A Self-Optimized Generic Workload Prediction Framework for Cloud Computing

Vinodh Kumaran Jayakumar
Department of Computer Science
The University of Texas at San Antonio
rvn028@my.utsa.edu

Jaewoo Lee
Department of Computer Science
The University of Georgia
jaewoo.lee@uga.edu

In Kee Kim
Department of Computer Science
The University of Georgia
inkee.kim@uga.edu

Wei Wang
Department of Computer Science
The University of Texas at San Antonio
wei.wang@utsa.edu

***Abstract*—The accurate prediction of the future workload, such as the job arrival rate and the user request rate, is critical to the efficiency of resource management and elasticity in the cloud. However, designing a generic workload predictor that works properly for various types of workload is very challenging due to the large variety of workload patterns and the dynamic changes within a workload. Because of these challenges, existing workload predictors are usually hand-tuned for specific (types of) workloads for maximum accuracy. This necessity to individually tune the predictors also makes it very difficult to reproduce the results from prior research, as the predictor designs have a strong dependency on the workloads.**

In this paper, we present a novel generic workload prediction framework, LoadDynamics, that can provide high accuracy predictions for any workloads. LoadDynamics employs Long-Short-Term-Memory models and can automatically optimize its internal parameters for an individual workload to achieve high prediction accuracy. We evaluated LoadDynamics with a mixture of workload traces representing public cloud applications, scientific applications, data center jobs and web applications. The evaluation results show that LoadDynamics have only 18% prediction error on average, which is at least 6.7% lower than state-of-the-art workload prediction techniques. The error of LoadDynamics was also only 1% higher than the best predictor found by exhaustive search for each workload. When applied in the Google Cloud, LoadDynamics-enabled auto-scaling policy also outperformed the state-of-the-art predictors by reducing the job turnaround time by at least 24.6% and reducing virtual machine over-provisioning by at least 4.8%.

***Keywords*-Cloud Computing; Workload Prediction; Long Short-Term Memory; Self-Optimized Framework; Resource Management;**

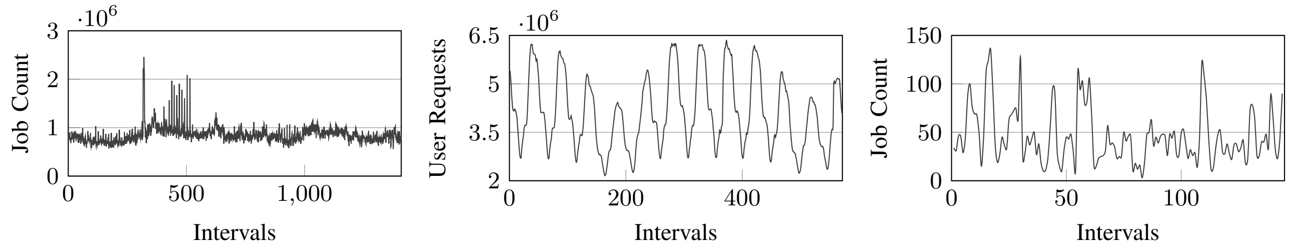
I. Introduction

The accurate prediction of future workloads, such as the number of jobs or user requests that will arrive in the next time interval, has long been recognized as a primary requirement for effective auto-scaling [1]–[3]. With accurate workload prediction, users or cloud service providers can design better auto-scaling policies or VM scheduling managers to properly provision virtual machines (VM) or containers in advance to avoid resource over- and under-provisioning, as well as the negative performance impact from resource cold startup [4], [5], which may lead to Service Level Agreement (SLA) violations or excessive cloud usage cost.

However, accurate workload prediction has been a challenging problem due to the large variations in the workload patterns (e.g., cyclic, bursty or increasing) across job types [6]. Fig. 1 shows the job workload traces from the Google data centers [7], Facebook data centers [8] and Wikipedia [9]. As shown in Fig. 1, the workload patterns drastically vary among different cloud applications. For example, the Wikipedia workload indicates a strong seasonality, while other workloads do not exhibit any clear periodicity. Moreover, the pattern within a workload may also change over time, as illustrated by the high spikes in the first half of the Google workload and the high fluctuations in the Facebook workload. The large variety of workload patterns demands that a workload predictor be tuned and optimized for each workload to ensure high accuracy. The changes within a workload require the predictor being sophisticated enough to recognize and accurately predict various patterns within one workload.

Ordinary cloud users, who usually do not have the expertise in statistics, time series and machine learning, are most likely hard-pressed to design such sophisticated predictors for their own workloads [10]. Ideally, a generic workload prediction framework, which can generate accurate predictors for various dynamic workloads, should be provided (e.g., by the cloud service providers) to help ordinary cloud users to maximize the benefit of auto-scaling.

Workload predictors from prior studies were usually developed for specific workloads or specific types of workloads [11]–[16]. These predictors were not designed to be generic and usually have low prediction accuracy when applied to other workloads. Furthermore, the need to hand-tuned predictors for each (type of) workload also made it difficult to reproduce the prediction results presented by previous research. We have also explored a generic workload prediction framework previously [3]. However, the accuracy of this prior framework still has room for improvement.



(a) Google Cluster (30min interval) (b) Wikipedia (30min interval) (c) Facebook Cluster (10min interval)

Fig. 1. Traces for three workloads with different patterns.

In this paper, we present LoadDynamics, a generic workload prediction framework that can generate highly-accurate workload predictors for various dynamic workloads. LoadDynamics employs the machine-learning (ML) model, Long-short Term Memory (LSTM) [17], to make predictions. Machine-learning models, however, cannot automatically produce accurate predictors for any workloads without proper hyperparameter tunning, such as tuning the number of layers of the model. To address this issue and improve the prediction accuracy for an individual workload, LoadDynamics further employs Bayesian Optimization to automatically optimize the hyperparameters of the model trained for each workload. Through this self-optimization, LoadDynamics is able to generate highly-accurate predictors optimized for individual workloads. Moreover, unlike ordinary feedforward neural network and many other machine-learning techniques, LSTM models can track relatively long-term dependencies in the data, making them potentially capable of detecting and predicting various patterns within a workload, given that the hyperparameters are carefully chosen [18], [19].

We applied the LoadDynamics to 14 workload configurations from four different types of jobs on the cloud. The average prediction error for the workloads using LoadDynamics is 18%, showing that LoadDynamics can provide predictions with high accuracy for various types of workloads with dynamic changes. When compared to three state-of-the-art workload predictors, LoadDynamics have lower prediction errors by 6.7%, 14.1% and 14.5%, respectively. To demonstrate that the higher accuracy from LoadDynamics can indeed benefit auto-scaling, we also implemented a predictive auto-scaling policy on Google Cloud with LoadDynamics. Experiment results show that the auto-scaling with LoadDynamics reduced the average turnaround time of the cloud applications by 24.6% and 38.1%, when compared to two state-of-the-art predictors. LoadDynamics can also improve cloud resource efficiency by reducing VM over-provisioning by 4.8% and 17.2%.

The contributions of this paper include:

1) The design of LoadDynamics, a workload prediction framework that automatically optimizes its predictors for individual workloads to provide high accuracy predictions for various workloads with dynamic changes. LoadDynamics provides significantly higher accuracy than state-of-the-art workload prediction techniques.
2) A thorough evaluation of LoadDynamics with 14 workload configurations of different patterns and application types to demonstrate that LoadDynamics can indeed provide high accurate predictions for various workloads with dynamic changes.
3) A case study to demonstrate that the highly-accurate predictions from LoadDynamics can further improve the effectiveness of auto-scaling on real public clouds.

The rest of this paper is organized as follows: Section II formally formulates the workload prediction problem; Section III presents the detailed design of LoadDynamics including its LSTM model, hyperparameter optimization, and the workflow; Section IV gives the experimental evaluation of LoadDynamics; Section V discusses the limitations and future work; Section VI discusses the related work; and Section VII concludes the paper.

II. Problem Definition and Motivation

A. Problem Definition

For a typical cloud system that executes ML training/inference jobs, HPC applications and serving web requests, a stream of jobs or requests arrive at different times. This stream of jobs or requests can be partitioned into time intervals and described with the number of jobs or requests arrived at each interval. For instance, the Google workload shown in Fig. 1a is partitioned into 30 minutes intervals. In the first 30-minutes internal, there were 814k jobs. In the second and third intervals, there were 757k and 791k jobs, respectively. The Wikipedia workload in Fig. 1b is also partitioned into 30-minutes-long intervals, where the first three intervals had 5.4, 5.2 and 4.9 million user requests, respectively.

Consider that at a certain time interval, a user wants to allocate VMs in advance for the jobs/requests arriving in the next interval. It is for the best interest of the user to allocate VMs that can exactly accommodate the jobs/requests at the next interval. Allocating too few VMs (under-provisioning) leads to additional VMs being created on-demand after the jobs/requests already arrive, potentially violating performance goals. Allocating too many VMs (over-provisioning) results in some VMs running idle, wasting money. To allocate the proper number of VMs, the user should know the number of jobs/requests that will arrive in the next time interval.

In this paper, we address this problem of predicting the number of jobs/requests that will arrive in the next time interval(s). For the rest of this paper, we define *Job Arrival*

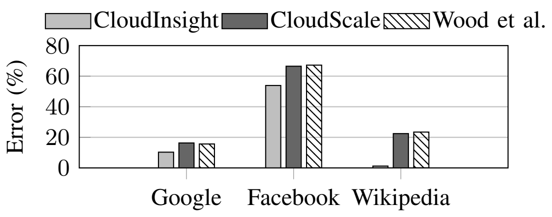


Fig. 2. Prediction errors (MAPE) of several prior predictive methodologies.

Rate (*JAR*) at an interval as the number of jobs/requests arrived in that interval. Without loss of generality, assume that at the $(i-1)$'th interval, the JARs at the current and previous intervals are known, and the JAR for the i'th interval is to be predicted. To ensure our prediction framework works for various types of workloads, the JARs at the interval i is predicted only based on the JARs from previous n intervals. Let $J_{i-1} \ldots J_{i-n}$ denote the actual JARs from the $(i-1)$'th interval to the $(i-n)$'th interval. Let P_i denote the predicted JAR at the i'th interval. The prediction problem of P_i can then be expressed as,

$$P_i = f(J_{i-1}, J_{i-2}, \ldots, J_{i-n+1}, J_{i-n}) \tag{1}$$

where f is a predictive model and n is the number of past JARs used in the prediction. The main task of LoadDynamics is to determine this function f for each workload. Note that, an implicit assumption made here is that J_i correlates with and depends on the previous n JARs, which is typically true for real workloads [20]. However, the exact number of previous JARs (i.e., the value of n) that correlates with J_i is workload-specific. Therefore, the value of n also needs to be determined.

B. Motivation

Fig. 2 shows the prediction errors (MAPE, defined in Section IV) for the three workloads shown in Fig. 1 using three different predictive methodologies based on prior studies, which are CloudInsight [3], CloudScale [1] and the predictor from Wood et al. [2]. These prediction methodologies were implemented the configurations recommended by their original authors. Note that, CloudInsight is an ensemble predictor that picks the best predictor from a group of predictive models, such as Random Forest, ARIMA and SVR. Therefore, CloudInsight represents the best prediction accuracy for the past research that employed these models.

As the figure shows, none of these existing predictive methodologies can always achieve less than 50% error for all workloads. Some of these predictive methodologies were designed for web workloads with clear seasonal trends. Consequently, they have low accuracy for the non-seasonal data center workloads. Such high errors often cause serious resource under- and over-provisionings. These errors also indicate there is room and necessity to improve the prediction accuracy.

III. The Design of LoadDynamics

A. LSTM and Hyperparameter Optimization

A pattern in a workload can usually be represented with the long-term and short-term trends/dependencies within the JARs

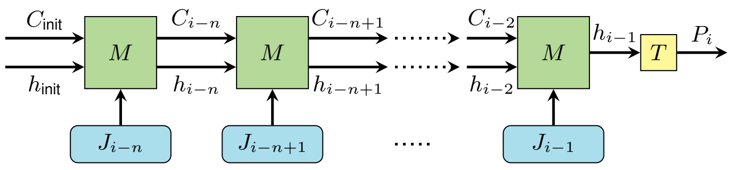


Fig. 3. Predicting JAR P_i with a one-layer LSTM model.

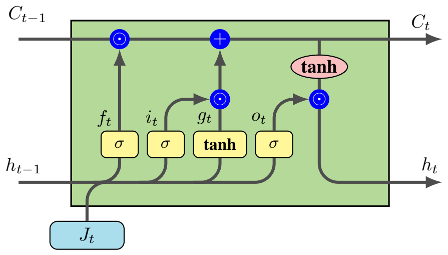


Fig. 4. Internal operation of a LSTM cell (M) at time interval t $(t \in \{i-n, i-n+1, \ldots, i-1\})$.

of continuous time intervals. For example, the long-term trend (JAR-dependency) of a workload may be that job counts continuously grow each day, whereas the short-term trend (JAR-dependency) might indicate that job counts fluctuate following the time-of-the-day. However, cloud workloads often yield complex non-linear dependencies which traditional time series prediction models (e.g., ARIMA and its variants) may not be able to capture [21]. To capture these trends/depends, we propose to use Long Short-term Memory (LSTM) network, as it is developed to efficiently learn long-term dependencies [17].

Fig. 3 shows how the LSTM predicts the JAR at the i'th interval, i.e., P_i. M represents a trained LSTM cell. C is a *cell memory* that serves as a long-term memory for temporal state information. h is a *hidden state* outputted from previous LSTM cell (M). Given the input sequence $\langle J_{i-n}, \ldots, J_{i-1} \rangle$ of length n, the LSTM network recursively updates values to obtain C_{i-2} and h_{i-2}. Finally, the lost output, h_{i-1}, is fed into a fully-connected layer T to produce the final output P_i.

Both C and h are determined by the internal operations of the LSTM cell (M), which are shown in Fig. 4. Let t be a general form of time intervals in the LSTM network and $t \in \{i-n, i-n+1, \ldots, i-1\}$. Four gates $(i_t, f_t, o_t,$ and $g_t)$ in M calculate the values of C_t and h_t by regulating the amount of information to flow along the network. At time interval t, the network takes J_t, h_{t-1}, and C_{t-1} as input and updates the memory and gate values as follows:

$$\begin{aligned}
i_t &= \sigma(W_i J_t + U_i h_{t-1} + b_i)\,, \\
f_t &= \sigma(W_f J_t + U_f h_{t-1} + b_f)\,, \\
o_t &= \sigma(W_o J_t + U_o h_{t-1} + b_o)\,, \\
g_t &= \tanh(W_g J_t + U_g h_{t-1} + b_g)\,, \\
C_t &= f_t \odot C_{t-1} + i_t \odot g_t\,, \\
h_t &= o_t \odot \tanh(C_t)\,,
\end{aligned}$$

where $\odot$ denotes the Hadamard product of two matrices, σ is the sigmoid function, and W_a, U_a and b_a represent the weights and bias for a gate, respectively. The input gate i_t determines

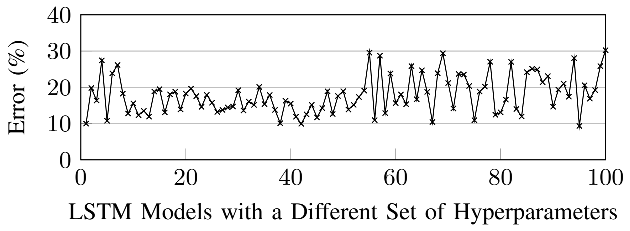


Fig. 5. Predictions errors (MAPE) for the Google workload using various LSTM modes with different hyperparameters.

how much new information from the input is written to the cell memory, and the forget gate f_t allows the network to erase past information from C_{t-1}. The output gate o_t controls how much information is output from C_t. One can control the capacity of an LSTM network to learn dependencies by adding more LSTM layers or increasing the number of units (i.e., the size of memory cell C_t) in an LSTM cell. This control means that the accuracy of an LSTM network is dependent on the proper choice of its hyperparameter values, such as the number of the LSTM layers, the size of C_t and the actual length of the history (the length n of input sequence).

However, finding a good set of these hyperparameters is a non-trivial task [10]. For example, when the history length n is too small, it is difficult for a model to learn a dependency that spans for a longer period of time. On the other hand, when n is too large, the model might learn irrelevant dependencies and suffer from exploding/vanishing gradient problem [22], resulting in poor prediction accuracy and yielding high computation overhead. A similar issue arises with the size of the cell memory C (the number of units) and the number of LSTM layers. The cell memory is represented by a vector $C \in \mathbb{R}^s$ of length s. If the size of the C vector and/or the number of layers are too large, it increases the complexity of the LSTM model. Complex models typically require larger data set to train, which may be infeasible to collect in practice. The increased model complexity may also increase the risk of overfitting, where the trained LSTM model may correspond too closely to the training data, losing the ability to reliably predict future data, which do not necessarily closely resemble the training data. The increased model complexity may also lead to higher computational cost. On the other hand, if the size of the C vector and/or the number of layers are too small, the class of predictors represented by the LSTM model may not be able to capture the complex temporal dependencies in the data, resulting in poor prediction accuracy [23], [24]. Besides the history length, the cell memory vector size and the number of layers, a fourth hyperparameter may also significantly affect the accuracy of the trained LSTM models, which is the size of the training data batches. Although the batch size does not affect the internal structure of the LSTM model like the other three hyperparameters, it may affect the effectiveness of the training process, and thus affect the accuracy of the trained model [10]. Therefore, we also consider the batch size as a hyperparameter in LoadDynamics.

The proper values of the hyperparameters, n (history length), s (size of C vector), the batch size, and the number of layers, depend on the workload and need to be determined before training M. Fig. 5 compares prediction errors (MAPE) of 100 different LSTM models on the Google workload (Fig. 1a), where values on x-axis correspond to different combinations of hyperparameters. It suggests that choosing a better combination of hyperparameters can lead to three times reduction in prediction errors. Unfortunately, manually selecting the best hyperparameters for each workload is infeasible given the large variety of workloads and their dynamic changes. The brute-force search of best hyperparameters is also impractical due to the combinatorial explosion in the number of combinations.

To address this challenge, we use *Bayesian Optimization* to intelligently search for a better set of hyperparameters for each workload and/or each portion of a workload [25]. Bayesian Optimization (BO) searches for the better hyperparameters with a non-linear regression technique called the Gaussian Process (GP) [26]. BO is an iterative optimization process. In each iteration, BO builds a regression model with GP using the sets of hyperparameters that are already explored and the accuracy of the LSTM models built with these sets. The regression model is then used to predict a new set of hyperparameters that is potentially better. This new set of hyperparameters is then used to train a new LSTM model, whose accuracy is evaluated with a cross-validation data set. After a fixed number of iterations, the best LSTM model found from these iterations is then chosen as the predictor.

Note that, besides BO, other optimization techniques were also proposed by prior work to optimize hyperparameters. In particular, we also experimented with random search and grid search [27], [28]. However, grid search was less effective than BO in improving the hyperparameters in our experiments. Random search could find hyperparameters with similar accuracy as those determined by BO. However, random search typically had longer execution time than BO in our experience. Therefore, LoadDynamics was designed to only employed BO.

B. *Workflow of LoadDynamics*

The general workflow of LoadDynamics is composed of three phases; (i) the model training, (ii) the hyperparameter optimization, and (iii) the prediction phase. Corresponding to these three phases, the past and future JARs in a workload consist of three parts; (a) the training data set with l samples used for training the model M and T, (b) the cross-validate data set with m samples used for hyperparameter optimization, and (c) the prediction of JAR. The first two parts – (a) and (b) – are from the past (*known*) JARs and the third part (c) is for the future (*unknown*) JARs. Fig. 6 shows the overall workflow of LoadDynamics. Note that each rectangle with a number in Fig. 6 means an internal step in LoadDynamics. Fig. 7 shows the three parts of the workload JARs.

As Fig. 6 shows, the training phase starts with a randomly selected set of hyperparameters. This set of hyperparameters is used to configure an initial LSTM model, which is then trained using the training data set (step 1). After training, a

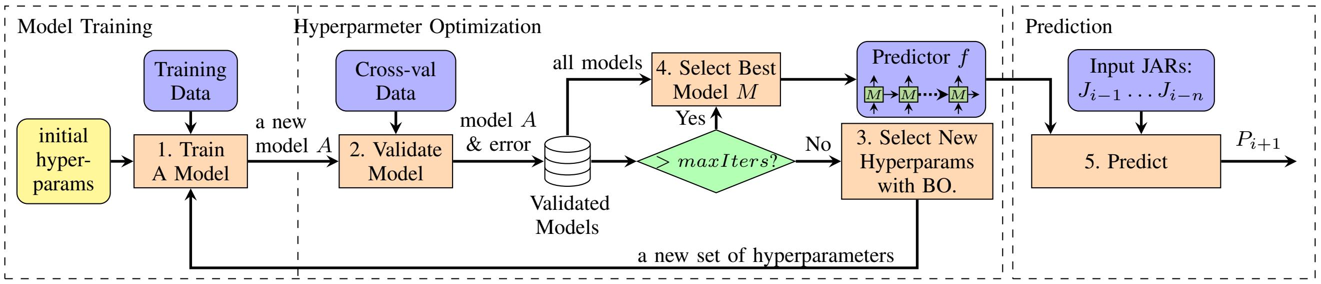


Fig. 6. The overall workflow of building a new predictor and making predictions with LoadDynamics.

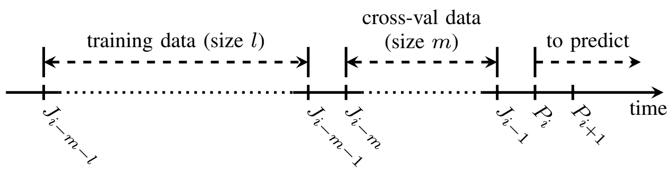
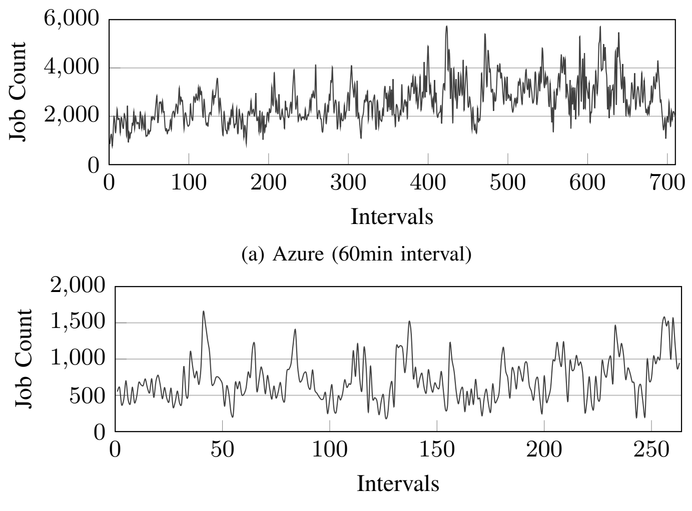


Fig. 7. Partitioning of the training data cross-validation data sets.

TABLE I
WORKLOADS USED FOR EVALUATION.

Trace	Type	Intervals (mins)
Wikipeida (wiki)	Web	5, 10, 30
LCG	HPC	5, 10, 30
Azure (AZ)	Public Cloud	10, 30, 60
Google (GL)	Data Center	5, 10, 30
Facebook (FB)	Data Center	5, 10



(b) LCG (30min interval)
Fig. 8. Workload Traces for Azure and LCG workloads.

new LSTM model A is generated. This model A includes both a new LSTM model M and a fully-connected layer T, as shown in Fig. 3. In step 2, A is then validated using the JARs in the cross-validation sets. For this validation, for all the JARs of $J_{i-1} \dots J_{i-m}$, their corresponding predictions $P_{i-1} \dots P_{i-m}$ are generated with A. Then the predicted JARs are compared with the actual JARs to calculated the average prediction error of A. After the validation, both A and its prediction error are stored in a database. For step 3, the hyperparameters of A and its error are also used to select a new and potentially-more-accurate set of hyperparameters using Bayesian Optimization from a predefined search space of possible hyperparameters. The new set of hyperparameters is then used to configure and train a new LSTM model in step 1. This train and optimization process is repeated for $maxIters$ iterations. After these iterations, all of the validated models are compared, and the model M with the lowest error is selected as the actual workload predictor f (step 4). This predictor f is then used to predict future JARs (step 5).

IV. EVALUATION

This section provides the experimental evaluations of LoadDynamics. These evaluations focused on the accuracy of LoadDynamics, as well as its benefits on auto-scaling.

A. Experiment Setup

Workloads. Five workloads from four categories were used to evaluate LoadDynamics. These workloads are: 1) Wikipedia (web application) workload from Wikibench [9]; 2) the high performance computing (HPC) workload from the LCG trace of the Grid Workload Archive [29]; 3) the public cloud workload from Microsoft Azure [30]; 4) the data center workload from Google [7]; and 5) the data center workload from Facebook [8]. The traces of the Google, Facebook and the Wikipedia workloads were shown in Fig. 1, and the traces of the Azure and LCG workloads are shown in Fig. 8. The characteristics of these workloads are summarized in Table I. To evaluate whether LoadDynamics works properly at different time granularity (which may have different workload patterns), these workloads were evaluated with varying interval lengths. For the Wikipedia, Google and LCG workloads, the intervals are 5, 10 and 30 minutes. For the Azure workload, as the JARs are very small at 5-minute intervals, it is evaluated with intervals of 10, 30 and 60 minutes. The Facebook workload is relatively short (just covering one day). To allow enough training samples, the Facebook workload is only evaluated with intervals of 5 and 10 minutes. Overall, 14 combinations of different workloads and intervals were used to evaluate LoadDynamics. For the rest of this paper, we refer to a workload with a specific interval as a *workload configuration*.

Metric. The prediction error is defined as the absolute percentage error of each prediction. For each workload configuration, the mean absolute percentage error (MAPE) of all prediction errors is reported. MAPE is formulated as $\frac{100\%}{n}\sum_i|\frac{P_i-J_i}{J_i}|$.

TABLE II
THE 21 PREDICTORS USED IN CLOUDINSIGHT [3] BASELINE.

Category (Preds #)	Workload Predictors
Naive (2)	*mean* and *k*NN
Regression (6)	Both local and global regression with linear, quadratic, and cubic models.
Time-series (7)	WMA, EMA, Holt-Winters DES, Brown's DES, AR, ARMA, and ARIMA.
ML (6)	Linear and Gaussian SVMs, Decision Tree, Random Forest, Gradient Boosting, and Extra Trees

TABLE III
THE HYPERPARAMETER SEARCH SPACE AND MAXIMUM ITERATIONS OF OPTIMIZATIONS.

Workload	Hist Len (*n*)	*C* size	Layers #	Batch #
Wiki	[1-512]	[1-100]	[1-5]	[16-1024]
LCC				
Azure				
Google				
Facebook	[1-100]	[1-50]		[8-128]

Baselines. For comparison, we also report the prediction errors from three state-of-the-art workload predictors, including CloudInsight [3], CloudScale [1] and Wood et al. [2]. CloudInsight is an ensemble model that dynamically chooses the best predictor from a pool of 21 predictors each built a different modeling techniques. Many of these techniques were employed by prior work in workload prediction, such as [13], [15], [16], [31], [32]. All the 21 predictors are listed in Table II. CloudInsight also dynamically rebuilds its predictors after every five intervals. CloudScale is based on Fast Fourier transform (FFT) and a discrete-time Markov chain. It uses FFT to detect repeating patterns in the workload. The detected pattern is then used with the Markov chain to predict the workload. Wood et al. employed robust linear regression to predict workloads. The model built with the linear regression is refined online to adapt with changes.

Implementation. LoadDynamics was implemented using Tensorflow [1], Scikit-learn [2] and GpyOpt [3]. The training and inference of LoadDynamics were executed on a 16-core Intel Xeon Platinum 8153 CPU. For LSTM model training, we used "mean square error" as the loss function and the Adam Optimization Algorithm [33] as the optimizer. For Bayesian Optimization, the Gaussian process [26] were used as the probabilistic model to conduct the regression and the "expected improvement" [34] was used as the acquisition function to select the next set of hyperparameters to examine/validate.

The sizes of the training sets (l) and cross-validation sets (m) are defined as follows. The first 60% JARs of each workload is set to be the training set, the next 20% is used as the cross-validation set, and the last 20% is used to test the accuracy of LoadDynamics.

Using Bayesian Optimization to search for the more-accurate hyperparameters requires defining the search space. This search space is expressed as the ranges of the potential hyperparameter values, including the ranges for the history length (i.e., the value of n), the C vector size, the number of LSTM layers and the batch size, as discussed in Section III. Table III gives the ranges of the potential values for each hyperparameter. Except for the Facebook workload, all other four workloads used the same ranges. These ranges are selected as they are large enough to cover the potentially more-accurate hyperparameter values. As shown later in the evaluation results, the best hyperparameters selected by LoadDynamics were typically smaller than the range maximums. Therefore, these ranges are likely to be sufficiently large even for the workloads that are not evaluated in this paper. For the Facebook workload, as it is short and thus cannot support a large history length, its ranges for the history length and C vector sizes were scaled down to be compatible the training data size. Besides the search space, employing Bayesian Optimization also requires defining the number of optimization iterations (i.e., the $maxIters$ Fig. 6). This iteration count represents the number of hyperparameter sets that will be generated with the BO. While the more sets generated, the better chance of finding high accuracy sets, more iterations also take more execution time. In our implementation of LoadDynamics, the iteration count was set to 100 for all workloads

[1]https://www.tensorflow.org/
[2]https://scikit-learn.org/
[3]https://sheffieldml.github.io/GPyOpt/

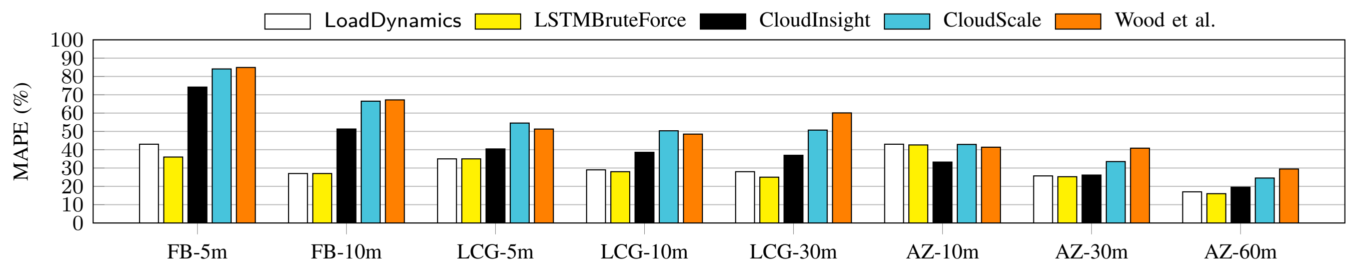
TABLE IV
MINIMUM AND MAXIMUM HYPERPARAMETER VALUES SELECTED BY LOADDYNAMICS.

Workload	Hist Len *n*	*c* size	Layers	Batch size
Wiki	35-102	7-98	2-4	18-400
LCG	12-176	5-69	1-4	16-250
Google	5–472	8-99	1-3	18-371
Azure	68-79	8-47	1-4	18-700
Facebook	17-38	1-78	1-4	16-521

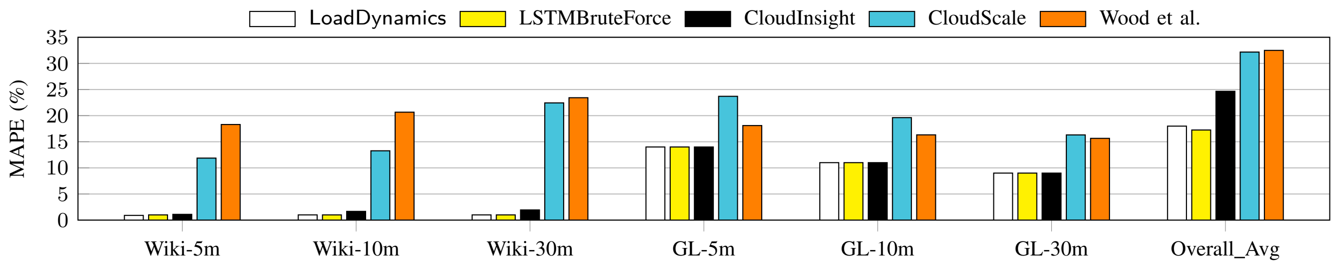
B. *Evaluation of LoadDynamics's Accuracy*

As stated previously, the workload predictor obtained by LoadDynamics for each workload was then tested on the last 20% JARs of the workload for this accuracy evaluation. The MAPEs of LoadDynamics for the last 20% testing data are reported in Fig. 9 for each workload configuration. The MAPEs of the baseline predictors from prior work were also included in Fig. 9 for comparison. Fig. 9 also includes the MAPE of the best LSTM predictor for each workload (the bar labeled with "LSTMBruteForce") obtained through brute-force searches in the hyperparameter optimization spaces (similar to the search spaces shown in Table III). One execution of brute-force search for one workload might take up to 6 weeks.

As Fig. 9 shows, LoadDynamics can provide highly accurate predictions for the evaluated workload configurations. The lowest MAPE of LoadDynamics was only 1%, when predicting for the three Wikipedia configurations. Except for Azure with 10-minutes intervals, LoadDynamics's MAPEs were always lower than those from CloudScale and Wood et al. predictors. Except for Azure with 10-minutes intervals, LoadDynamics's MAPEs were also either lower or similar to the prediction errors of CloudInsight. On average, LoadDynamics's MAPE is about 6.7% lower than CloudInsight, 14.1%



(a) Prediction errors for the Facebook (FB), LCG grids and Azure (AZ) workloads.



(b) Prediction errors for the Wikipedia (Wiki) and Google (GL) workloads, as well as the overall average MAPE for all workload configurations.

Fig. 9. Prediction errors (MAPE) of LoadDynamics and the baseline predictors.

lower than CloudScale and 14.5% lower than the Wood et al. predictors. CloudScale and the Wood et al. predictor employ predictive methodologies that were less capable of tracking complex workload patterns/dependencies, leading to higher errors than LoadDynamics. CloudInsight employs multiple predictive techniques to handle various patterns. However, these predictive techniques in CloudInsight are also heavily affected by their hyperparameters. As CloudInsight uses fixed hyperparameters, its accuracy is negatively affected if the fixed hyperparameters do not fit the current workload. Moreover, LoadDynamics's overall average MAPE is only 1% higher than the average MAPE of the brute-force-searched predictors, suggesting that our hyperparameter optimization method was capable of determining hyperparameters that are nearly optimal within the search pace. This near-optimal accuracy was also obtained with a much lower cost. Comparing to the 1-day to 6-weeks search time required by the brute-force search, LoadDynamics only spent at most three hours on each workload configuration. Note that, the time it took to make an inference with LoadDynamics's models was much smaller, which was less than 4.78ms.

Overall, except for a few cases with 5-min or 10-min intervals, the majority of LoadDynamics's MAPEs were less than 30%. The average MAPE of all 14 workload configuration is only 18%. These low errors show that LoadDynamics can indeed generate high accuracy predictions for various types of workloads. Moreover, as illustrated in Fig. 1 and Fig. 8, the Google, Facebook, Azure and LCG workloads all have pattern changes. Even with these pattern changes, LoadDynamics could still provide high accuracy predictions, showing the design of LoadDynamics allows it to detect and predict multiple patterns within one workload.

The highest MAPE of LoadDynamics was 43% for predicting the Facebook workload with 5-minutes interval and the Azure workload with 10-minutes intervals (both workload configurations had 43% error with LoadDynamics). This high error was mainly caused the relatively low JARs at each 5-min or 10-min interval. In general, we observed the LoadDynamics's MAPEs were higher when the time interval is smaller, for the Facebook, LCG and Azure workloads. For these workloads, smaller time intervals typically have small JARs. These smaller JARs are more susceptible to the random burstiness, making them more difficult to predict (random fluctuations typically lacks track-able patterns/dependencies). For larger workloads, such as Google and Wikipedia, the JARs are still fairly large at small intervals. Therefore, their prediction errors remained low even when predicted at small intervals with LoadDynamics.

Table IV gives the values of the hyperparameters that were selected by LoadDynamics (through BO). Due to space limitations, we could not provide the hyperparameters selected for each workload configuration. Instead, the minimum and maximum of the selected hyperparameters of all the intervals for a workload were reported. As Table IV shows, the selected hyperparameter values had very high variation, suggesting that it could be very challenging and time-consuming if the hyperparameters were to be determined manually. Therefore, the automatic hyperparameter optimization process provided by LoadDynamics is indispensable for building accurate and generic workload prediction techniques. This high variation also shows the necessity to individually optimize the LSTM model for each workload. Moreover, for the majority of the workload configurations, the selected hyperparameter values were typically below the maximums of the search space,

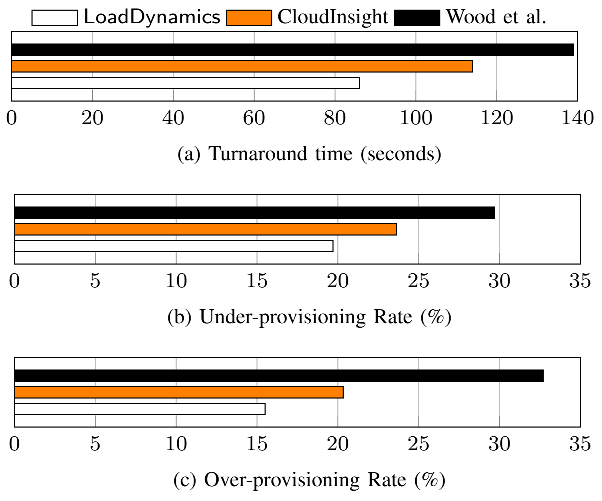


Fig. 10. Performance, VM underprovisioning rates and overprovisioning rates of different prediction techniques.

indicating the search space is likely to be sufficiently large.

C. *Evaluation of LoadDynamics with Auto-scaling*

To demonstrate that the higher accuracy brought by LoadDynamics can indeed improve the performance and resource efficiency of cloud systems, we applied the LoadDynamics to an auto-scaling policy that managed the VMs executing in the Google Cloud. For simplicity, in this auto-scaling evaluation, we assumed all jobs arrived at the beginning of an interval. The algorithm of this auto-scaling policy is described as follows. At each interval, the JAR for the next interval is predicted. Without loss of generality, assume the next interval is the i'th interval and P_i is predicted at the $(i-1)$'th interval. Right after the prediction, P_i VMs are created in advance in the $(i-1)$'th interval with the anticipation that P_i jobs will arrive at interval i. At the beginning of interval i, arrived jobs are sent to the created VMs to execute, with one VM for each job. If there are more than P_i jobs arrive at interval i (i.e., $J_i > P_i$), then VM under-provisioning occurs and more VMs will be created on-demand to accommodate the extra jobs. In this case of VM under-provisioning, the extra jobs require additional time to finish due to the VM startup time. If there are fewer than P_i jobs arrive (i.e., $J_i < P_i$), then VM over-provisioning occurs with the extra VMs running idle. In this case of VM over-provisioning, the extra VMs incur unnecessary costs.

For this auto-scaling evaluation, we used Google Cloud's *n1-standard-1* VMs, due to financial limitations. Because of the same cost issue, only a subset of the workloads and predictive techniques were evaluated. For the workload, only the Azure workload with 60-minutes intervals was evaluated. Additionally, to comply with the Google Cloud's resource quota and to ensure reasonable experiment cost, the JARs from the Azure workload were scaled down by 100 times so that at each interval there were less than 50 jobs arrived (i.e., less than 50 VMs needed to be created at each interval). We verified that this scale-down of JARs did not affect the predictions and the accuracy of the evaluated predictive techniques. For the predictive techniques, only LoadDynamics, CloudInsight and Wood et al. were evaluated. CloudScale's accuracy was similar to the Wood et al. predictor, and was thus dropped for this evaluation. Cloud Suite's *In-Memory Analytics* benchmark was used as the jobs to execute, mimicking a system serving machine-learning training and inference requests [35]. At each time interval, the time it took to finish all arrived jobs were recorded (i.e., the turnaround time), and the percentages of the under- and over-provisioned VMs (over the actual required VMs) were recorded.

Fig. 10 gives the average job turnaround time for all intervals, as well as the average under- and over-provisioning rates. As the figure shows, the jobs managed under LoadDynamics finished 24.6% faster than CloudInsight and 38.1% faster than the Wood et al. predictor. This improved performance was mainly due to the lower under-provisioning rates under LoadDynamics, as shown in Fig. 10b. More specifically, the under-provisioning rate of LoadDynamics was 4% lower than CloudInsight and 10% lower than the Wood et al. predictor. Besides performance, LoadDynamics also improved resource efficiency by having lower over-provisioning rates. The LoadDynamics's over-provisioning rate was 4.8% less than CloudInsight and 17.2% less than the Wood et al. predictor. These results indicate that the high prediction accuracy brought by LoadDynamics can indeed translate into actual performance gains and resource efficiency improvements.

V. Discussion and Futurework

Other Hyperparameters. In addition to the four hyperparameters explored in this paper, there are other hyperparameters related to the training and the structure of the LSTM models. For example, activation functions other than the $tanh$ function may be used. Other loss functions and training/optimization algorithms may also affect the accuracy of the trained model. Although in our experiments, we observed that tuning these hyperparameters did not improve the prediction accuracy for the evaluated workloads, these additional hyperparameters may affect the accuracy of LoadDynamics when it is applied to other workloads. These additional hyperparameters may also be optimized with LoadDynamics's current auto-optimization process. However, it may take a longer time to build predictors as the search space is much larger.

Online Adaptive Modeling LoadDynamics may experience high prediction errors if the workload completely changes to a new pattern that is not represented by any of the training data. In such a case, a completely new predictive model needs to be built to achieve high accuracy predictions. To handle such cases, LoadDynamics needs to be capable of detecting that a previously-unobserved new workload pattern occurs. It also needs to be able to adaptively retrain its model to handle such drastic pattern changes. We plan to explore this adaptive variant of LoadDynamics in the future.

VI. Related Work

There is a large body of works, proposed to predict diverse workloads (e.g., job arrival rates, resource demands) from

Cloud [7], [30], [36], HPC/Grid [29], and web applications [9]. The most common approach is to represent such workloads as time series data so that a set of various time series models has been applied. i.e., ES/WMA [13], [32], [37]–[39], AR [40]–[42], ARMA [14], [16], ARIMA [12], [15]. Moreover, ML and statistical-analysis based approaches (e.g., LR [2], [11], [43], [44], SVR [31], Random Forest [30], Gradient Boosting [30], and FFT [1]) have been adopted to forecast future workloads and to design proactive resource management mechanisms. However, the workload predictors from prior works tend to be optimized/trained on workloads of interests. As evaluated in [6], different predictors yield accurate prediction results only for particular workloads, indicating that it is very challenging to apply such predictors to *different* or *unknown* workload patterns due to the lack of generality.

To overcome the limitation from the above approaches, multi-predictor based approaches have been explored [3], [20], [45]–[49]. The approaches employed a set of predictors, such as multiple time series and ML models, and tried to improve adaptability to different workloads or sudden changes in a workload by intelligently using/combining the predictors. In particular, CloudInsight [3] employs any predictors of users' choice, leverages multi-class regression to estimate the future accuracy of the predictors, and allocates more weights to the predictors that perform best in the near future. Unfortunately, the multi-predictor based approaches need to process several predictors in parallel, so they have unnecessary computation overhead for making predictions. Furthermore, the predictors used in these approaches have to be chosen by the users, so selecting the predictors used in the frameworks is challenging for ordinary users who do not have a background in workload prediction and characterization. Additionally, the hyperparameter tuning problem also exists in these studies. Poorly chosen hyperparameters can still negatively affect the prediction accuracy for these multi-predictor approaches.

Due to the proliferation of deep learning [50], there are several attempts to apply deep learning models to predict cloud workloads [51]–[56]. These works are in spirit similar to LoadDynamics in terms of considering LSTM [17] or LSTM-variants as a base predictor. Unfortunately, the approaches commonly have the following drawbacks. First, the LSTM model is narrowly trained from limited workloads, often resulting in overfitting to the training dataset, and the evaluations with real-world traces are not comprehensive. Also, it is unclear how the approaches tune/update the model hyperparameters, which often changes the prediction accuracy. In fact, without knowing these hyperparameters, it becomes very difficult to reproduce the prediction accuracy from prior studies. Differing from the prior works, LoadDynamics leverages Bayesian Optimization for the hyperparameter tunning, thus LoadDynamics becomes more *adaptive* and can quickly adjust the LSTM predictor, resulting in effectively addressing different cloud workloads. In Section IV, we evaluated LoadDynamics with multiple workloads from Google, Azure, HPC, and web applications, and the results showed that LoadDynamics always outperforms state-of-the-art workload predictors, indicating that LoadDynamics is generic enough to handle diverse workloads. Employ model self-optimization within the LoadDynamics framework also makes it easier to reproduce the results of LoadDynamics.

VII. Conclusion

In this paper, we present the design of LoadDynamics, a generic workload predictor for cloud computing. The goal of LoadDynamics is to provide a generic and automated predictive technique to allow accurate predictions for the large variety of dynamically-changing workloads that user applications may experience on the cloud. To achieve this goal, LoadDynamics employs Long Short-term Memory (LSTM) models with automatically tuned hyperparameters, allowing it to build workload-specifically optimized predictors. We evaluated LoadDynamics with five real-world workload traces from four types of cloud applications. The evaluation results showed that the predictors built by LoadDynamics had low prediction errors for all workloads. The average prediction error of LoadDynamics is only 18%, which was considerably lower than state-of-the-art workload predictors. We also applied LoadDynamics to an auto-scaling policy in a real cloud. The LoadDynamics-enable auto-scaling reduced turnaround time by at most 38.1% than state-of-the-art predictors.

Acknowledgment

This work was partially supported by the National Science Foundation under grant CCF-1617390. The views and conclusions contained herein are those of the authors and should not be inter-preted as necessarily representing the official policies or endorse-ments, either expressed or implied of NSF. The authors would also like to thank the anonymous reviewers for their insightful comments.

References

[1] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems. In *ACM Symposium on Cloud Computing (SoCC)*, 2011.

[2] Timothy Wood, Ludmila Cherkasova, Kivanc M. Ozonat, and Prashant J. Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. In *ACM/IFIP/USENIX 9th International Middleware Conference (Middleware)*, 2011.

[3] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. CloudInsight: Utilizing a Council of Experts to Predict Future Cloud Application Workloads. In *IEEE Int'l Conf. on Cloud Computing (CLOUD)*, 2018.

[4] M. Mao and M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *IEEE Int'l Conf on Cloud Computing*, 2012.

[5] Supreeth Shastri and David Irwin. HotSpot: Automated Server Hopping in Cloud Spot Markets. In *ACM Symp. on Cloud Computing*, 2017.

[6] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling. In *IEEE Int'l Conference on Cloud Computing*, 2016.

[7] Charles Reiss, Alexey Tumanov, Gregory Ganger, Randy Katz, and Michael Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *ACM Symp. on Cloud Computing*, 2012.

[8] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2011.

[9] Erik-Jan van Baaren. Wikibench: A Distributed, Wikipedia-based Web Application Benchmark. *VU University Amsterdam*, 2009.

[10] Leslie N. Smith. A Disciplined Approach to Neural Network Hyper-parameters: Part 1 – Learning rate, Batch size, Momentum, and Weight decay, 2018.

[11] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Zexiang Mao, and Junliang Chen. Workload Predicting-Based Automatic Scaling in Service Clouds. In *IEEE International Conference on Cloud Computing (CLOUD)*, 2013.

[12] Hao Lin, Xin Qi, Shuo Yang, and Samuel P. Midkiff. Workload-Driven VM Consolidation in Cloud Data Center. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015.

[13] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Trans. on Computer Systems*, 30(4), 2012.

[14] Gueyoung Jung, Matti A. Hiltunen, Kaustubh R. Joshi, Richard D. Schlichting, and Calton Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[15] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Trans. on Cloud Computing*, 3(4), 2015.

[16] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting. In *IEEE International Conference on Cloud Computing*, 2011.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8), 1997.

[18] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR)*, 2018.

[19] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. In *A Field Guide to Dynamical Recurrent Networks*, 2001.

[20] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach. In *IEEE International Symposium on Network Operations and Management (NOMS)*, 2012.

[21] Sadeka Islam, Srikumar Venugopal, and Anna Liu. Evaluating the Impact of Fine-scale Burstiness on Cloud Elasticity. In *ACM Symposium on Cloud Computing (SoCC)*, 2015.

[22] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Int'l Conf. on Machine Learning*, 2015.

[23] Douglas M Hawkins. The Problem of Overfitting. *J. of Chemical Information and Computer Sciences*, 44(1), 2004.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. of Machine Learning Research*, 15(1), 2014.

[25] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 2012.

[26] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 1st edition, 2006.

[27] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. of Machine Learning Research*, 13(Feb):281–305, 2012.

[28] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554. 2011.

[29] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H.J. Epema. The Grid Workloads Archive. *Future Generation Computer Systems, 24(7)*, 2008.

[30] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource Central: Understanding and PredictingWorkloads for Improved Resource Management inLarge Cloud Platforms. In *ACM Symp. on Operating Systems Principles*, 2017.

[31] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. Wrangler: Predictable and Faster Jobs using Fewer Resources. In *ACM Symposium on Cloud Computing (SoCC)*, 2014.

[32] Eyal Zohar, Israel Cidon, and Osnat Mokryn. The Power of Prediction: Cloud Bandwidth and Cost Reduction. In *ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2011.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Int'l Conf. on Learning Representations*, 2015.

[34] Jonas Mockus. On bayesian methods for seeking the extremum and their application. In *7th IFIP Congress on Information Processing*, 1977.

[35] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[36] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2019.

[37] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *IFIP/IEEE Int'l Symp. on Integrated Network Management (IM)*, 2007.

[38] Haibo Mi, Huaimin Wang, Gang Yin, Yangfan Zhou, Dianxi Shi, and Lin Yuan. Online Self-reconfiguration with Performance Guarantee for Energy-efficient Large-scale Cloud Computing Data Centers. In *IEEE International Conference on Services Computing (SCC)*, 2010.

[39] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David E. Culler, and Randy H. Katz. NapSAC: Design and Implementation of a Power-Proportional Web Cluster. *ACM Computer Communication Review*, 41(1), 2011.

[40] Peter A. Dinda and David R. O'Hallaron. Host Load Prediction using Linear Models. *Cluster Computing*, 3(4), 2000.

[41] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *USENIX Symposium on Networked Systems Design & Implementation*, 2008.

[42] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *International Symposium on Quality of Service (IWQoS)*, 2003.

[43] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical Prediction models for Adaptive Resource Provisioning in the Cloud. *Future Generation Computer Systems*, 28(1), 2012.

[44] Peter Bodik, Rean Griffith, Charles A. Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[45] Chunhong Liu, Chuanchang Liu, Yanlei Shang, Shiping Chen, Bo Cheng, and Junliang Chen. An Adaptive Prediction Approach based on Workload Pattern Discrimination in the Cloud. *Journal of Network and Computer Applications*, 80, 2017.

[46] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2013.

[47] Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong N. Chang. ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning. In *IEEE Int'l Conf. on Data Mining*, 2011.

[48] Joao Loff and Joao Garcia. Vadara: Predictive Elasticity for Cloud Applications. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.

[49] Shuja ur Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, and David Carrera. Adaptive Prediction Models for Data Center Resources Utilization Estimation. *IEEE Trans. on Network and Service Management*, 2019.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521, 2015.

[51] Chanh Nguyen, Cristian Klein, and Erik Elmroth. Multivariate LSTM-Based Location-Aware Workload Prediction for Edge Data Centers. In *Int'l Symp. on Cluster, Cloud and Grid Computing*, 2019.

[52] Siddhant Kumar, Neha Muthiyan, Shaifu Gupta, Dileep A.D., and Aditya Nigam. Association Learning based Hybrid Model for Cloud Workload Prediction. In *Int'l Joint Conf. on Neural Networks*, 2018.

[53] Xiaoyong Tang. Large-Scale Computing Systems Workload Prediction Using Parallel Improved LSTM Neural Network. *IEEE Access*, 7, 2019.

[54] Jing Bi, Shuang Li, Haitao Yuan, Ziyan Zhao, and Haoyue Liu. Deep Neural Networks for Predicting Task Time Series in Cloud Computing Systems. In *IEEE Int'l Conf. on Networking, Sensing and Control*, 2019.

[55] Qingchen Zhang, Laurence T. Yang, Zheng Yan, Zhikui Chen, and Peng Li. An Efficient Deep Learning Model to PredictCloud Workload for Industry Informatics. *IEEE Trans. on Industrial Informatics*, 14, 2018.

[56] Hoang Minh Nguyen, Gaurav Kalra, and Daeyoung Kim. Host load prediction in cloud computing using Long Short-Term Memory Encoder–Decoder. *Journal of Super Computing*, 2019.