Online Performance Modeling and Prediction for Single-VM Applications in Multi-Tenant Clouds

Hamidreza Moradi, Wei Wang, *Member, IEEE* and Dakai Zhu, *Senior Member, IEEE*

Abstract—Clouds have been adopted widely by many organizations for their supports of flexible resource demands and low cost, which is normally achieved through sharing the underlying hardware among multiple cloud tenants. However, such sharing with the changes in resource contentions in virtual machines (VMs) can result in large variations for the performance of cloud applications, which makes it difficult for ordinary cloud users to estimate the run-time performance of their applications. In this paper, we propose online learning methodologies for performance modeling and prediction of applications that run repetitively on multi-tenant clouds (such as on-line data analytic tasks). Here, a few micro-benchmarks are utilized to probe the in-situ perceivable performance of CPU, memory and I/O components of the target VM. Then, based on such profiling information and in-place measured application's performance, the predictive models can be derived with either Regression or Neural-Network techniques. In particular, to address the changes in the intensity of resource contentions of a VM over time and its effects on the target application, we proposed *periodic model retraining* where the sliding-window technique was exploited to control the frequency and historical data used for model retraining. Moreover, a *progressive modeling* approach has been devised where the Regression and Neural-Network models are gradually updated for better adaptation to recent changes in resource contention. With 17 representative applications from PARSEC, Nas Parallel and CloudSuite benchmarks being considered, we have extensively evaluated the proposed online schemes for the prediction accuracy of the resulting models and associated overheads on both a private and public clouds. The evaluation results show that, even on the private cloud with high and radically changed resource contention, the average prediction errors of the considered models can be less than 20% with periodic retraining. The prediction errors generally decrease with higher retraining frequencies and more historical data points but incurring higher run-time overheads. Furthermore, with the neural-network progressive models, the average prediction errors can be reduced by about 7% with much reduced run-time overheads (up to $265X$) on the private cloud. For public clouds with less resource contentions, the average prediction errors can be less than 4% for the considered models with our proposed online schemes.

Index Terms—Performance Modeling and Prediction; Periodic re-training; Progressive learning; Profiling; Multi-Tenant Clouds;

1 Introduction

Due to the low cost-of-ownership, public clouds have been increasingly adopted by many organizations to serve as their main computing infrastructures. However, this low cost is generally achieved through sharing the hardware resources by multiple virtual machines (VMs) on the same host with *multi-tenancy* of different users in the clouds. Such sharing of hardware can lead to resource contention, which in turn will negatively affect the performance of the applications running in the VMs [1]. The impacts of such contention on the performance can be application specific depending on their resource requirements. Moreover, as the number of VMs and their running applications on the same host change over time, the impact and severity of such contention can vary significantly, causing the performance of the same application fluctuates in a quite large range at run-time.

Such performance variations have made it very challenging to predict the performance of an application running in the cloud environment. However, to design efficient auto-scaling policies and select the correct type and number of VM instances, it is critical for cloud users to obtain the accurate knowledge on the performance of given applications to meet their performance and cost objectives. There have been many studies on the prediction of the application's performance under hardware resource contention [2, 3, 4, 5]. However, most of these studies have focused on system level techniques that normally require extensive knowledge of the applications (including their resource demands and detailed memory behaviors) that share the hardware resources [6, 7, 8]. Although some studies on predictive techniques do not require prior knowledge of the co-running applications (such as Bubble-flux [9] and ESP [10]), they would require access to low level hardware performance monitoring units (PMU).

In public clouds, ordinary users typically do not have control over which groups of applications and their associated VMs will share the hardware, nor do they have a-prior knowledge of the behaviors of these applications. In addition, such users generally do not have access to special hardware registers or counters about the underlying host machines in the cloud environment. Therefore, it is imperative to design and develop a user-level prediction framework for ordinary cloud users.

To estimate the resource contention of key components (such as CPU, memory and storage) in clouds, profiling techniques with micro-benchmarks have been widely deployed [11, 12]. For instance, Leitner et al. studied the usage of 23 micro-benchmarks to model the performance of two applications running on different cloud instances [13]. Moreover, Baughman et al. used actual application executions with different input data sizes to model the application's performance [14]. However,

• *H. Moradi, W. Wang and D. Zhu are with the Department of Computer Science, The University of Texas at San Antonio. Emails:* {hamidreza.moradi, wei.wang, dakai.zhu}@utsa.edu

Manuscript received XXX. 2020; revised XXXX 2020

both studies have focused on profiling the average performance of a particular hardware resource rather than considering the run-time resource contention severity. Note that, to enable efficient resource management and job scheduling, it may be necessary to predict the performance of a cloud application at the run-time by considering the in-situ severity of resource contention.

Based on the profiling technique with micro-benchmarks, we have studied *uPredict*, a user-level performance predictive framework for cloud applications, in our previous work [15]. With a set of specially devised micro-benchmarks that run right before an application was executed, the resource contention in CPU, memory and disk I/O was accessed respectively. Then, uPredict exploits such contention information with the in-place measured performance of a give application to build *offline* application-specific regression or neural-network based predictive models. The models essentially reflect the sensitivity of the application's performance to the levels of encountered resource contention and thus can be used to predict its performance based on the in-situ profiling information from the micro-benchmarks. The evaluation results verified the feasibility of uPredict with reasonable prediction errors in both a private cloud with potentially high resource contention and two public clouds [15].

Note that, the predictive models in uPredict were trained offline and rather static, which may not be able to model the rapidly changing resource contention in cloud environment. To address this problem, we propose online performance modeling and prediction schemes in this paper, where some preliminary results have been reported in [16]. In particular, to take advantage of the recently collected profiling information and performance data, we studied an online approach with *periodic re-training* for model refinement. Here, the predictive models were re-trained periodically based on a user-defined *adaptation frequency*, which is specified as the *batch size* to indicate the number of new iterations before a model is re-trained at run-time. Moreover, based on the concept of sliding-window, the number of historical data points to be utilized to re-train a model is determined by a *window-size*, which denotes the number of batches [16].

For neural-network based predictive models, simple (shallow) models are not capable of accurately modeling the high dimensional relation between the profiling information and application performance. However, complex (deep) models require a large number of training data points. Therefore, to address such a model complexity issue, we proposed another online approach based on a recently studied *progressive model* [17]. Here, the model complexity increases over time by adding more layers to the neural-network with the ability of knowledge transfer [18]. That is, instead of learning the model parameters from the scratch as in periodic re-training, the progressive modeling approach can gradually update the existing neural-network model's parameters and architecture with recently collected profiling and performance data. This gives progressive model training a better chance to model the application performance and respond faster and more accurately to changes in the intensity of resource contention without losing the knowledge learned from past data points.

The proposed online schemes have been evaluated extensively with 17 representative applications from PARSEC, Nas Parallel Benchmark (NPB) and CloudSuite [19, 20, 21] running on both a private cloud and two public clouds, including Amazon Web Services (AWS) [22] and Google Compute Engine (GCE) [23]. The private cloud was managed with OpenStack on a local cluster with intentionally introduced high levels of resource contention caused by co-located VMs running either CPU or I/O intensive applications.

For the private cloud with high resource contention, the evaluation results show that, with the periodic re-training scheme, smaller batch sizes (i.e., higher adaptation frequencies) and larger window sizes (i.e., more historical data points) can generally lead to more accurate predictive models with reduced prediction errors. However, higher run-time overheads can be introduced with smaller batch sizes and large window sizes. For instance, with the batch size of 1 and window size of ∞ (i.e., the model is re-trained after each iteration with all historic data points being considered), the resulting Lasso regression based predictive models have the average prediction error of 17% with overall run-time overhead of 1325 seconds. When the batch size is 50 and the window size is 5, the average prediction error is about 19.58% with the overall run-time overhead of 10.8 seconds (a reduction of $120X$). When the online progressive modeling approach is adopted with the batch size of 50, the resulting neural-network models have the average prediction error of 16%, which is a significant improvement over the 35% error of the Lasso models with the periodic re-training approach. For public clouds with less resource contention, the average prediction error is less than 4% for the resulting progressive neural-network models, which indicates that the online schemes can accurately predict the performance of the considered applications in public clouds.

The main contributions of this work can be summarized as follows:

- A periodic re-training approach for the online performance modeling and prediction is proposed, where the sliding-window technique is utilized to control the frequency and historical data points for re-training [16];
- An online progressive modeling approach has been proposed for neural-network based predictive models to better incorporate the recent changes of resource contention in cloud environment without losing past knowledge;
- The proposed online schemes were evaluated extensively with representative benchmark applications on both a private and two public clouds. The trade-offs between the achieved prediction accuracy of the online schemes and the associated overhead were thoroughly evaluated.

The remainder of this paper is organized as follows. Section 2 discusses the closely related works. Section 3 reviews the preliminary concepts and presents the overview of the online framework. Section 4 introduces the online periodic model re-training and adaptation techniques. The progressive modeling approach is presented in Section 5. The evaluation results are presented and discussed in Section 6. Section 7 discusses the limitation of this work and points out our future work. Finally, Section 8 concludes the paper.

2 Closely Related Works

Many research studies have been investigated to increase the quality of resource provisioning in clouds by predicting and leveraging the incoming requests received by the cloud providers [24, 25, 26, 27]. In [28], Huang et. al. used Recurrent Neural Networks (RNN) with LSTM cells to predict the load and performance of the cloud servers based on the number of requests received. Here, dedicated systems serving only a single type of application to multiple clients were considered. Most of these research studies have focused on the number of requests received and different workload resource requirements without considering contention due to collocation.

There are also many research studies that exploit resource profiling for performance analysis of applications in a controlled environments (e.g., on dedicated servers) [3, 10, 29, 30]. Caunta estimates the performance degradation due to collocation using resource profiling [29]. QuMan uses profiling to estimate the required resource of an application and thus makes collocation decisions [30]. ESP and DeepDive use hardware counters to capture the source of interference and performance bottleneck due to collocation [3, 10]. Despite these studies, the performance degradation due to contention in the cloud environment is not preventable and still exist. This comes from the fact that the information of applications may not be known before they are deployed in clouds. Moreover, migrating a VM can be expensive and will cause network contention after the deployment. Such migration can also lose the access to the VM which jeopardises the Service Level Agreement (SLA) objectives.

Some research studies have mainly focused on modeling and prediction of a specific type of applications (e.g., Spark) due to the existence of additional information about job states and execution [31, 32, 33, 34]. For instance, Fei Zu et. al. used existing Directed Acyclic Graph (DAG) information of a Spark job about the task to predict the average performance of the big data application in different instances. The predicted performance was used to decide the number of spot instances to use for execution of the spark jobs with the objective of reducing the cost [31]. This category of work only predicted the average performance, and would require DAG information which is not applicable to all applications.

Performance evaluation and modeling of applications running on the cloud has been the topic of many research studies [35, 36, 37, 38, 39]. Mao et. al. [35] considered customizable workloads on the cloud to profile the performance of different instances and build a performance-cost model of cloud providers. In [37], the authors presented Graphalytics, a graph analytics benchmark for GPU-based cloud services, with the focus on modeling the performance regarding variable input sizes. TeaStore [36] uses micro-service based profiling on the cloud to model and evaluate the effect of parametric changes of a workload on achieved performance. In [38], instructions are injected into an application's source code to benchmark the progress of different execution stages, which are later used for predicting the overall application performance. To assist users in selecting the best instance satisfying their performance needs, the authors exploited user-provided weights and profiled performance of different system resources to provide a ranked list of VM configurations [39]. To predict the performance of applications in cloud environments, PARIS has been designed as a predictive model that exploits resource profiling information obtained from the OS on different public cloud services [40]. Similarly, Scheuner and Leitner used micro-benchmark profiles to predict the application's performance on various VMs in different clouds [41]. However, these studies can only model and predict an application's average performance on various clouds. In particular, they cannot be utilized to predict the *in-situ* performance of an application running on clouds. Such runtime performance information can be important for cloud users to make proper decisions on when to trigger auto-scaling operations [42] and for time-sensitive applications to satisfy their timeliness requirements [43, 44].

3 Overview and Preliminary

In this work, we focus on applications (such as data and graph analytics that deployed in online machine learning applications [7, 45]) that repeatedly run on a single VM at the request of users in the cloud environment. To focusing on evaluating the accuracy of modeling the experienced contention and corresponding application performance, we assume that the applications' execution times are affected only by the resource contention (i.e., interference) caused by the collocated VMs on the same host machine and do not consider varying input data sizes. That is, the data to be processed in each iteration is assumed to have a fixed size. However, we would like to point out that the predictive model considered in this work can be easily extended to incorporate different data sizes, especially when such a size has a known (e.g., linear) relation with the application's execution time. For these cases, two models could be trained, where one input-performance model focuses on the application execution time regarding to different input sizes, and another contention-performance model for different contention levels. Using the contention-performance model, the performance for the fixed input size in different contention levels can be predicted and scaled to get the execution time for the targeted input data size using the input-performance model.

As in our previous work [15], we assume that ordinary cloud users do not have access to performance counters in the hypervisor and special registers in the underlying hardware of the host machine. That is, the proposed online schemes interacts only with the target VM in which the user's application will run. In what follows, we first present the overview and workflow for the proposed online predictive framework. The micro-benchmarks for assessing resource contention in CPU, memory and I/O are reviewed next.

3.1 Overview of Online Methodology

The overview of the proposed online framework for performance modeling and prediction is shown in Fig. 1. Here, as the first step, a few specially designed micro-benchmarks [15] (see Section 3.2 for details) will run in the target VM to assess the current level of resource contention in CPU, memory and I/O components caused by other applications running in VMs

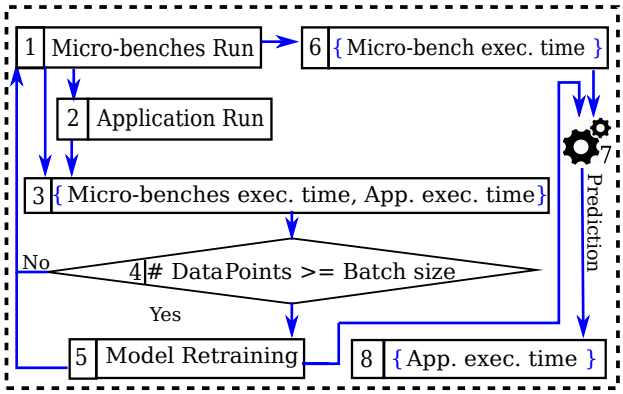


Fig. 1: Overview and workflow of the online framework.

collocated on the same host machine. Then, the application will run in the target VM in step 2. The measured performance (execution time) of the application will be correlated with the obtained profiling information from the micro-benchmarks in step 1 to form a data tuple. Due to changes in collocated VMs and/or their applications, the interference experienced by the application may change during its execution. It is possible to exploit interval based profiling for long-running applications. However, exploring such changes is beyond the scope of this paper and will be left for our future work.

Once the first set of data tuples are collected, the initial performance model can be trained using machine learning techniques, such as Regression or Neural Networks (step 5; see Sections 4.1 and 5.2). Then, with every new set of data collected, the existing performance predictive models for the application can be retrained or updated at run-time. As shown in the figure, we use the *batch size* to control how frequently a predictive model is re-trained (see Section 4.2). Once the model is obtained, based on the profiling information from running the micro-benchmarks at the beginning of each iteration, the application's performance can be predicted at run-time (step 7).

3.2 Resource-Contention and Micro-benchmarks

In [15], we have devised a set of micro-benchmarks to estimate the contention levels of key resource components (i.e., CPUs, memory and disks) in a VM due to interference from the collocated VMs and their applications on the same host machine. From our previous study [15], we found that the prolonged (shortened) execution of a micro-benchmark can be a indicator of increased (decreased) contention level for the corresponding resource, respectively. That is, the profiling information from the micro-benchmarks provides an estimation on the perceivable performance of these components delivered to a user application at run-time. Given that we focus on applications running only on a single VM, we do not consider network issues in this work, which will be studied in our future work when applications running on multiple VMs are considered. In what follows, we explain the detailed design for each micro-benchmark [15].

CPUs: For resource contention in CPUs, a multi-threaded micro-benchmark is utilized to stress the performance of the virtual CPUs of a given VM [15]. Here, each thread will loop through and decrements an *in-register counter* that is initiated with a given value. These in-register operations ensure that the performance of this micro-benchmark is not affected by memory at run-time and thus examine the contention in CPUs to the maximum extent. The amount of time for each thread to reach zero for the in-register counter is recorded. The number of created threads at run-time for this micro-benchmark will be equal to the number of virtual CPUs of the target VM. In the end, the averaged execution time (t_{cpu}) from all threads will be used as the indicator of the overall contention level for all the virtual CPUs in the target VM.

Memory: Similarly, to stress the memory bandwidth of the target VM, the memory micro-benchmark will access a 2GB array with the stride size of 128 sequentially [15]. The objective of such a memory access pattern is to ensure that each data access needs to access off-core memory rather than the on-chip caches. Again, the number of threads in this micro-benchmark is the same as the number of virtual CPUs in the VM and each of them accesses the equal portion of the array. The execution time of this micro-benchmark will provide us the insight into the performance impact of the memory contention experienced by the target VM in the system.

Disk I/Os: For the I/O performance of the target VM, we designed the disk micro-benchmark that reads 256MB data from the disk with the page size of 4KB [15]. During the execution of this micro-benchmark, the OS file caching will be disabled to prevent the data file being cached by the OS. The micro-benchmark adopts four threads, which will introduce enough I/O operations to stress the disk's bandwidth while avoiding too much inter-thread communication. Again, the execution time for this micro-benchmark to access the required amount of data in the file will be utilized to indicate the contention level of the disk.

3.3 Profiling and Performance Data Tuples

In each iteration, these micro-benchmarks will be invoked sequentially first, followed by the execution of the user's application. We assume that the changes in the contention level of the resources measured by the micro-benchmarks will affect the execution of the application and be reflected by the corresponding changes in its execution time. Here, the profiling information refers to the measured execution times of the micro-benchmarks. Together with the measured execution time of the application in each iteration, they form a data tuple $\{t_{CPU}, t_{mem}, t_{disk}, t_{app}\}$, which represents the implicit relationship between the application's performance and resource contention of the VM.

For the data tuples, the execution times of a cloud application can be obtained directly after its executions without incurring additional overhead. Hence, in each iteration, the profiling overhead only includes the extra time to execute the micro-benchmarks. Although it is desirable to have smaller profiling overhead, short executions of the micro-benchmarks may not be able to accurately capture the actual contention of the resources. For the experiments conducted in this paper, each micro-benchmark was executed for about 3 seconds

in each iteration to ensure the proper resource contention was obtained. In [15], we have shown that the execution of each micro-benchmark can be shortened to half second while obtaining almost the same prediction accuracy for the considered predictive models being deployed offline.

3.4 Predictive Models and Performance Prediction

Once enough profiling and performance data tuples of an application are collected, various performance predictive models can be developed and trained based on different machine learning techniques. For example, several offline regression and neural-network based predictive models have been studied in our previous work [15]. Here, the regression based models can effectively present simple relationship between an application's performance and the profiling information, which are generally fast to train with high prediction accuracy. However, for applications with complex behaviors and non-linear relationships, neural-network based models would be needed to get better prediction results, which is usually achieved through time-consuming hyper-parameter optimizations [15].

When an application-specific predictive model was obtained, it can be utilized to predict the performance (e.g., execution time) of the application based on the profiling information from the micro-benchmarks. It has been shown that both regression and neural-network based predictive models can quite accurately predict the performance of the considered applications in both a private and public clouds [15].

Note that, for a given target application and its VM, the collocated VMs on the same host and their applications may keep changing constantly. Researches show that 90% of VMs deployed on the cloud live less than a day [46]. Such frequent changes in the collocated VMs and their applications can introduce new resource contention patterns, which may not be captured by the offline trained predictive models. This in turn can lead to large prediction errors. In this work, to incorporate such changing new patterns of resource contention and adjust the predictive models, we propose two online model adaption approaches: *periodic model retraining* and *progressive modeling*, which are detailed in the next two sections, respectively.

4 Periodic Model Retraining

In order to incorporate the new resource contention patterns, the straightforward approach is to *re-train* the predictive models *periodically* utilizing the profiling information from the most recent executions of an application. For the approach of periodic model retraining, there are two major issues that have to be carefully addressed. The first issue is the *retraining frequency*, that is, *how often* and *when* the model should be retrained. The second one is related to the number of historical data tuples should be exploited to retrain the model. Intuitively, more historical data tuples with higher retraining frequency can improve the obtained predictive model on its accuracy, which in turn can introduce higher run-time overheads.

In this work, we consider a *sliding-window* based approach to control the retraining frequency and required historical data tuples for retraining. Moreover, considering the excessive overheads to train the neural-network based predictive models due to its time-consuming hyper-parameter optimization process [15], we focus on regression-based predictive models for the periodic model retraining approach.

4.1 Regression-based Predictive Models

When the relationship between the execution times of a target application and the levels of resource contention profiled by the micro-benchmarks is relatively simple, the predictive models can be derived (or learned) with regression techniques on a set of data tuples. In general, regression models can provide high accuracy for simple relationships, and they are fast to train and not very sensitive to hyper parameters [47]. Based on the profiled data tuples, we have conducted extensive experiments with different degrees of polynomial regression and found that, the linear regression and 3-degree polynomial regression can result in either under-fitting or over-fitting problems, respectively, and the 2-degree polynomial regression fit the relationship for the considered applications the best, which will be considered in this work.

$$\begin{aligned} t_{app} =& f(t_{CPU}, t_{mem}, t_{disk}) \\ =& \alpha_1 \cdot t_{CPU}^2 + \alpha_2 \cdot t_{mem}^2 + \alpha_3 \cdot t_{disk}^2 + \\ & \alpha_4 \cdot t_{CPU} \cdot t_{mem} + \alpha_5 \cdot t_{CPU} \cdot t_{disk} + \alpha_6 \cdot t_{mem} \cdot t_{disk} \\ & + \alpha_7 \cdot t_{cpu} + \alpha_8 \cdot t_{mem} + \alpha_9 \cdot t_{disk} + \alpha_{10} \end{aligned} \tag{1}$$

For the 2-degree polynomial regression models, the relationship between the execution times of an application and the profiled information of the micro-benchmarks can be represented as Equation (1). For a given set of obtained data tuples, there are many existing packages that can be exploited to find out the values of the coefficients in the above equation and train the corresponding predictive model between the application performance and the profiled resource contention, such as Lasso, Elastic Net and Ridge regression techniques [48, 49, 50]. We have evaluated the methodology with these different algorithms and similar results have been obtained [15]. In this work, for the periodic model retraining approach, we focus only on the Lasso regression algorithm and report the evaluation results on its prediction accuracy and model training overheads accordingly due to space limitation.

4.2 Sliding-Window based Periodic Retraining

Once the coefficients in Equation (1) is determined with the chosen regression techniques, the obtained predictive model can be utilized to predict the performance of the given application based on the profiling resource contention information from running the micro-benchmarks. Moreover, after each iteration, a new data tuple can be obtained from the measured execution time of the target application as well as the profiled information of running the micro-benchmarks. Theoretically, the predictive model could be retrained after each new data tuple is obtained. With the latest relationship between the profiled resource contention and the application's performance being incorporated, the updated model could provide more

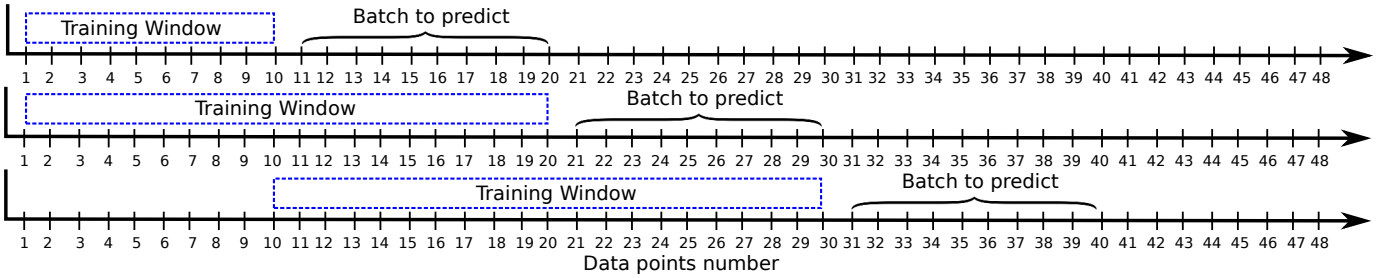


Fig. 2: Sliding-window based adaptations for the predictive model with batch size of 10 and window size of 2.

accurate prediction for the application's execution time in the next iteration. However, retraining after each iteration would introduce prohibitive run-time overheads as shown in our evaluations (see Section 6). Therefore, there is an interesting trade-off between retraining frequency and run-time overheads.

Retraining Frequency (Batch Size): In this work, to control the adaptation frequency and regulate how often (and when) the predictive model should be retrained at run-time, a *batch size* is utilized. It denotes the number of data tuples that should be accumulated in a batch before training the initial model or re-training/updating the existing model. Fig. 2 shows an example with the batch size of 10. That is, once an initial predictive model is obtained using the first batch of data (after the 10^{th} data point), it will be utilized to predict the performance of the target application for its execution in the next 10 iterations. In the meantime, 10 new data tuples will be generated from the profiling information of the micro-benchmarks and the measured performance of the target application. Before the 21^{st} iteration, the predictive model will be retrained by utilizing the recent data tuples in the training window.

Clearly, having smaller batch sizes will enforce the predictive model be retrained more frequently at run-time, which may improve the accuracy of the prediction results. On the other hand, it will in turn lead to higher run-time overheads. For the extreme case where the batch size equals 1, the predictive model will be retrained after every iteration of running the target application. The trade-offs between the prediction accuracy and run-time overheads have been extensively evaluated as reported in Section 6.

Window Size (Number of Historical Data Tuples): When it is the time to retrain the predictive model, we have to decide *which part* and *how many* historical data tuples should be exploited. Intuitively, the most recent data tuples should be utilized as they contain recent resource contention information that can help the predictive model to get better prediction results for the target application's execution in future iterations. Moreover, instead of utilizing all historical data tuples, which can lead to prohibitive run-time overhead as well as excessive memory space demand for model retraining, we adopt the *sliding-window* technique to control the number of historical data tuples to be utilized for retraining the model.

Specifically, a sliding-window contains a certain number of the most recent batches (which is denoted as *window size*). Only the data tuples in these batches will be utilized to retrain the predictive model. At the beginning of the execution, the first few sliding-windows may not have enough batches and contain fewer number of data tuples for training. For the example shown in Fig. 2, it has the window size of 2. Here, the first sliding-window has only one batch of 10 data tuples.

Once enough number of batches are accumulated, the predictive model will be retrained with a certain number of most recent data tuples that are determined by both batch and window sizes. For the example in Fig. 2, it has the batch size of 10 and window size of 2. Therefore, (up to) 20 most recent data tuples in the training window will be utilized to train/retrain the predictive model.

When the window size is set as ∞, this extreme case will reduce to where all historical data tuples are needed for retraining the predictive model. Moreover, when a given number (e.g., 100) of historical data tuples are desired to retrain the model, various combinations of batch and window sizes can be adopted (e.g., batch and window sizes of being 20 and 5 vs. 10 and 10). Apparently, these settings will affect the overall prediction accuracy and the runtime overheads (see Section 6 for the detailed evaluation results).

5 Progressive Modeling

For the periodic model retraining approach, one major limitation is the number of historical data tuples (i.e., window size) utilized for retraining the predictive models. Ideally, all historical data tuples should be utilized (with the window size being ∞) to incorporate all observed patterns of resource contention. However, in addition to incurring excessive run-time overheads, it may not be always feasible to keep all the historical data tuples [51, 52]. Therefore, instead of completely retraining the predictive models from scratch for each iteration, in this section, we consider progressive modeling approach that focuses on updating the existing predictive models utilizing the recently collected data tuples. The goal is to have the resulting predictive models to incorporate the recently observed patterns of resource contention while still keeping the knowledge of the old resource contention patterns. In particular, we consider two types of progressive modeling techniques: regression models with online adaptation and the progressive neural-network models.

5.1 Regression Models: Online Adaptation

In Section 4, the periodic retraining adopts the Lasso regression to take advantage of Singular Value Decomposition

(SVD) matrix factorization technique to find the coefficients in Equation (1) [53]. Although SVD can accurately estimate the parameters with relatively low overhead, the SVD based approach is lack of the ability to update the existing parameters with new performance data tuples. On the other hand, Stochastic Gradient Decent (SGD) algorithm has shown a great potential in recent years for model optimization and many existing packages use SGD for online model adaptation to update the coefficients [54]. Moreover, SGD has the ability to incorporate a large number of features and data tuples [55], which make it is very suitable for the objective of online performance modeling, especially considering the long running features of the cloud applications.

In this section, to adjust the coefficients in Equation (1) based on their current values (i.e., the existing predictive models) and the newly collected data tuples, we consider SGD algorithms for the online adaptation of regression models. Specifically, to adjust the coefficients of the regression models, we consider two different implementations of the online regression techniques using SGD algorithm in Scikit-learn package [56]. First, the default regression model with L2 regularization and online modification using SGD (denoted as *sgdReg*). Second, a slightly modified version in which the L2 regularization has been replaced with L1 regularization to implement an online version of the Lasso regression (denoted as *sgdRegLasso*). In Lasso regressions, L1 regularization will limit the size of the coefficients and yields a sparse model by eliminating coefficients with small values that can help better deal with high-dimensional data, where L2 regularization using Euclidean distance would be better in dealing with the scale of the features [57].

5.2 Progressive Neural Network Models

Neural-network based models have shown the ability to model any linear and non-linear behaviors efficiently [58, 59]. In our previous work [15], we observed that using a complex neural-network with many layers would require many training data points in the very beginning. This is not feasible for online learning as the training data points are gradually collected. Also, very shallow neural networks have poor performance and are not capable of learning higher dimensional correlations between the micro-benchmarks and application performance.

In [60], a Neural-Network (NN) based progressive modeling approach has been studied, which updates the existing model for its parameters instead of retraining all parameters from scratch. Such a progressive approach allows the knowledge in an existing trained model be transferred to a new model that is capable of extracting more insights from all collected data. Here, the model's hyper-parameters are updated gradually when more new data tuples are collected [17]. Following the same principles, we also consider a progressive neural network model in this work, where the number of layers can increase when more data tuples become available. This will help to model the higher dimensional correlation between the micro-benchmarks and application performance more accurately.

Knowledge Transfer and equivalent Identity Matrix: Specifically, in order to preserve the learned relationship between an application's execution times and the profiled resource contention from previous data tuples, we adopt a progressive neural network model that has the ability of knowledge transfer [18], where the knowledge from a previously trained shallow model can be transferred to a deeper one with more layers. Here, with the first batch of data tuples, a fully connected neural network with a single hidden layer of randomly initiated weights and biases will be trained as the predictive model to predict the executions of the application during the next batch. Equation 2 shows the micro-benchmarks progress as inputs (t_{CPU}, t_{mem}, and t_{disk}), first hidden layer weight ($w^1_{x,y}$) and bias ($b^1_{1,y}$) matrices, output layer weight ($w^{out}_{y,1}$) and bias ($b^{out}_{1,1}$) matrices, and the final output of the DNN (t_{app}), respectively. In the equation, x is the number of micro-benchmarks (3), and y represents the number of neurons in the first layer.

$$\left(\begin{bmatrix} t_{CPU} & t_{mem} & t_{disk} \end{bmatrix} * \begin{bmatrix} w^1_{x,y} \end{bmatrix} + \begin{bmatrix} b^1_{1,y} \end{bmatrix}\right) * \begin{bmatrix} w^{out}_{y,1} \end{bmatrix} + \begin{bmatrix} b^{out}_{1,1} \end{bmatrix} = t_{app} \tag{2}$$

Once the second batch of data tuples are collected, the single-layer neural network trained in the previous step will be replaced by a 2-layer neural network by transferring the weights and biases of the previous model to the new model with 2 layers. Fig. 3 illustrates the process of augmenting the neural network from one to two hidden layers while transferring the knowledge. Specifically, weights and bias in the blue boxes ($w^1_{x,y}$, $b^1_{1,y}$, $w^{out}_{y,1}$, and $b^{out}_{1,1}$ matrices above) will be transferred from the existing neural network model to the exact same position in the new neural network with two layers. Equation 3 shows the input, weight, and bias matrices for a Neural Network (NN) with two hidden layers. As discussed, the weight and bias matrices for the first hidden layer and output layer are transferred from NN with 1 layer.

$$\left\{\left(\begin{bmatrix} t_{CPU} & t_{mem} & t_{disk} \end{bmatrix} * \begin{bmatrix} w^1_{x,y} \end{bmatrix} + \begin{bmatrix} b^1_{1,y} \end{bmatrix}\right) * \begin{bmatrix} w^2_{y,y} \end{bmatrix} + \begin{bmatrix} b^2_{1,y} \end{bmatrix}\right\} * \begin{bmatrix} w^{out}_{y,1} \end{bmatrix} + \begin{bmatrix} b^{out}_{1,1} \end{bmatrix} = t_{app} \tag{3}$$

After a new layer is added to the neural network, if the new weights ($w^2_{y,y}$) and biases ($b^2_{1,y}$) are initialized to random values, the accuracy of the model could significantly degrade. To preserve the functionality of the model and prevent the decrease of prediction accuracy, instead of random initialization of the weights and biases for the new layer, the weights will be initialized to the equivalent identity matrix (I) and biases to zero [18]. In the identity matrix (I), the principal diagonal values are all ones. Considering i as the current number of layers, L^i as the weights matrix for the last layer of the existing model, and the I^{i+1} as the weights for the new layer, the functionality of the existing model can be preserved as there is $L^i I^{i+1} = L^i$. Note that, during the training process, all layers will be able to learn and can take any value as their parameters. The weights and biases in the green boxes in Fig. 3 shows the weights initialized to the corresponding identity matrix and biases initialized to 0.

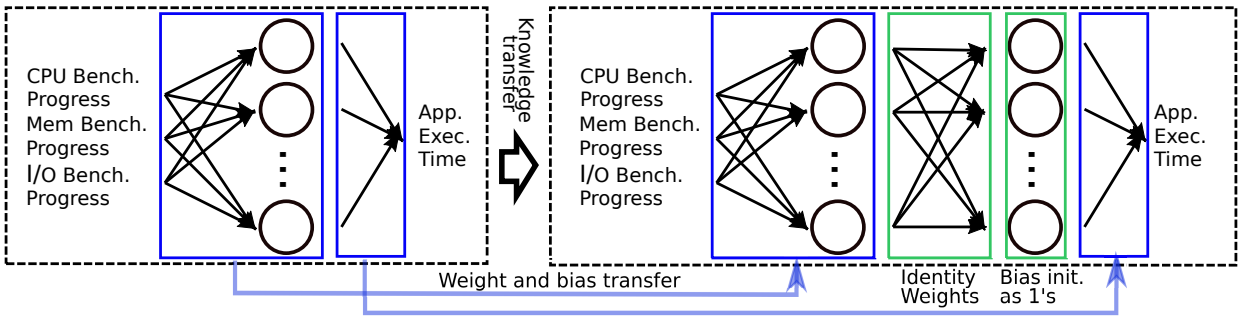


Fig. 3: Progressive Neural Networks - number of layers increases from 1 to 2 with knowledge transfer.

Moreover, the model will converge faster and achieves better accuracy compared to the weights that are initialized with random values since using the identity matrix will preserver the functionality [18].

Number of Hidden Layers: In the above process, to process each new batch of data tuples, a new layer would be added to the neural network, which can lead to rather deep and complex models with higher run-time overheads. In this work, to control the complexity of the resulting neural network models for them being utilized at run-time, the maximum of hidden layers is set as 5. That is, after adding five (5) hidden layers when processing the first five batches of data tuples, additional batch of data tuples will only be utilized to adjust the parameters of the five (5) hidden layers without further adding more layers. In our experiments, we have considered up to 10 layers and the results show that having five (5) hidden layers is enough to accurately model all the considered representative benchmarks.

6 Evaluations and Discussions

In what follows, we first present the evaluation methodology and experiment settings. Then, the evaluation results for periodic model retraining with Lasso regression models on a private cloud are discussed by considering different batch and window sizes. After that, the evaluation results for the progressive modeling approach on the private cloud are presented, which are compared against that of the periodic model retraining in terms of prediction accuracy and run-time overheads. Finally, the performance of the progressive modeling on two public clouds (i.e., Amazon AWS and Google GCE) are discussed.

6.1 Evaluation Methodology and Setups

We considered a total of 17 benchmarks from CloudSuite [21], NAS Parallel Benchmarks (NPB) [20] and PARSEC [19]. Specifically, four are from CloudSuite, including *In-Memory Analytic*, *Graph Analytic*, *Web Search* and *Data Serving*; five are from NPB, including *ua*, *lu*, *sp*, *ep* and *bt*, with the input class size of C; and eight from PARSEC, including *streamcluster*, *blackscholes*, *bodytrack*, *canneal*,*facesim*, *ferret*, *swaptions* and *dedup*. These benchmarks were chosen with the consideration to represent a wide range of applications with different resource utilization and the duration of experiments.

On the private cloud, these benchmark applications were compiled and run with the Ubuntu 16.04 environment on a virtual machine (VM) with 16 vCPUs and 16GB memory. The VM was created under the OpenStack installed on a server with dual Intel Xeon E5-2630 16-core processors and 128GB memory. Before the benchmark application run on the VM, each of the three designed micro-benchmarks runs for 3 seconds sequentially to probe the resource contention levels of the VM on the host machine. Then, the target benchmark applications were executed on the VM with 16 worker-threads.

To introduce interference and emulate resource contentions in the private cloud, varying number of background VMs have been created on the same host machine during the executions of the benchmark applications. Specifically, after each 2-hour interval, a random number (up to 7) is generated to indicate the number of background VMs should be created for the next interval of 2 hours. Moreover, each background VM will randomly choose either a CPU or memory intensive synthetic application from iBench suite [61] to introduce the different levels of interference for the key components of the VM that runs the target benchmark applications.

With the randomly introduced resource contention from the background VMs and their applications, the micro-benchmarks and the selected benchmark applications run repeatedly until 1,000 data tuples are collected for each benchmark. These data tuples are utilized to evaluate the periodic model retraining and progressive modeling for different regression models on their prediction accuracy and run-time overheads.

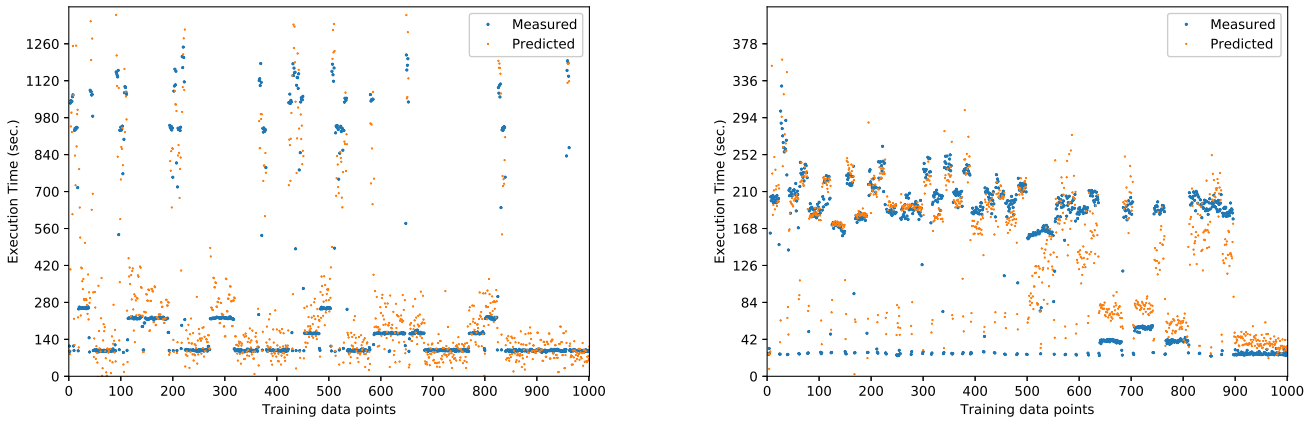
To quantify the accuracy of the predicted execution time of an application from a model, we define the *prediction error* for each data point as:

$$Pred^{err} = \frac{|time_{measured} - time_{predicted}|}{time_{measured}} \quad (4)$$

where $time_{predicted}$ is the predicted execution time of the application using a given predictive model with the online learning approaches based on the profiling data of the micro-benchmarks and $time_{measured}$ denotes the measured execution time of the application. We report the *average* prediction error of all the data points for each benchmark application.

6.2 Periodic Model Retraining on Private Cloud

Fig. 4 first shows the measured execution times (the blue points) for two representative benchmarks, *Streamcluster* and



a. Streamcluster b. Canneal

Fig. 4: Measured and predicted execution times for *Streamcluster* and *Canneal* with batch size of 10 and window size of ∞.

Cannel, respectively, on the private cloud. From the figure, we can see that their performance does change radically at run-time due to varying levels of resource contention on the host machine, where their execution times can vary up to 10 times. Therefore, to support cloud users for proper planning of their operations, it is crucial to get reasonably accurate performance prediction for the execution of their applications. Note that, for *Cannel*, the execution times for some iterations can be as low as around 30 seconds. This comes from the fact that, at the beginning of each 2-hour interval when the background VMs change, we stop the executions of the interfering applications for the first 5 minutes.

The figure also shows the predicted execution times of the Lasso regression model with the periodic model retraining for these two benchmark applications, where the batch size is 10 and window size is ∞ (i.e., the predictive model is retrained after every 10 data points and each time all historical data tuples are utilized for model retraining). We can clearly see that the predicted results have the same pattern (or trend) as that of the measured execution times. This also validates our hypothesis that the devised micro-benchmarks can properly assess the level of resource contention at run-time.

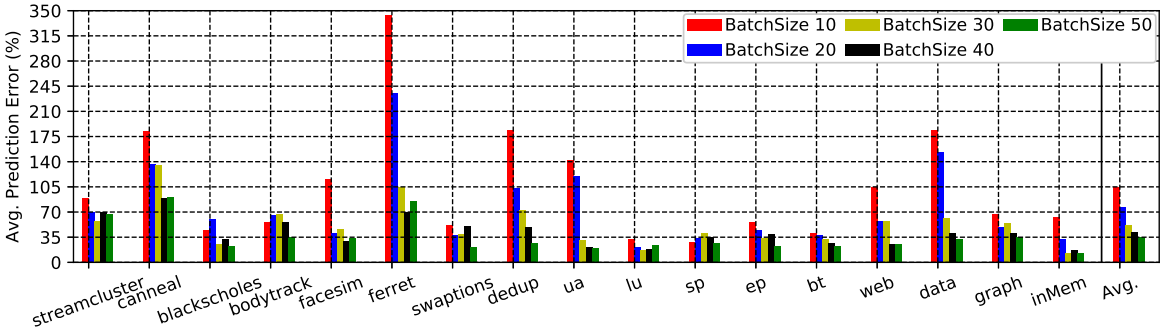
6.2.1 Prediction Accuracy vs. Batch/Window Sizes

As discussed in Section 4.1, we adopted 2-degree polynomial regression technique for periodic model retraining. In particular, we train the predictive model in periodic model retraining with the Lasso regression [49] and have implemented it using the Scikit-learn library (version 0.19.2) [56]. We tuned the regression-based model with an alpha of 1 and a tolerance value of 0.001.

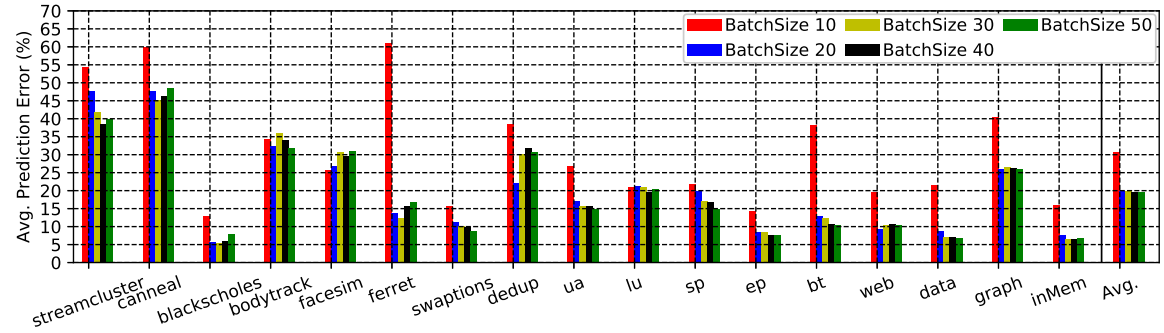
Fig. 5a first shows the average prediction errors of the Lasso regression models with periodic model retraining for the benchmark applications with different batch sizes (i.e., 10, 20, 30, 40 and 50). Here, the window size is set as 1; that is, only the data tuples in the last batch are utilized to retrain the predictive model at run-time, which will be used to predict the executions of the application in the next batch. From the figure, we can see that, although having larger batch sizes reduces the adaptation frequency of the predictive model, more accurate model can be obtained with more historical data tuples being used in the last batch, which generally results in smaller prediction errors. In particular, for *ferret*, the average prediction error is reduced from about 341% to 82% for the batch sizes of 10 and 50, respectively. The overall average prediction errors for all the benchmarks are reduced from 105% to 34% when the batch size increases from 10 to 50.

Fig. 5bc further show the average prediction errors of the Lasso regression models with the periodic model retraining for the cases of window sizes being 5 and 10, respectively. Clearly, when the window size increases from 1 to 5 (i.e., the number of historical data tuples for retraining the model increases from 10 to 50), the average prediction errors can be significantly reduced where the overall average for all applications can reduce from 112% to 24%. When the window size increases to 10, the prediction errors can be further reduced, but with relatively less magnitude. From the results, we can see that having more historical data tuples for retraining can generally improve the accuracy of the predictive model in periodic model retraining. However, such benefits normally reduce as more data tuples are included with larger window sizes. When the window size is 10, we can see that the overall average prediction errors are almost the same (about 18%) for all different batch sizes. That is, once the number of data tuples in the training window reaches a certain number (e.g., 100), the benefit of having even more data tuples is very limited.

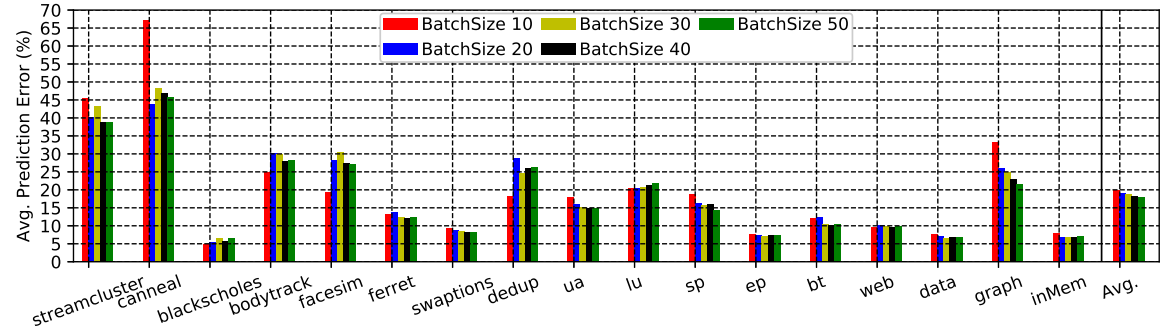
For the case of window size being ∞ (i.e., where all historical data tuples are used for retraining the predictive model), Fig. 6 further shows the average prediction errors of periodic model retraining for the benchmark applications with different batch sizes. In particular, we include the case with the batch size of 1 to illustrate the limit of increasing the retraining frequency. From the results, we can see that the prediction errors do decrease compared to the cases of window sizes being 5 or 10, but with very limited improvements.



a. Varying batch sizes with window size of 1.



b. Varying batch sizes with window size of 5.



b. Varying batch sizes with window size of 10.

Fig. 5: Prediction errors of the Lasso Regression Models with periodic model retraining for different batch/window sizes.

Moreover, even when the model is retrained after each iteration of executing the applications (for the case of batch size being 1), the overall prediction errors reduce less than 5%. However, as shown below, the overhead of periodic model retraining for the batch size of 1 can be prohibitive and prevent it from being deployed at run-time.

TABLE 1: Overall time (sec.) to retrain/predict.

Batch Size	Win. of 1	Win. of 5	Win. of 10	Win of ∞
1	-	-	-	1325.7
10	12.2	21.8	31.1	129.8
20	7.6	15.8	24.8	64.0
30	5.5	13.4	21.3	40.8
40	4.5	12.2	19.9	31.3
50	3.8	10.8	17.2	22.9

6.2.2 Model Retraining and Prediction Overheads

In addition to the prediction accuracy of the Lasso regression models with the periodic model retraining approach, it is also important to report its run-time overheads, which can be a evaluation metric for the approach being utilized as an online predictor. Table 1 shows the overall time required for the periodic model retraining to retrain the Lasso regression predictive models and to predict all 1000 data points under the different settings of batch/window sizes. Not surprisingly, with increased batch sizes, the retraining frequency decreases and it leads to less overhead. On the other hand, when window size increases, more historical data tuples are utilized for model retraining, which results in higher run-time overheads. In particular, for the case of batch size being 1 and window size of ∞, it can take more than 1325 seconds to process these 1000 data points. In comparison, for the case of window size

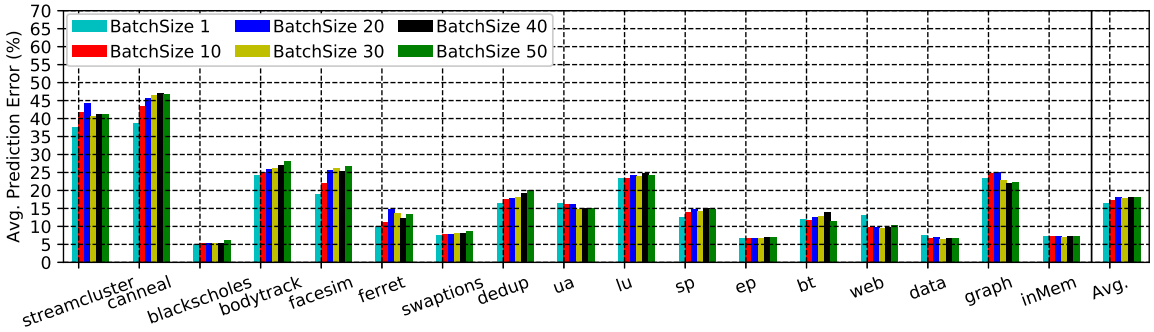


Fig. 6: Prediction errors of periodic model retraining: window size of ∞.

being 5 and batch size of 50, it takes only 10.8 seconds (a reduction of more than $120X$), which however can result in almost the same level of prediction errors (with only up to 3% difference).

TABLE 2: Average time (sec.) to re-train/predict per batch.

Batch Size	Win. of 1	Win. of 5	Win. of 10	Win of ∞
1	-	-	-	1.32
10	0.12	0.22	0.31	1.31
20	0.15	0.32	0.50	1.30
30	0.16	0.40	0.64	1.23
40	0.18	0.50	0.82	1.30
50	0.20	0.56	0.90	1.20

Table 2 shows the average time to retrain the model using a new batch of data and to predict the execution time of the next batch of executions. Here, we can find that, even with a windows size of 10 and batch size of 50, the time to retrain and predict all executions of an application in the next batch is 0.9 seconds. In the case of batch size 20 and window size 10, the time to retrain and predict a batch is only 0.5 seconds. This time is much shorter than the execution time of the shortest PARSEC benchmark, which is at least 20 seconds on our VM. This low overhead demonstrates that periodic model retraining is efficient for online prediction. For the window size of infinity (considering all historical data), we can observe that with different batch sizes, the average overhead is roughly the same. This is due to the fact that model retraining will happen on the full history, similarly for all batch sizes, and the time to predict the next batch of data is negligible and only a few CPU cycles for different batch sizes using Intel Advanced Vector Extensions (AVX) [62] vector processing capability. Note that, as presented earlier in Table 1, this time does not reflect the overall time spend on retraining and prediction. For instance, the batch size of 10 requires 99 times model retraining and prediction, whereas, with the batch size of 50, this process will be repeated only 19 times.

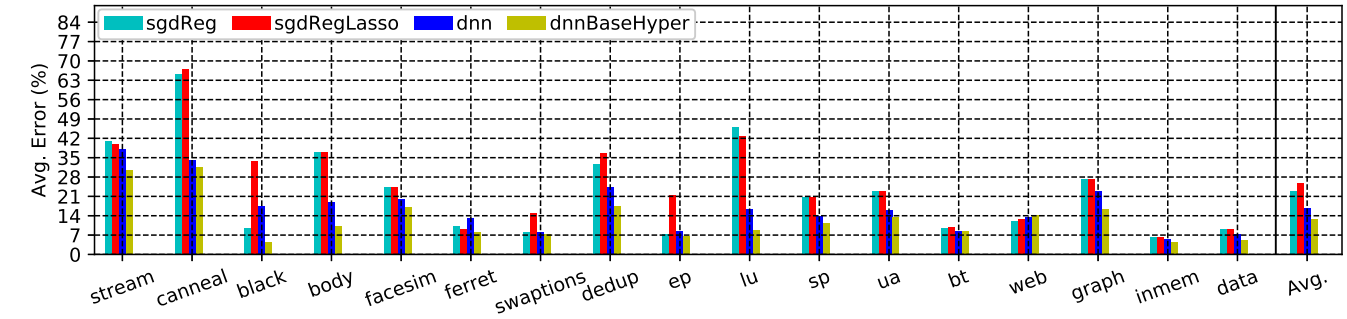
6.3 Progressive Modeling on Private Cloud

For the progressive modeling approach, in addition to the two SGD Regression models, which are denoted as *sgdReg* and *sgdRegLasso*, respectively, and the progressive neural-network model (denoted as *dnn*), we consider a neural-network model with hyper-parameter optimization, which will be used as the baseline for comparison and denoted as *dnnBaseHyper*. Here, the baseline DNN model is obtained by utilizing all the historical data tuples and the Tree Parzen Estimator (TPE) technique to explore the provided search space and find optimal hyper-parameters for the model [15].

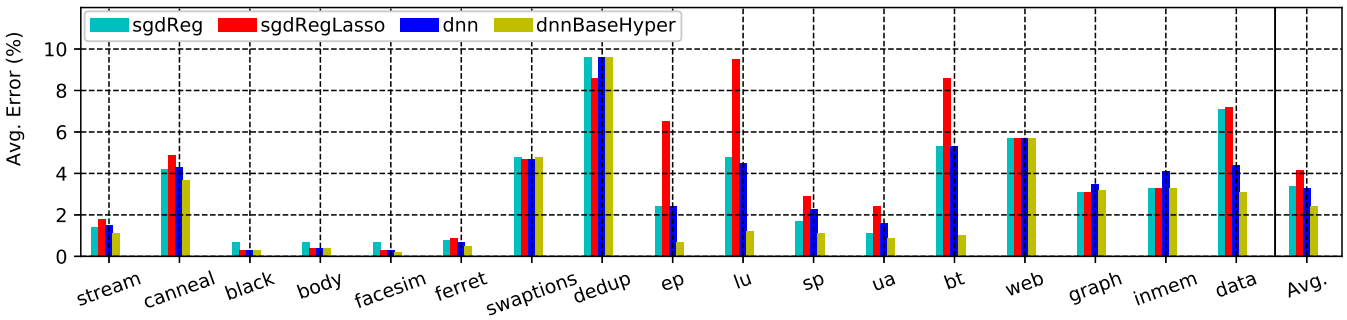
Another factor to consider for the progressive modeling is the batch size. From Section 5, we know that only the data tuples in the newly collected batch will be utilized to update the parameters of the predictive models in the progressive modeling approach. Here, based on the evaluation results in the last section, we considered the batch size of 50 aiming at a good trade-off between model prediction accuracy and run-time overhead.

Fig. 7a first shows the average prediction errors of the considered predictive models with the progressive modeling on the private cloud for the 17 benchmark applications. Here, we can see that, the progressive neural-network model has an average prediction error of 16% while the SGD regression models have average prediction errors of 22% and 25%, respectively. This implies that the progressive neural-network model can obtain 7% to 9% improvement in prediction accuracy. This comes from its ability of knowledge transfer by updating its model structure as well as parameters with additional data tuples. On the other hand, the baseline DNN model with hyper-parameter optimization has the average prediction error of 13%, where the improvement is marginal (i.e., 3%), especially considering its excessive training time (about 20 minutes) due to the time-consuming process of hyper-parameter optimization.

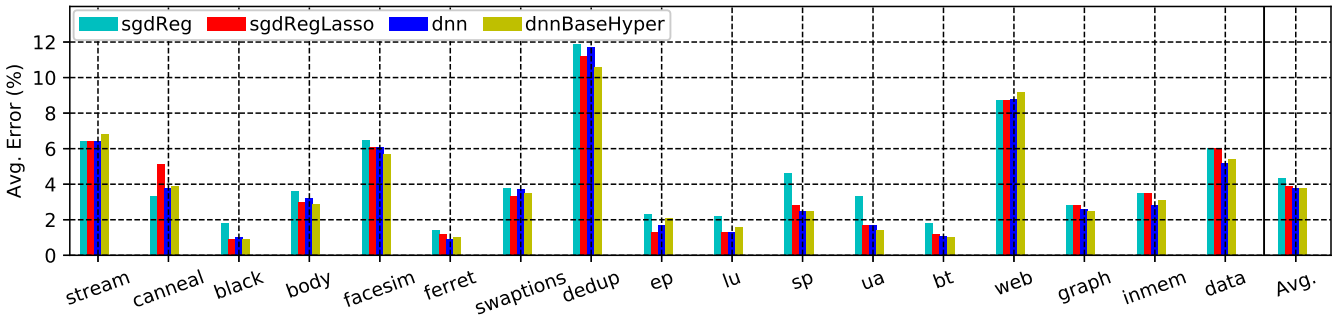
In addition, when comparing to the results of the periodic model retraining (as shown in Fig. 5), we can see that the prediction accuracy of the progressive neural-network model under the progressive modeling approach with the batch size of 50 is comparable to that of the Lasso regression model with the batch size of 1 and all historical data tuples. However, as presented in Table 3, the progressive modeling is more efficient, where the average overhead of model training and prediction is only 0.25 seconds per batch and 4.9 times less than that of the periodic model retraining approach.



a. Private Cloud.



b. Amazon AWS Cloud.



c. Google GCE Cloud.

Fig. 7: Prediction errors of different predictive models with the progressive modeling on the private and public clouds.

TABLE 3: Progressive Modeling Overhead (sec.)

Time \ Alg.	SGD Regression	Progressive DNN
Overall	1.75	4.75
Average (per batch)	0.09	0.25

6.4 Progressive Modeling on Public Clouds

We have also conducted extensive experiments on real public clouds, where we have no control on the type, intensity and sources of resource contentions, to evaluate the prediction accuracy for the executions of considered benchmark applications with the proposed progressive modeling approach. As two widely adopted public clouds, we have run experiments on both Amazon Web Services (AWS) and Google Compute Engine (GCE). For AWS, *m5d.4xlarge* instance type with 16 CPUs, 64GB memory and 80GB standard EBS SSD have been selected to execute the selected 17 benchmark applications. Similarly, on GCE, we used a VM of type *n1-standard-16* that has 16 VCPUs, 60GB of memory and 80GB SSD drive to execute the 17 benchmark applications. The same as in the private cloud, in the experiments, the micro-benchmarks first run within the VM followed by the executions of the benchmark applications to collect the data tuples related to the experienced contention levels in different resources. A total of 700 to 1000 data tuples for each of the benchmark application have been collected over the course of a month. The number of collected data tuples for each application varies due to their different length of execution and the cost constraint.

Fig. 7ab show the average prediction errors of the considered predictive models under the progressive modeling approach for the 17 benchmark applications on AWS and GCE, respectively. Here, we can see that, the prediction errors are significantly lower when compared to that of the private cloud. The main reason is that, the resource contention of the VMs on the AWS and GCE clouds is much less intensive compared to the setting of the private cloud. With such low level of resource contention, even the SGD Regression models

can obtain quite accurate prediction on the performance of the applications, where similar prediction results from different predictive models under the progressive modeling approach can be observed. Specifically, from the figure, we can see that the prediction accuracy achieved by different predictive models varies by at most 1% with the progressive neural-network model performs slightly better. However, we would like to point out that, the model adaptation overhead for the SGD regression algorithms is 2.7 times less when compared to that of the progressive neural-network model (Table 3), which make them more appealing for being utilized in the online setting.

7 Limitations and Future Work

Multi-VM applications: In this study, we considered performance modeling and prediction of single-VM applications. However, with the growing need for more computation power, applications running on Multi-VM will become more prominent. In our future work, we will extend this study to model and predict the performance of multi-VM applications. This requires closely monitoring the resource contention experienced by each collaborating VM and evaluating its effects on the overall application performance. For instance, high contention levels may slow down a VM, negatively affecting an application's overall performance and vice versa. Thus, micro-benchmarks should execute on each VM before the target application to collect contention information. Then, the results should be sent to a managing VM for modeling and prediction. Additionally, for multi-VM applications, micro-benchmarks should be devised to evaluate the contention in shared network resources (i.e., Network bandwidth/latency) between the collaborating VMs. High network contention between the VMs translates to a longer communication duration, negatively affecting the application performance.

Action-driven performance modeling: Based on a predicted performance of an application, different actions can be taken to satisfy the required performance needs. These actions include migration to a new VM, change in the VM configurations (scale-up), or increase in the number of the VMs for load-balancing or multi-VM processing (scale-up) of incoming requests. In our future work, we plan to evaluate the best actions that can be taken based on the model's predicted performance and accuracy. This study can help to identify the performance modeling configurations (i.e., batch and windows size) and approaches (i.e., adaptive retraining or progressive training) that can support the required accuracy for actions considered.

8 Conclusions

Applications running in cloud environment can incur performance variations due to resource contention caused by collocated VMs and their applications on the same host machine. Hence, it is important for ordinary cloud users to have accurate predictions on the executions of their application to make better planning of their operations and expenditure. In this paper, considering the changes in resource contention over time, we studied two online performance modeling and prediction approaches: periodic model retraining and progressive modeling, where both the regression and neural-network based predictive models have been considered. Specifically, based on the profiling technique, a few micro-benchmarks are utilized to probe the resource contention of major resources (such as CPU, memory and disks) of a target VM. The in-place profiled information from the micro-benchmarks and the measured execution times of an application will be used to retrain or update the considered predictive models. For the periodic model retraining approach, the sliding-window technique is utilized to control the retraining frequency and the number of historical data tuples to retrain the models.

Our evaluation results show that, for the periodic model retraining, the prediction errors of the considered models generally decrease with higher retraining frequencies and more historical data tuples, which in turn leads to higher run-time overheads. Moreover, for the progressive modeling approach, both the regression models and progressive neural-network model can obtain better prediction accuracy with a moderate batch size of 50, with much less overhead compared to that of the periodic model retraining. In addition, for public clouds where the resource contention is relatively low, the regression models can achieve comparable prediction accuracy as the progressive neural-network model with less run-time overhead.

References

[1] P. Leitner and J. Cito, "Patterns in the Chaos - A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 3, 2016.

[2] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, "ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computers," in *Proc. of Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[3] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," in *USENIX Annual Technical Conference (USENIX ATC)*, 2013.

[4] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Modeling performances of concurrent big data applications," *Software: Practice and Experience*, vol. 45, no. 8, pp. 1127–1144, 2015.

[5] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *IEEE Int'l Conf. on High Performance Computing and Communications (HPCC)*, 2015.

[6] 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 *Proc. of Annual IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*, 2011.

[7] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters," in *Proc. of Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[8] F. Romero and C. Delimitrou, "Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems," in *Proc. of Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018.

[9] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *Proc. of the 40th Annual Int'l Symposium on Computer Architecture (ISCA)*, 2013.

[10] N. Mishra, J. D. Lafferty, and H. Hoffmann, "ESP: A Machine Learning Approach to Predicting Application Interference," in *IEEE International Conference on Autonomic Computing (ICAC)*, 2017.

[11] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Middleware*. Springer, 2008, pp. 366–387.

[12] V. Shrivastava and D. Bhilare, "Cbud micro: A micro benchmark for performance measurement and resource management in iaas clouds," *Int'l Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 11, pp. 433–437, 2013.

[13] J. Scheuner and P. Leitner, "Estimating Cloud Application Performance Based on Micro-Benchmark Profiling," in *Int'l Conf. on Cloud Comput. (CLOUD)*, 2018.

[14] M. Baughman, R. Chard, L. Ward, J. Pitt, K. Chard, and I. Foster, "Profiling and predicting application performance on the cloud," in *IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, 2018.

[15] 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," in *The IEEE Int'l Conference on Cloud Engineering (IC2E); A longer version is availalbe as arXiv:1908.04491*, 2020.

[16] H. Moradi, W. Wang, and D. Zhu, "Adaptive Performance Modeling and Prediction

of Applications in Multi-Tenant Clouds," in *Proc. of the IEEE Int'l Conf. on High Performance Computing and Communications (HPCC)*, 2019.
[17] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016.
[18] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.
[19] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.
[20] D. H. Bailey, "NAS parallel benchmarks," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1254–1259.
[21] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *Proc. of the Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.
[22] Amazon Web Services, https://aws.amazon.com/ec2/.
[23] Google Compute Engine, https://cloud.google.com/.
[24] W. Zhang, B. Li, D. Zhao, F. Gong, and Q. Lu, "Workload prediction for cloud cluster using a recurrent neural network," in *Intll Conf. on Identification, Information and Knowledge in the Internet of Things (IIKI)*, Oct 2016, pp. 104–109.
[25] Y. Yu, V. Jindal, F. Bastani, F. Li, and I. Yen, "Improving the smartness of cloud management via machine learning based workload prediction," in *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, July 2018, pp. 38–44.
[26] J. Kumar and A. K. Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution," *Future Generation Computer Systems*, vol. 81, pp. 41 – 52, 2018.
[27] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An efficient deep learning model to predict cloud workload for industry informatics," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 7, pp. 3170–3178, July 2018.
[28] Z. Huang, J. Peng, H. Lian, J. Guo, and W. Qiu, "Deep recurrent model for server load and performance prediction in data center," *Complexity*, vol. 2017, 2017.
[29] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Virtual Machines," in *Proc. of ACM Symposium on Cloud Computing*, 2011.
[30] Y. Sfakianakis, C. Kozanitis, C. Kozyrakis, and A. Bilas, "QuMan: Profile-based Improvement of Cluster Utilization," *ACM Transactions on Architecture and Code Optimization (TACO)*, no. 3, pp. 27:1–27:25, 2018.
[31] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective cloud server provisioning for predictable performance of big data analytics," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 5, pp. 1036–1051, May 2019.
[32] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *IEEE Int'l Conference on High Performance Computing and Communications (HPCC)*, Aug 2015.
[33] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Modeling performances of concurrent big data applications," *Software: Practice and Experience*, vol. 45, no. 8, pp. 1127–1144, 2015.
[34] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, "Stage aware performance modeling of dag based in memory analytic platforms," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 2016, pp. 188–195.
[35] H. Mao, Z. Qi, J. Duan, and X. Ge, "Cost-performance modeling with automated benchmarking on elastic computing clouds," *Journal of Grid Computing*, vol. 15, no. 4, pp. 557–572, 2017.
[36] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, "Teastore: A micro-service reference application for benchmarking, modeling and resource management research," in *2018 IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 223–236.
[37] A. Iosup, M. Capotă, T. Hegeman, Y. Guo, W. L. Ngai, A. L. Varbanescu, and M. Verstraaten, "Towards benchmarking iaas and paas clouds for graph analytics," in *Workshop on Big Data Benchmarks*. Springer, 2014, pp. 109–131.
[38] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for hpc applications: A user-oriented approach," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 524–533.
[39] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker, "Cloud benchmarking for maximising performance of scientific applications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 170–182, 2019.
[40] 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 *Proc. of ACM Symposium on Cloud Computing (SoCC)*, 2017.
[41] J. Scheuner and P. Leitner, "Estimating Cloud Application Performance Based on Micro-Benchmark Profiling," in *IEEE Int'l Conference on Cloud Computing (CLOUD)*, 2018.
[42] M. Mao and M. Humphrey, "Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," in *Proc. of Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.
[43] R. Begam, W. Wang, and D. Zhu, "Virtual machine provisioning for applications with multiple deadlines in resource-constrained clouds," in *Proc. of the IEEE Int'l Conference on High Performance Computing and Communications (HPCC)*, 2017.
[44] R. Begam, H. Moradi, W. Wang, and D. Zhu, "Flexible vm provisioning for time-sensitive applications with multiple execution options," in *Proc. of the IEEE Int'l Conference on Cloud Computing (CLOUD)*, 2018.
[45] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, "Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 89–100, Mar. 2013.
[46] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 153–167.
[47] J. Kiefer, J.; Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
[48] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.
[49] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.
[50] A. E. Hoerl and R. W. Kennard, "Ridge regression: applications to nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 69–82, 1970.
[51] B. Li, J. Wang, Y. Li, and Y. Song, "An improved on-line sequential learning algorithm for extreme learning machine," in *International Symposium on Neural Networks*. Springer, 2007, pp. 1087–1093.
[52] N. Wang, J.-C. Sun, M. J. Er, and Y.-C. Liu, "Hybrid recursive least squares algorithm for online sequential identification using data chunks," *Neurocomputing*, vol. 174, pp. 651 – 660, 2016.
[53] J. Mandel, "Use of the singular value decomposition in regression analysis," *The American Statistician*, vol. 36, no. 1, pp. 15–24, 1982.
[54] T. Zhang, "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms," in *Proc. of Int'l Conference on Machine Learning*, 2004.
[55] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.
[56] Scikit-learn, http://scikit-learn.org/stable/.
[57] R. Moore and J. DeNero, "L1 and l2 regularization for multiclass hinge loss models," in *Symposium on Machine Learning in Speech and Language Processing*, 2011.
[58] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.
[59] N. Ebadi, B. Lwowski, M. Jaloli, and P. Rad, "Implicit life event discovery from call transcripts using temporal input transformation network," *IEEE Access*, vol. 7, pp. 172 178–172 189, 2019.
[60] R. Venkatesan and M. J. Er, "A novel progressive learning technique for multi-class classification," *Neurocomputing*, vol. 207, pp. 310 – 321, 2016.
[61] C. Delimitrou and C. Kozyrakis, "iBench: Quantifying interference for datacenter applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2013.
[62] Intel, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

Hamidreza Moradi is currently a Ph.D. student in the Department of Computer Science at The University of Texas at San Antonio. His research interests include cloud computing, performance prediction and modeling, and resource management and scheduling algorithms.

Wei Wang holds a Ph.D. in computer science from University of Virginia in 2015. He is currently an assistant professor at the Computer Science Department of the University of Texas at San Antonio. His research interests include system software, cloud computing, computer architecture and software engineering. He is a member of the IEEE.

Dakai Zhu received the PhD degree in Computer Science from University of Pittsburgh in 2004. He is currently a Professor in the Department of Computer Science at the University of Texas at San Antonio. His research interests include real-time systems, cloud computing, power aware computing and fault-tolerant systems.He was a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2010. He is a senior member of the IEEE.