in-situ* performance of an application while taking the runtime resource contention into consideration. An in-situ performance prediction model enables the users to schedule their tasks/requests to the VMs that provide the best performance during execution and thus to improve their quality of services (as illustrated with the case study in Section V-E).

In this paper, focusing on single-VM applications, we propose ***uPredict***, a *user-level* predictive framework for multi-tenant Infrastructure-as-a-Service (IaaS) clouds. To profile and assess the resource contention of a VM, three micro-benchmarks are devised to probe its CPUs, memory and disks. To establish the application-specific relationship between its

978-1-7281-1099-8/20/$31.00 ©2020 IEEE
DOI 10.1109/IC2E48712.2020.00015

performance and the profiled resource contention, the micro-benchmarks are executed right before the target application in a given VM to collect the profiling and performance data while the colocated unknown VMs/applications on the same host may change over time.

With the profiled resource contention of a VM by the micro-benchmarks and the measured performance of an application, the application/VM-specific performance predictive models can be built to learn the application's sensitivity to the contention of different resources. In this work, we considered both regression and machine learning based techniques, including 2-degree polynomial regression [19, 44, 49, 50], Support Vector Regression (SVR) [17] and Neural Networks (NN) models [18]. Once an application/VM specific predictive model is derived, the micro-benchmarks can be executed in the current execution environment and the in-situ profiled resource contentiousness can be fed into the model to predict the application's execution times at runtime.

We have evaluated *uPredict* extensively using the representative benchmark applications from PARSEC [5], NAS Parallel Benchmarks (NPB) [6] and CloudSuite [13] in three different clouds, including a private cloud with OpenStack, Amazon Web Services (AWS) [4] and Google Compute Engine (GCE) [16]. The prediction errors (i.e., accuracy) of the considered predictive models were evaluated. The results show that, for the considered applications and VMs, the NN-based models are generally more accurate with the average prediction errors being 10.4%, 2.11% and 3.09% for the private cloud, AWS and GCE, respectively. In comparison, the polynomial regression and SVR models perform slightly worse in the private cloud that has high resource contention with the average prediction errors being 17% and 13%, respectively. However, on AWS and GCE where the level of contention is lower than our private cloud, the polynomial regression models and SVR models have almost the same prediction errors on average compared to those of NN-based models. Moreover, we would like to point out that, the NN-based models require hyperparameter optimizations, which can introduce larger training overheads.

As an application of *uPredict*, a case study on load-balancing for two VM servers on two different host machines was presented. For comparison, we considered a simple queue-based load-balancing scheme that makes load distribution based on the queue length (i.e., the number of requests) on each VM server without considering their resource contention. The results show that the *uPredict* based load-balancing can achieve about a 19% reduction in average application execution times and a 10% reduction in their turnaround times.

The rest of the paper is organized as follows. Section II reviews closely related work. Section III discusses the micro-benchmarks that are devised to profile the resource contention of a VM in a multi-tenant environment. Section IV presents the proposed *uPredict* framework and several predictive models. The experimental setups and evaluation results are discussed in Section V. Section VI points out the limitations of this study and our future works. Section VII concludes the paper.

II. Closely Related Work

Contention-aware Performance Prediction from Cloud Service Provider's Perspective. There have been research works on predicting application performance under resource contention from the perspective of cloud service providers and data center operators. Paragon is heterogeneity and interference-aware data center scheduler [11]. Quasar estimated the resources that a data center application required to meet its QoS goals by profiling its performance on specific hardware running with specific micro-benchmarks [12]. Bubble-up characterized the sensitivity of a data center application and predicted the application's performance under contention by injecting pressure into the memory system [27]. Bubble-flux dynamically injected pressure into the memory system to measure the application's instantaneous sensitivity to contentions using readings from PMUs [47]. ESP predicted the performance impact of contentions for a known set of applications using regularization [28]. These cloud-provider prediction methodologies typically required controlling the execution environment in their profiling phases, including directly specifying the co-running tasks used in the profiling. Many of these studies also assumed a known set of applications that might be executed and/or required accesses to low-level hardware PMUs. Our work, however, aims at performance prediction for ordinary cloud users who have no control of the execution environments, no knowledge of the co-running applications and no access to hardware PMUs.

Performance Prediction from the User's Perspective. There were also studies on cloud performance prediction techniques for cloud users. Scheuner and Leitner employed micro-benchmarks to predict the average performance of different types of VM instances across public cloud services [36]. They considered 23 different micro-benchmarks and validated the methodology against only two applications. PARIS predicts the performance of an application when it is deployed on different types of cloud instances [46]. PARIS did not consider the impact of resource contention and experienced up to 50% RSME (Root Mean Squared Error). Ernest built performance models based on the behavior of job on small inputs and then predicted the performance on larger data sets [45]. Friese et al. presented a novel hierarchical critical path analysis methodology to predict the performance of irregular applications [15]. Unlike uPredict, these studies did not intend to predict the performance of cloud applications under currently observed level of resource contention in multi-tenant clouds.

III. VM Performance Profiling

In a multi-tenant cloud, a VM normally shares and contends for the underlying hardware resources with other colocated VMs and applications. The deliverable performance to user applications by a VM depends heavily on resource contention in CPUs, memory and disks, Given that ordinary cloud users do not have access to PMUs and control over co-running VMs, we focus on user-level profiling techniques to obtain the in-situ resource contention of a VM. Note that, although cloud

users can retrieve the utilization of various (virtual) resources from a VM, such information normally does not reflect the impact of resource contention.

Therefore, to infer the perceivable performance from the perspective of cloud user applications regarding the resource contention of a VM, we employ several micro-benchmarks to assess its resource contention due to interference from other collocated VMs and applications on the same host machine. Intuitively, the slowed progress of a micro-benchmark reflects the increased contention for the corresponding resource.

Although there have been many micro-benchmarks, they are usually designed for other use cases and cannot be directly applied to evaluate resource contention. Therefore, based on the open-source benchmark suite lmBench3 [24], we designed our micro-benchmarks to profile resource contention through stressing the particular resource. Prior work has shown that resource contention mainly happens in CPUs, memory, storage and network resources [11]. Note that, for single-VM applications, the impact of network contention is negligible. Hence, we designed three micro-benchmarks for uPredict to probe the contention of CPUs, memory and disks, respectively. In the following paragraphs, we explain the detailed design of our micro-benchmarks.

A. Micro-benchmarks

CPUs: For contention in CPUs, a multi-threaded micro-benchmark is designed to stress the performance of the virtual CPUs of a given VM. Here, each thread maintains an *in-register counter* that is initiated with zero. During execution, the thread repeatedly increments the counter for a fixed amount of profiling execution time specified by the user. Such in-register operations ensure that the micro-benchmark's performance is not affected by memory at runtime and thus examine the contention in CPUs to the maximum extent. The number of threads deployed in this micro-benchmark will be equal to the number of virtual CPUs of the target VM. The total number of increment operations carried out by all threads for the in-register counter will be recorded as the progress of this micro-benchmark. In the end, this number (c_{cpu}) will be used as the indicator of the progress of this benchmark and the contention level for the virtual CPUs in the target VM.

Memory: Similarly, the memory micro-benchmark will try to stress the memory bandwidth of the target VM to the maximum extent. This micro-benchmark accesses a 2GB array with a stride of 128 byes. The objective of such a memory access pattern is to ensure that each data access is issued to off-core memory rather than the caches. Again, the number of threads in the micro-benchmark equals the number of virtual CPUs in the VM and each thread will access an equal portion of the array and increases the local counter by one for each access. The total number of memory accesses by all threads in a specific amount of profiling time for this micro-benchmark (c_{mem}) will provide us the insight into the memory contention and its impacts on performance experienced by user applications in the target VM.

Disk I/Os: For the I/O performance of the target VM, we design the disk micro-benchmark that reads 256MB data from the VM's disk with the page size of 4KB, and by each disk access, the local counter value will be incremented. During the execution of this micro-benchmark, the OS file cache should be disabled so all file operations need to access the disk. Four threads are used for this micro-benchmark to fully exercise the disk without causing too much internal I/O contention. Again, the total number of disk access operations within a specific profiling time (c_{disk}) is used as the progress of this disk micro-benchmark to assess the contention level of the VM's disk operations.

These micro-benchmarks will be invoked sequentially right before the execution of a user's application to get the in-situ resource contention for the respective resources.

B. The Length of Profiling Executions

Although it is desirable to reduce profiling overhead, short executions of the micro-benchmarks may not be able to completely capture the actual severity of resource contention. For the experiments in this paper, we executed each micro-benchmarks for 3 seconds to ensure that the actual severity of contention was properly captured. We have also conducted a sensitivity test on the effects of profiling length on the accuracy of uPredict and the details can be found in [29]. The test shows that 3 seconds indeed can provide accurate profiling results, while shorter profiling length may also suffice.

IV. uPredict and Predictive Models

A. Overview of uPredict

Figure 1 gives an overview of *uPredict* and illustrates the workflow of performance modeling and prediction for an application running on a VM in a multi-tenant cloud. There are two major phases in *uPredict*: the *training* and *prediction* phases. The first step in the training phase is to collect the training performance data. Here, in each iteration, our three micro-benchmarks are executed first for a fixed amount of time and their progress being denoted as $\{c_{cpu}, c_{mem}, c_{disk}\}$ to assess the in-situ contentiousness of CPUs, memory and disks of a VM. Then, the target application is executed right after the micro-benchmarks with its execution time being denoted as t_{app}. The progress of the micro-benchmarks and the execution time of the application will form a data tuple $\{c_{cpu}, c_{mem}, c_{disk}, t_{app}\}$, which represents the implicit relationship between the application's performance and the resource contention profiled by the micro-benchmarks.

A set of training data tuples needs to be collected first by repeating the above process for the target application and VM, where the resource contention from other VMs and applications on the same host machine can vary. Then, the data tuples can be used to train various applications and VM specific performance predictive models based on different regression and neural network techniques, as discussed next. The number of data tuples in the training set can affect the accuracy of the derived models and the trade-offs are evaluated in Section V. The second phase of performance

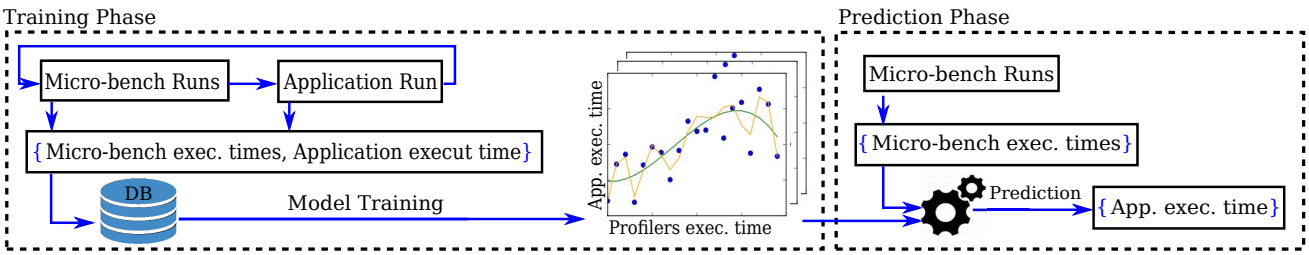


Fig. 1: Overview of uPredict and its workflow for performance modeling and prediction.

prediction utilizing the derived predictive models is detailed in Section IV-C.

B. *Predictive Models in* uPredict

The key step in the training phase is to learn the relationship between the application execution time and the resource contention represented by the micro-benchmarks' access counter values from the set of collected data tuples and derive a predictive model. That is, the parameters of the function f in Equation (1) need to be learned.

$$t_{app} = f(c_{cpu}, c_{mem}, c_{disk}) \quad (1)$$

where the exact form of the function f and its parameters depend on the specific regression or machine-learning technique being used.

In other words, we use the micro-benchmark's progress as the features to predict the execution time of the target application. Intuitively, the progress of the micro-benchmarks is the indicator of contention severity of different resources, which are in turn used to predict the execution time of the application as shown in Section IV-C. Note that, the execution time of an application may also depend on its input data. In this work, we assume that an application's execution time is affected only by the resource contention from other collocated VMs and their applications on the same host machine, where the input data for the application in each execution has the same or similar size. It has been shown that many recurring cloud applications are indeed repeatedly executed with similar workloads [2, 3, 14]. Moreover, the predictive models developed in this work can be easily extended to consider an application's input data, especially when its data size has a known (e.g., linear) relation with its execution time.

Moreover, as applications have different behaviors and sensitivities to resource contention when running in a given VM, application and VM specific predictive models will be derived for each application and its underlying VM. In what follows, the details of several regression and neural network based modeling techniques employed by *uPredict* are presented.

1) Polynomial-Regression based Models: We first considered polynomial-regression based predictive models, which usually take a short amount of time to train and predict. Consequently, if polynomial models can provide good prediction accuracy for an application running in a VM, there is no need to employ other more complex and heavy-headed machine-learning models. To ensure that polynomial-regression models are thoroughly evaluated, we explored four regression (model training) techniques, which are Elastic Net Regularization [50], Lasso Regression [44], Ridge Regression [19], and Stochastic Gradient Descent (SGD) [49]. The exact parameters for training these models can be found in Section V. Additionally, we have conducted experiments with both linear regression and 3-degree polynomial regression. However, for the considered benchmark applications, their resulting models perform inferior comparing to 2-degree polynomial models. Therefore, we only report the results of 2-degree polynomial predictive models in this paper.

2) Support Vector Regression (SVR) based Models: We also considered Support Vector Regression (SVR) based models [17], which may potentially provide higher accuracy than polynomial models but with larger training and prediction cost. SVR is based on the popular machine-learning classifier, Support Vector Machine (SVM), with the introduction of an alternative loss function (in our case, the popular epsilon insensitive function) [39]. The main benefit of SVR is that it allows us to build more complex and non-linear models within reasonable amount time, as an application's behaviors running in the clouds may not always be expressible with polynomial or linear equations of resource contention [17].

3) Neural Network (NN) based Predictive Models: In addition to SVR, we also considered the Neural Network (NN) based models [18, 42]. In uPredict, NN models are configured to conduct regression analysis. These models are more generic than SVM and can approximate nearly any function, potentially allowing *uPredict* to model any behaviors of an application running in a VM with higher training costs [9, 20].

However, we have observed that the accuracy of NN models can be significantly affected by their structures, that is, the number of layers and the number of neurons in each layer. Even for the same set of training data, the worst NN structure can have the prediction error to be more than 10 times higher than the best one. Therefore, training NN models with good accuracy is not simply just training a NN model with a *fixed structure*, it also involves optimizing the structures of the NN models. Moreover, the best NN model structure also varies for different applications and VMs, implying that *uPredict* needs to individually optimize the model structure for each pair of application and VM.

This optimization process should be automated so that ordinary cloud users, who do not have expertise in machine learning, can apply uPredict to a new application and/or VM.

To automatically optimize NN structures, we employed hyperparameter optimization techniques, including Tree-structured Parzen Estimator (TPE) approach and Bayesian Optimization (BO) [8, 40]. Hyperparameters refer to the NN parameters defining the number of layers and the number of neurons in each layer of a NN model. Both optimization techniques conduct a search in the search space of the NN models to find a structure with good accuracy.

This search space defines the maximum number of layers and neurons per layer that can be used when training NN models. The TPE technique explores the search space using a tree structure following the accuracy distribution obtained from previously sampled points in the search space. The BO searches for the high-accuracy NN structures through the Gaussian Process (GP), which is a non-linear regression technique. BO uses GP to build a regression model with the already-explored NN structures and their accuracies. The regression model is then used to predict a potentially better NN structure until a fixed number of NN structures are searched. To conduct the hyperparameter search, both TPE or BO require knowing the accuracy of the models trained with the already-explored NN structures. Therefore, besides the training data that are used to train the NN models, a separate set of cross-validation data also need to be collected to assess the accuracy of the trained models.

It is also worth noting that the effectiveness of the hyperparameter optimization depends on the definition of the search space. It is commonly recommended that the number of neurons per layer is typically "no more than 1/30 of the number of training cases" [35]. As our training sets only contain up to 1,000 data samples for each benchmark application in the given VMs, we set the maximum neurons per layer to be 35 in *uPredict*. As neural networks with two hidden layers (four total layers) can be fully general, we define the maximum number of layers of uPredict's NN models to be 5 [41]. The extra layer is added to accommodate the cases where the maximum number of neurons defined above is not large enough. In summary, uPredict employs a NN structure search space of maximum 5 fully-connected layers and maximum 35 neurons per layer. The final optimized NN models for different applications may not have the same number of layers and neurons at each layer.

C. Performance Prediction in uPredict

In the prediction phase, the micro-benchmarks are first executed sequentially to profile the in-situ contentiousness of a VM's CPUs, memory and disks in the current execution environment, respectively. Such profiled resource contention by the micro-benchmarks is used to estimate the contention to be experienced by the application. The access counter values, c_{cpu}, c_{mem} and c_{disk}, are then fed into a trained model f for the application to predict its execution time at runtime.

For the benchmark applications used in our evaluations, they typically take less than one hour to execute and our observation shows that the resource contention is less likely to change significantly within such a short period time, especially on the public clouds. However, when an application does experience a change in resource contention during its execution, the prediction accuracy of the derived models can be negatively affected with much higher prediction errors as shown in the evaluation results (see Section V).

For long-running applications, they will be more likely to experience changes in resource contention during their executions and a periodically re-profiling technique may be deployed to catch such changes. However, exploring such a periodic re-profiling option would require significant modifications to the model building and prediction process (to consider, for instance, re-profiling intervals), which is well beyond the scope of this paper and will be investigated in our future work.

V. Experimental Evaluations

This section provides the experimental evaluation results of *uPredict*, including the prediction accuracy of *uPredict* on both private and public clouds, as well as the performance benefits brought by *uPredict* when it was applied to a case study of cloud resource management on load balancing.

A. Experiment Setups

Representative Benchmark Applications: We have considered a total of 17 benchmarks from PARSEC [5], NAS Parallel Benchmarks (NPB) [6] and CloudSuite [13] in our evaluations. Eight are from PARSEC, including *streamcluster*, *blackscholes*, *bodytrack*, *canneal*, *facesim*, *ferret*, *swaptions* and *dedup*. For these PARSEC benchmarks, their native inputs were used in the experiments. Five are chosen from NPB, which are *ua*, *lu*, *sp*, *ep* and *bt*, and they used the class C data inputs. The other four are from CloudSuite, including *In-Memory Analytic*, *Graph Analytic*, *Web Search* and *Data Serving*. Here, the large data inputs were used for *In-Memory Analytic*, while *Graph analytics*, *Web Search* and *Data Serving* benchmarks used the default data inputs.

These 17 benchmarks are representative and cover a wide range of applications running in various clouds. In each benchmark suite, the selection of these benchmarks is a combination of technical difficulties (e.g., compilation problems), benchmarks' resource requirements (needs of more memory) and budget limitation to run the costly experiments on AWS and GCE clouds. For all the 17 selected benchmarks, sixteen worker threads were created in their executions.

Clouds and VM Configurations: First, for the private cloud, we utilize an Ubuntu 16.04 server with two Intel Xeon E5-2630 processors (for a total of 16 cores) and 128GB memory that has OpenStack Ocata installed. Given that the selected benchmarks include parallel and data/graph analytic applications, we created a VM of 16 VCPUs and 16GB memory on OpenStack to execute these benchmarks. Moreover, to introduce resource contention into the private cloud, up to seven (7) background VMs (with the same VCPU and memory configuration as the target VM) were randomly created at runtime, which executed either CPU- or memory-intensive synthetic applications from iBench [10]. The background VMs and their applications changed every 2 hours.

For public clouds, we evaluated *uPredict* on AWS EC2 and Google Cloud Engine (GCE). In AWS EC2, a single *m5d.4xlarge* VM was used to execute the selected benchmarks. *m5d.4xlarge* is the latest general purpose VM instance with 16 CPUs and 64GB memory [4]. The VM is configured to use an 80GB standard EBS SSD drive. Non-dedicated VM instances were used to ensure the existence of resource contention. For GCE, we used a single *n1-standard-16* VM to execute the selected benchmarks, where *n1-standard-16* is the standard VM instance with 16 VCPUs and 60GB of memory [16]. The VM is also configured with an 80GB SSD drive. Again, the background VMs and their applications were managed by GCE and were unknown to us. Both the selected VM types in the public clouds closely match CPUs and memory of the VM in our private cloud to make the evaluation results from the private and public clouds more comparable. For all the experiments on the three clouds, we used Ubuntu Server 16.04 as the OS for the created VMs.

Data Collection: As illustrated in Figure 1, a key step in the training phase of *uPredict* is to collect experimental data regarding the execution times of the benchmark applications and those for the micro-benchmarks. From Section III, the three micro-benchmarks are invoked sequentially right before the execution of a user's application to get the in-situ resource contention, which has the total profiling overhead of roughly 9 seconds in each iteration. For the private cloud, with up to seven (7) background VMs and associated applications, we have run the 17 benchmark applications individually with the micro-benchmarks for a total of roughly 70 days. For each benchmark application, more than 1,000 data points have been collected and the first 1,000 were used in the evaluations.

For the PARSEC benchmarks that have relatively short execution times, we have executed them for about 10 days on both AWS and GCE to collect more than 1,000 data points in each cloud setting. Again, the first 1,000 were used in the evaluations. For the benchmarks in NPB and CloudSuite that take more time for executions, we run them for about 20 days on both AWS and GCE, where 777 and 688 data points have been collected for each of these benchmark applications on the two clouds, respectively, and all these data points were used in the evaluations.

For each benchmark application, 80% of the collected data points are randomly selected to train and optimize the regression and neural network models. Within these 80% data points, 80% (i.e, 64% of the total data) were used as the training data set, and 20% (i.e., 16% of the total data) were used as the cross-validation data set to aid hyperparameter optimizations. The remaining 20% of the total data points were used as testing data to evaluate the prediction accuracies (i.e., errors) of the derived predictive models.

Implementation and Training of the Predictive Models: We used the scikit-learn version 0.19.2 library [37] to implement the four different 2-degree polynomial regression models and the SVR model. In particular, for the *ElasticNet* and *Lasso* algorithms, we used an alpha of 1 as a constant and a tolerance value of 0.001 for optimization. For the *Ridge* algorithm, we used the same tolerance value and an alpha value of 1 as regulation strength. For the SGD algorithm, we used the following settings: squared loss, penalty L2, alpha value of 0.0001, L1 ratio of 0.15, epsilon as 0.1 and eta as 0.01. For the SVR model, we used RBF (Gaussian) kernel with a C value of 1000. For all the aforementioned algorithms, we set the maximum number of iterations to 10,000. These models are denoted as *2-D poly: ElasticNet*, *2-D poly: Lasso*, *2-D poly: Ridge*, *SGD* and *SVR* in the result figures, respectively.

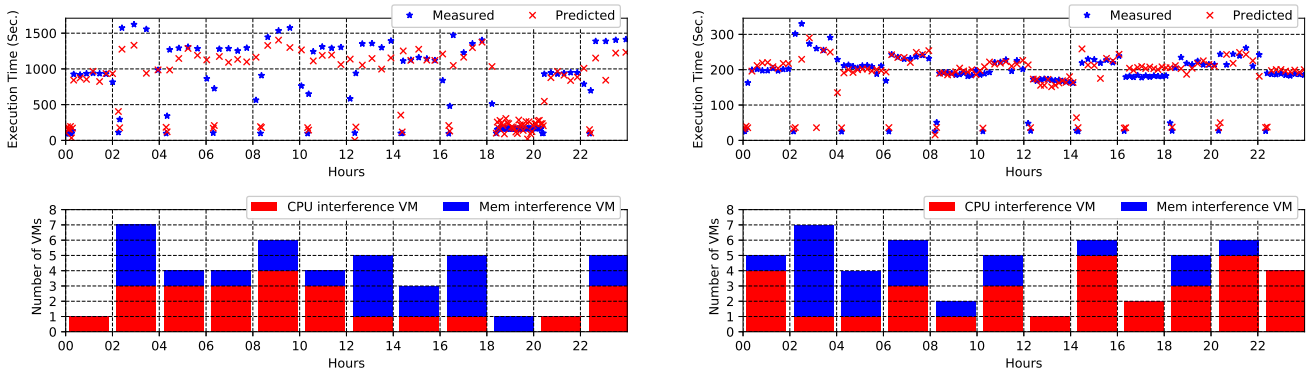
The NN-based models are implemented using TensorFlow version 1.12 [1]. We evaluated both a fixed NN structure and the automatically optimized NN structures for each benchmark application as described in Section IV-B3, to demonstrate the importance of hyperparameter optimization for NN models. Here, the fixed structure had 5 fully-connected layers and 35 neurons per layer, which is the same as the largest NN structure of the hyperparameter optimization search space as defined in Section IV-B3. This largest NN structure is chosen with the assumption that a more complex NN model would be expected to provide better prediction accuracy.

For hyperparameter optimization, we employed two libraries, HyperOpt version 0.1.1 [21] (for TPE optimization) and Scikit-optimize version 0.5.2 [38] (for Bayesian Optimization). Both of the hyperparameter optimization libraries have been set to 200 iterations for finding the high-accuracy parameters. Our evaluations show that increasing the number of iterations up to 1,000 will not significantly improve the prediction accuracy (less than 2 percent) for the resulting NN models. However, with the optimization time having a linear relation with the number of iterations, the training time can increase by up to 5 times for 1,000 iterations. The resulting NN models are denoted as *NN:HyperOpt* and *NN: SkOpt*, respectively.

B. Validation of uPredict

We first validated the effectiveness of *uPredict* with the execution of two PARSEC benchmarks, *Streamcluster* and *Cannel*, which have higher prediction errors as shown later, on our private cloud. Here, Figure 2 shows the measured (actual) execution times (the blue star points in the top figures) for the two benchmarks as well as the corresponding number of background VMs and their applications (in the bottom figures) for the duration of 24 hours. Clearly, the execution times of the benchmarks can vary drastically (more than 10 times) due to variations in the severity of resource contention caused by the background VMs and their applications on the same host machine. Therefore, it is imperative to develop a user-level framework and tools for ordinary cloud users to get reasonably accurate performance prediction for their applications and to support their cost-effective planning and operations.

We can see from the figures that the execution times for *Streamcluster* and *Cannel* can be as low as around 90 and 30 seconds, respectively, at the beginning of each 2-hour interval. This is due to the fact that, when the number of background VMs changes at each 2-hour interval, the executions of the



a. Streamcluster b. Canneal

Fig. 2: Measured and predicted execution times for *Streamcluster* and *Canneal* with the background VMs on the private cloud.

interfering applications in all background VMs stop for the first 5 minutes, during which the level of resource contention is rather low. Moreover, if the resource contention is low for a given 2-hour interval (e.g., from 18 to 20 hours for *Streamcluster*), the application took less time to execute and more data points were collected compared to the intervals with higher levels of contentions.

The predicted execution times (the red x points) utilizing the derived SVR models for the two benchmarks are also shown in the top two figures, respectively. Although the predicted execution times have several outliers for both benchmarks due to the limitations of the predictive model, especially during the transition period of changing background VMs and applications, it can be clearly seen that the pattern (or trend) of the predicted execution times closely follows that of the measured ones. Such patterns match the severity of resource contention due to the background VMs and their applications as shown in the bottom figures. Therefore, we can also say that this experiment validates our hypothesis on the devised micro-benchmarks, which can properly assess the resource contention in the target VM at runtime.

C. *Evaluation of* uPredict *in a Private Cloud*

For the private cloud where the resource contention is rather high with the controlled background VMs and applications, Figure 3 shows the prediction errors of the considered seven predictive models in *uPredict* for all the 17 benchmark applications. Here, the solid bars indicate the average and the associated vertical lines show the 95-percentile of the prediction errors. As explained earlier, for each benchmark, 80% of the data points were utilized to train/derive its application/VM-specific predictive models while the remaining 20% were used to test their prediction accuracy.

First, for average prediction errors, the two NN-based predictive models with hyper-parameters optimizations (i.e., *NN:HyperOpt* and *NN:SkOpt*) perform the best for almost all benchmark applications with lower than 20% errors (except *streamcluster* has 28% error). Moreover, the overall average prediction errors by considering all 17 benchmark applications were only 10.4% for the NN-based models. This indicates that the proposed uPredict with NN-based models can indeed provide quite accurate performance predictions for applications in multi-tenant clouds even with high resource contention (where our private cloud has up to 7 background VMs). However, without hyperparameter optimizations, the fixed structure (i.e., 5 layers and 35 neurons per layer) NN models can perform rather worse with the overall average prediction errors being 60% (and up to 154% prediction errors for some benchmark applications), which is not shown in the figure.

For comparison, all four polynomial-regression based predictive models perform relatively worse with the overall prediction errors for all benchmarks being around 17%, which is about 8% higher than those of the NN-based models. One possible reason for the worse performance of the polynomial regression models compared to that of the NN-based models is that, for certain applications, the relationships between the profiling results from the micro-benchmarks and the actual execution times of the applications are not necessarily polynomial. Neural networks, on the other hand, have shown great potential in finding relationships that are neither linear nor polynomial [48]. For SVR models, while they performed relatively better than the polynomial regression models for most benchmark applications, their overall prediction accuracy is still behind the NN-based models. The results indicated that SVR models might be able to find non-polynomial relationships, however, they are not as powerful as the NN-based models for predicting the performance of cloud applications in clouds with high resource contention.

On the other hand, the prediction errors of the predictive models are application-dependent and vary with large fluctuations. For several applications (such as *web, data, ep, blackscholes* and *swaptions*), their 95-percentile prediction errors can be lower than 20% for all the considered predictive models. However, for other memory-intensive applications (such as *lu, streamcluster* and *canneal*), their 95-percentile prediction errors can be more than 90% for the polynomial regression predictive models. In particular, for *canneal*, its 95-percentile prediction error for the SVR model can be as high as 178%. Such high prediction errors for the outliers may be the result of several causes. First, it can come from the limitations of

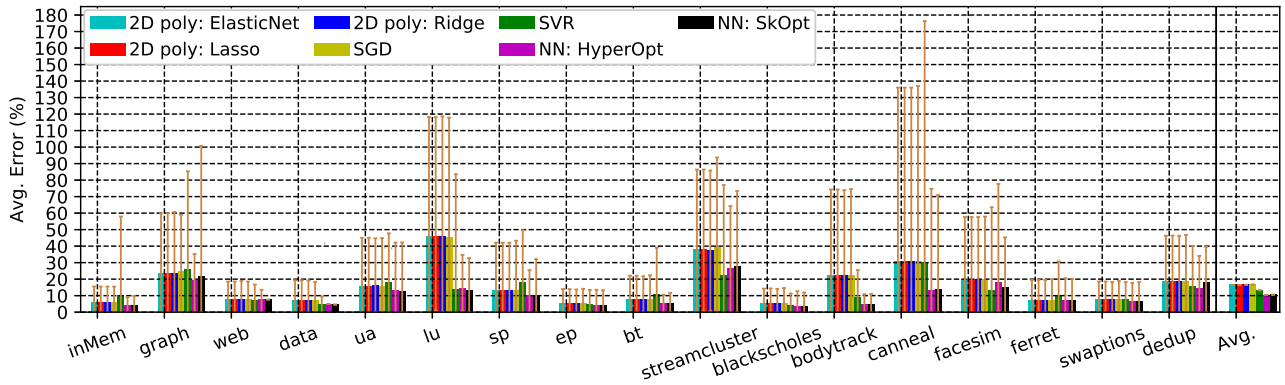
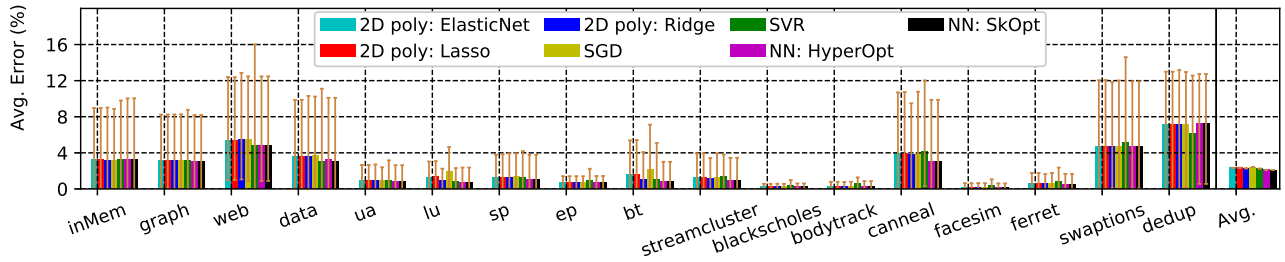
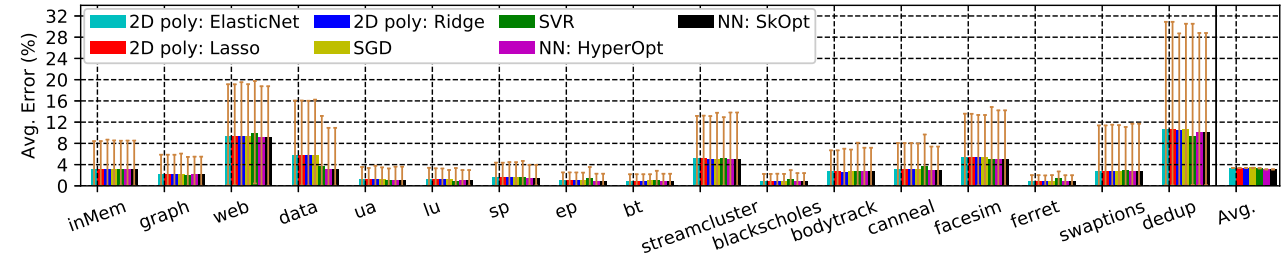


Fig. 3: Prediction errors of the predictive models in *uPredict* for the benchmark applications on the private cloud.



(a) Amazon EC2 Cloud.



(b) Google Cloud Engine.

Fig. 4: Prediction errors of the predictive models in *uPredict* for the benchmark applications on the public clouds.

the predictive models and the micro-benchmarks that can only profile a subset of factors that affect the performance of cloud applications. We suspect that the relatively-high errors were caused by the mismatch between the memory access patterns of the micro-benchmarks and the aforementioned benchmarks. They may also be caused by the fact that the memory micro-benchmark is mainly profiling off-chip memory contention, which does not include cache access patterns.

Moreover, a detailed analysis of the results shows that most of the large errors were from predicted points when the background VMs and applications change. During these changes, the micro-benchmark can profile the resource contentiousness of the startup or shutdown of background VMs. The benchmarks, however, were later executed along with iBench applications, which have different resource contentiousness. Consequently, the predicted results using the profiled resource contention during VMs startups and shutdowns were relatively inaccurate. Third, benchmark applications can have different sensitivities to the contention of various underlying hardware resources, which make some to be more difficult to predict their performance accurately than the others.

D. *Evaluation of* uPredict *in Public Clouds*

Figure 4 shows the average and 95-percentile prediction errors of the studied predictive models for the benchmarks on Amazon AWS and Google GCE, respectively. Here, Figure 4a gives the results on AWS, where all predictive models have the average prediction errors less than 7% and 95-percentile less than 14% for all benchmarks. Moreover, the average prediction errors of all the predictive models for all the benchmarks differ by at most 3%, which suggests that the predictive models have almost equivalent accuracy on AWS. The lower prediction errors in AWS than the private cloud are mainly due to the relatively low resource contention. The execution times of the benchmarks on AWS can fluctuate up to 25%, compared to up to 10 times fluctuation in the private cloud.

These results show that the proposed profiler-based framework with any considered predictive model can predict most

balancers	turnaround time (s)	execution time (s)
dummy-alternate	3563	312
queue-based	987	276
uPredict-based	1066	256

(a) High-load;

balancers	turnaround time (s)	execution time (s)
dummy-alternate	360	339
queue-based	318	300
uPredict-based	287	242

(b) Low-load;

TABLE I: The average turnaround and execution times (seconds) of *Graphic Analytic* under different load-balancers.

single-VM benchmarks quite accurately on AWS. Hence, *uPredict* is feasible for ordinary AWS users to obtain accurate performance prediction without the need for the exact knowledge or control of the underlying execution environments. Although the NN-based models can provide slightly higher prediction accuracy, it takes much more time to train compared to that of polynomial regression-based models, especially with the hyperparameter optimizations.

Figure 4b further shows the results on GCE. Here, the average and 95-percentile prediction errors from all predictive models for all benchmark applications are less than 11% and 32%. In fact, except for *dedup*, the 95-percentile prediction errors for all other benchmarks are less than 20%. The overall average prediction errors for all the benchmarks are less than 4% for all the studied predictive models. These results show that the proposed predictive framework is highly accurate on GCE for the considered benchmark applications as well. Combining with the findings from AWS, these results further confirm that it is feasible for ordinary cloud users to utilize uPredict to get accurate performance prediction on public clouds without the need for the exact knowledge or control of the underlying execution environments.

Moreover, similar to the results on AWS, the average prediction errors from all predictive models for the benchmarks differ by at most 2%, suggesting that these predictive models have almost equivalent performance in terms of prediction accuracy on GCE. Also, the same as in AWS results, the figure shows that the polynomial regression and SVR models are comparable to neural network models in terms of prediction accuracy, although the NN models perform slightly better.

E. Case Study: Load-Balancing with uPredict

To illustrate the usage of *uPredict*, we have conducted a case study of load-balancing for two cloud servers. Here, each cloud server is a VM with the same configuration as the one in previous experiments (i.e., 16 VCPUs and 64 GB memory) and both run under OpenStack on two separate host machines. Each machine has two Intel Xeon E5-2630 processors (for a total of 16 cores) and 128GB memory. In addition to the VM acting as one cloud server, up to three (3) background VMs may be created randomly to run applications from iBench on each host machine, which changes at different fixed intervals.

Three different user-level load-balancing schemes were investigated in this study, which directs the requests of cloud users to run the benchmark *Graphics Analytic* of CloudSuite to one of the two cloud servers at runtime. First, the *dummy* load-balancer just distributes the received user requests *alternatively* to the two cloud servers. Second, a simple *queue-based* load-balancer considers the number of requests in the waiting queues of both cloud servers and distributes a new user request to the server with fewer requests. Finally, the *uPredict-based* load-balancer considers the predicted execution times for the requests in the waiting queues based on the current profiled resource contention from the micro-benchmarks on both cloud servers. A new user request would be distributed to the server where the request is expected to complete earlier.

In the first 3-day experiment, user requests were periodically sent to the load-balancer starting with 12 requests per hour. The rate gradually increases to 24 and then decreases to 12 again in the end with changes in every 4 hours. With the resource contention from the background VMs/applications, the execution times for *Graphics Analytic* vary from 45 seconds to about 7 minutes. The request rate (especially at 24 requests per hour) is rather high and denoted as *high-load*.

Table I(a) shows the average turnaround and execution times for the generated requests of running *Graphic Analytic* under different load-balancers. Clearly, without considering resource contention and workload (i.e., queue length) on the cloud servers, the dummy-alternate scheme can result in a very high turnaround time, which is more than 3 times those for the queue-based and uPredict-based schemes. Although the queue-based scheme does not consider the resource contention, the queue length actually implies the delivered performance on each server. Hence, the average turnaround times for the queue-based and uPredict-based schemes are quite close.

For the average execution time of the requests, by avoiding the cloud server with high resource contention (implied by its queue length), the queue-based scheme can improve it for about 11.5% over dummy-alternate. Given that the uPredict-based scheme tries to execute most requests on the server with less resource contention, its average execution time can be improved by 18% compared to that of dummy-alternate.

In the second 3-day experiment, we reduced the request rate by half through the duration (denoted as *low-load*), and the results are shown in Table I(b). In this case, as the waiting queues on both cloud servers are empty for most of the time, the queue-based scheme performs relatively worse, where both of its average turnaround and execution times of the requests are around 12% better than those of the dummy-alternate. However, by exploiting the resource contention on the cloud servers, the uPredict-based scheme can further improve 19% and 10% over the queue-based scheme for the average execution and turnaround times of the requests, respectively. Note that, the profiling overheads of running the micro-benchmarks in the uPredict-based scheme were already included in the resulting turnaround times of the generated requests.

VI. Discussions and Future Work

Data-Input: The goal of this research is to investigate the feasibility of predicting the cloud running application's per-

formance under resource contention from the ordinary cloud users' perspective. Hence, we intentionally used the same data inputs for the considered benchmarks to eliminate the impact from input variations, so that the main cause of performance fluctuation is resource contention. The evaluation can thus focus on the accuracy of the proposed methodology to predict the impacts of resource contention only. In our future work, we will extend the work to include predictions under input data variations.

Long Running Applications: The evaluation results show that the predictive models do not work well for long execution times (outliers) that were not seen in the training data. We plan to incorporate extreme value theory with Neural Network to improve the prediction accuracy for these unusual cases [34]. Moreover, for long-running applications, they are more likely to experience changes in resource contention during their executions. We will consider adaptive predictive models by exploiting periodic profiling techniques in our future work.

VII. Conclusions

In this paper, we proposed *uPredict*, a user-level profiler-based performance predictive framework for single-VM applications running in multi-tenant clouds. First, *uPredict* adopts three specially devised micro-benchmarks to assess the contention of CPUs, memory and disks, respectively, in a VM. Then, predictive models based on regression and neural network (NN) techniques are developed. uPredict and the considered predictive models have been evaluated extensively. The results show that, even on the private cloud that has a quite high resource contention, the average prediction errors are between 10.4% to 17% for different predictive models. For public clouds that normally have much less contention, the average prediction errors of the considered benchmarks are below 4%. A use case of *uPredict* in load-balancing shows that the execution and turnaround times of the considered application can be reduced by up to 19% and 10%, respectively, compared to the simple queue-based load-balancer.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Agarwal, S. Kandula, N. Bruno, M. Wu, I. Stoica, and J. Zhou. Reoptimizing Data Parallel Computing. In *USENIX NSDI*, 2012.

[3] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *USENIX NSDI*, pages 469–482, 2017.

[4] Amazon Web Services. https://aws.amazon.com/ec2/.

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

[6] D. H. Bailey. Nas parallel benchmarks. In *Encyclopedia of Parallel Comput.*, pages 1254–1259. Springer, 2011.

[7] R. Begam, H. Moradi, W. Wang, and D. Zhu. Flexible vm provisioning for time-sensitive applications with multiple execution options. In *CLOUD*, 2018.

[8] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-parameter Optimization. In *NIPS*, pages 2546–2554, 2011.

[9] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[10] C. Delimitrou and C. Kozyrakis. iBench: Quantifying interference for datacenter applications. In *IISWC*, 2013.

[11] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS*, 2013.

[12] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*, 2014.

[13] M. Ferdman et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *ASPLOS*, 2012.

[14] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *EuroSys*, 2012.

[15] R. D. Friese, N. R. Tallent, A. Vishnu, D. J. Kerbyson, and A. Hoisie. Generating Performance Models for Irregular Applications. In *IPDPS*, 2017.

[16] Google Compute Engine. https://cloud.google.com/.

[17] S. R. Gunn et al. Support vector machines for classification and regression. *University of Southampton technical report*, 14(1):5–16, 1998.

[18] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks for Perception*, pages 65–93. Elsevier, 1992.

[19] A. E. Hoerl and R. W. Kennard. Ridge regression: applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

[20] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[21] HyperOpt. https://github.com/hyperopt/hyperopt.

[22] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *TPDS*, 22(6):931–945, 2011.

[23] P. Leitner and J. Cito. Patterns in the Chaos&Mdash;A Study of Performance Variation and Predictability in Public IaaS Clouds. *TOIT*, 16(3), 2016.

[24] LMbench. http://www.bitmover.com/lmbench/.

[25] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and Deadline-constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. In *HPCC*, 2012.

[26] M. Mao and M. Humphrey. Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *HPCC*, 2011.

[27] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *MICRO*, 2011.

[28] N. Mishra, J. D. Lafferty, and H. Hoffmann. ESP: A Machine Learning Approach to Predicting Application Interference. In *ICAC*, 2017.

[29] H. Moradi, W. Wang, A. Fernandez, and D. Zhu. upredict: A user-level profiler-based predictive framework for single vm applications in multi-tenant clouds, arXiv:1908.04491, 2019.

[30] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *USENIX ATC*, 2013.

[31] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In *Cloud Computing*, 2010.

[32] RightScale. State of the Cloud Report, 2018.

[33] F. Romero and C. Delimitrou. Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems. In *PACT*, 2018.

[34] E. M. Rudd, L. P Jain, W. J. Scheirer, and T. E. Boult. The Extreme Value Machine. *TPAMI*, 40(3):762–768, 2018.

[35] W. S. Sarle. Neural network faq. ftp://ftp.sas.com/pub/neural/FAQ.html.

[36] J. Scheuner and P. Leitner. Estimating Cloud Application Performance Based on Micro-Benchmark Profiling. In *CLOUD*, 2018.

[37] Scikit-learn. http://scikit-learn.org/stable/.

[38] Scikit-Optimize. https://github.com/scikit-optimize.

[39] A. J. Smola and B. Schölkopf. On a Kernel-Based Method for Pattern Recognition, Regression, Approximation, and Operator Inversion. *Algorithmica*, 22(1):211–231, Sep 1998.

[40] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Adv. in NIPS.*, pages 2951–2959. 2012.

[41] E. D. Sontag. Feedback Stabilization using Two-hidden-layer Nets. *IEEE Trans. on Neural Networks*, 3(6):981–990, Nov 1992.

[42] D. F. Specht. A General Regression Neural Network. *IEEE Trans. on Neural Networks*, 2(6):568–576, 1991.

[43] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa. ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computers. In *ASPLOS*, 2013.

[44] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[45] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *USENIX NSDI*, 2016.

[46] N.J. Yadwadkar, B. Hariharan, J.E. Gonzalez, B. Smith, and R.H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *SoCC*, 2017.

[47] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *ISCA*, 2013.

[48] G. P. Zhang, B. E. Patuwo, and M. Y. Hu. A Simulation Study of Artificial Neural Networks for Nonlinear Time-series Forecasting. *Computers & Operations Research*, 28(4):381–396, 2001.

[49] T. Zhang. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In *ICML*, 2004.

[50] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *J. of the Royal Stat. Soc.: Series B (Stat. Meth.)*, 67(2):301–320, 2005.