
Article:
Gyeera, T.W. orcid.org/0000-0002-2567-4197, Simons, A.J.H. orcid.org/0000-0002-5925-7148 and Stannett, M. orcid.org/0000-0002-2794-8614 (2022) Regression analysis of predictions and forecasts of cloud data center KPIs using the boosted decision tree algorithm. IEEE Transactions on Big Data. ISSN 2332-7790

https://doi.org/10.1109/tbdata.2022.3230649

Regression Analysis of Predictions and Forecasts of Cloud Data Center KPIs Using the Boosted Decision Tree Algorithm

Thomas Weripuo Gyeera IEEE member, Anthony J.H. Simons, and Mike Stannett
Department of Computer Science, University of Sheffield, United Kingdom

Abstract—Cloud data centers seek to optimize their provision of pooled CPU, bandwidth and storage resources. While over-provision is wasteful, under-provision may lead to violating Service Level Agreements (SLAs) with their consumers; yet the relationship between low-level Key Performance Indicators (KPIs) and SLA violations is not well understood. State-of-the art monitoring systems typically react to service failures after the fact, partly due to unexpected nonlinearities in the aggregated performance data. We seek to provide better modelling of KPIs using predictive algorithms that could be used for the proactive monitoring and adaptation of cloud services. In this paper, we investigate the Boosted Decision Tree (BDT) regression algorithm. We tested the BDT algorithm in a real monitoring framework deployed on a novel Azure cloud test-bed distributed over multiple geolocations, using thousands of robot-user requests to produce huge volumes of KPI data. The BDT algorithm achieved an R-Squared score of 0.9991 at the 0.2 learning rate. This closely predicted the KPI data and outperformed other approaches, such as Ordinary Least Squares and Stochastic Gradient Descent; and is a promising candidate for making short- and long-term predictions for cloud resource allocation.

Index Terms—Boosted decision tree, machine learning, deep learning, algorithms, computational modelling, virtual infrastructure network, and cloud computing

1 Introduction

Cloud service providers seek to offer timely and adequate services to their consumers. The burden falls on cloud data centers to optimize their provision of pooled CPU, bandwidth and data storage resources across all tenants of the cloud. Over-provision is wasteful, since resources lie idle; but under-provision leads to service degradation and failure. In order to attempt the right balance, cloud data centers employ monitoring mechanisms to measure CPU performance, bandwidth and storage consumption and to detect anomalies [1]. Mostly, this is at the level of Infrastructure-as-a-Service [2], and the Key Performance Indicator (KPI) statistics are used privately by the provider to understand the health of their system.

Cloud consumers are interested in different aspects, such as latency and response times, which are also affected by multi-tenant occupancy on the platform [3], [4]. However, the relationship between low-level resource management and high-level customer experience is poorly understood; and Service Level Agreements (SLAs) may frequently be violated without the consumer knowing. State-of-the-art monitoring often reacts to failures only after the fact. This is partly due to unexpected nonlinearities in the aggregated KPI data and weak predictive models. Understanding the different contributions to service degradation and failure is hard, due to the sheer volume of monitored KPI data (characterized by the '4Vs' of volume, velocity, variety and veracity [5], [6]). This is a kind of Big Data analysis problem, to be solved in real-time [7].

This motivated us to develop a real cloud test-bed that would allow us to read from a variety of push- and pull-notification systems, which at the limit can flood a network with data. It also allowed us to study different predictive models, such as the Boosted Decision Tree algorithm, and test its accuracy in modeling the collected KPI data.

Our goal is to move from reactive responses to proactive adaptation of services to prevent failure. The contributions of this paper include:

- A novel framework for measuring cloud resource consumption, via a set of monitored KPIs, implemented using a realistic cloud test-bed, driven by massively parallel robot users sending requests to a target web service.
- A set of supervised machine learning experiments, applying the Boosted Decision Tree algorithm to different KPIs, achieving an R-squared value of 0.9991 at a learning rate of 0.2, which compares favourably with state-of-the-art benchmark algorithms.
- An evaluation of the BDT's KPI predictions for short- and long-term planning, in particular, the ability to detect SLA violations in application response times and throughput.

The rest of the paper is organized as follows. We discuss previous work related to this investigation in section 2. Section 3.1 presents the conceptual framework and the relevant mathematical constructs and re-expresses the underlying problem in terms of the boosted decision tree (BDT) supervised machine learning algorithm. Section 4 covers the experimental test bed and describes the procedure and tools used in data sampling for building and evaluating the models for the predictive analytic framework. In section 5 we provide a critical review of the experimental results based on the boosted decision tree regression method used in predicting and forecasting cloud server KPIs for adaptation

purposes. In section 6 we evaluate and interpret the trained and tested models and compare our results with the state-of-the-art techniques in previous works. Potential applications, benefits and threats to the validity of our work are discussed in section 7. We summarize our main conclusions in section 8, where we also include a number of topics for further investigation.

2 Related work

We extend our discussions on the related work to cover critical analysis of some key machine learning algorithms that have been employed in solving the problem of workload prediction in cloud data centers in the literature. The boosted decision tree algorithm can be used in both classification and regression problem settings. The BDT algorithm can be employed in linear and nonlinear training set examples and for a training set with nonlinear properties, the algorithm does not require any additional steps in transforming the features. One key advantage of the BDT algorithm compared to other non-parametric (e.g. KNN) and classification algorithms is that it is fast, efficient and provides easy interpretation, understanding and visualization of a model.

Ardagna *et al.* [8] proposed the VM capacity allocation (CA) and load redirection (LR) reactive predictive models that dynamically adapt the resources of cloud infrastructure with the goal of optimizing the mean response times of clients' requests without violating consumer SLA parameters. In this two-phase framework, the CA model characterizes the complete number and properties of VMs that are required to handle the arrival of clients' requests per second without violating the average response time in the SLA document. The LR model on the other hand determines at every time the total execution rate of web service requests at a particular site and attempts to redirect workload from overburdened VMs which seem to violate the mean response time. The LR dual reactive scheme addresses an aggregated way of balancing workload in which detailed information about the mean response time of the incoming load is determined in order to project the optimal workload. The main drawback with this approach is that for highly distributed cloud systems, the response time predicted comes with a noticeable network communication overhead. This also leads to a noisier prediction model on the response time with the VM capacity allocation.

Aslanpour *et al.* [9] proposed an autonomic 3-D mechanism for provisioning cloud resource-aware, SLA-aware and user-behaviour-aware features. They applied the radial basis function (RBF) neural network to predict workload arrival rates projected for future demands. In the RBF implementation, only one hidden layer is used to approximate continuous function to a high degree of accuracy. The function takes the behavioral patterns of users as its input and determines the future load from it. In the resource behaviour analysis, the CPU utilization is sampled every minute from past observations. The weighted moving average (WMA) is then applied in order to increase the measurement accuracy. Experimental results indicate that the proposed mechanism is able to reduce resource provisioning costs by 50%, avoid rushed decisions, and stay resilient against extreme workload and that it is able to efficiently select surplus virtual machines.

Menascé *et al.* [10], [11] presented a model framework for predicting and comparing performance metrics on resources. Their work focused on the application of queuing theory formulae in building relations between the mean values of response times, throughput or resource utilization and the mean demand placed on the type of requests on the resource. Specifically, they designed experimental techniques in a controlled environment to measure these performance metrics (response time, throughput or resource utilization) in order to estimate the mean demand on the CPU utilization. A similar queuing network built by [12] outperforms the regression-based approximations characterizing the CPU utilization from consumer demands. Both frameworks can be used in estimating and profiling workload characteristics of individual virtual machines that have been provisioned in the cloud.

Kumar *et al.* [13] applied an artificial neural network and the adaptive differential evolution algorithm in predicting cloud data centers' workload. They performed experiments on some HTTP server benchmark datasets from NASA and the predictive models are seen to present an optimal mutation. For a given time series obtained from a historical dataset, the techniques of Autoregresssion Integrated Moving Average (ARIMA), Moving Average (MA), Autoregression (AR), Exponential Smoothing (ES), and Hidden Markov (HMM) models are known to perform quite well on historical sampled datasets with a uniform time interval.

Ban *et al.* [14] proposed the k–nearest neighbor (kNN) approach for making predictions on financial time series datasets. This algorithm was again applied by Eddahech *et al.* [15] to model predictions on multi–media workload fluctuations. kNN learners are generally considered lazy trainers and may give rise to high computational cost in the training phase. The combined techniques of neural network and linear regression were presented by Islam *et al.* [16] in predicting workload variations in data centers. The framework also described the sliding window concept and was tested on historical CPU demand data. Experimental results reveal that the sliding window performs better than the non-sliding window framework.

Tofighy *et al.* [17] proposed the Bayesian information criterion (BIC) ensemble algorithm for predicting CPU workloads. The proposed algorithm takes, as its inputs, traces from a smooth filter, smooth window and history window and maps these to the predicted resource usage in the output. In each time slot the algorithm selects the best model based on the usage history of the cloud resources. The resources are then monitored for a designated time period to generate a time series from the history of resource consumption which is fed into the prediction model. Both smoothing and fitting functions are applied for smoothing the data points for fitting the model. The proposed scheme outperformed the genetic algorithm (GA) and other state-of-the-art ensemble algorithms when evaluated using several statistical metrics such as the mean absolute error, root means square error etc.

Chauhan and Agrawal [18] proposed the optimized kernel naive Bayes approach, based on the concept of maxi-

TABLE I. Survey of studies related work in terms of evaluation tools, performance metrics, datasets, advantages, and disadvantages.

Evaluation Tools	Performance Metric	Datasets	Advantages	Disadvantages
Bayesian information criterion + smooth filters [17]	MAE, RMSE, MAPE,MASE	PlanetLab	Higher accuracy compared with the other ensemble prediction algorithms	The choice of priors can be a challenge.
Deep belief networks (DBN) [26]	MSE	Google cluster traces	Can solve supervised and unsupervised learning problems. It is a generative model.	Requires a large training set to flawlessly build a model.
AdaBoost algorithm [29]	MAE, MRE	Four real-world production servers	Fast and easy to program. Expandible to more complicated learning tasks. Can be combined with other machine learning algorithms	It is a weak classifier. Can lead to poor results and retraining. Vulnerable to evenly distributed noise.
Bayesian network [30]	MSE	Amazon EC2 and Google CE	Real world problems can be interpolated. Involves specifying a prior and integration.	Computationally infeasible. It turns out that specifying a prior is extremely difficult. For real-life inference problems it is often impossible to elicit the actual prior
Time series [31]	MSE	Axp0, Axp7, Sahara, and Themis load traces	Analysis helps in comparing present and future performance of the series. Insight into past behaviour can be gained to make future predictions	Trends within the series may be difficult to understand. Can lead to imperfect conclusions arising from differences in factors that influence the series.
Kalman Filtering Technique (KFT) [32]	MSE, MAE	Amazon EC2 and Google CE	Robust and works well in non-stationary environment	They are strictly Gaussian processes.
Long short-term memory (LSTM) [33]	MSE, MSSE	Axp0, Axp7, Sahara, and Themis load traces	Able to solve the vanishing gradient effects of backpropagation. LSTM units allow gradients to also flow unchanged.	They can still suffer from the gradient explosion problems problems.

mum probability for selecting cloud services. The model is able to predict cloud resources with maximum probability. A beta distribution of the model implements a data normalization model in order to handle inherent difficulties from the raw data set. It is suitable for handling bounded and asymmetric data and can be used to design recommended systems. Experimental results from predicting the response time, CPU and memory utilization achieve 88.76%, 88.445% and 93.65% accuracy, respectively.

Ghobaei-Arani and Shahidinejad [19] presented the GFA meta-heuristic-based algorithm for efficiently analyzing cloud workloads. They applied a combination of genetic algorithm (GA) and fuzzy C-means techniques for heterogeneous cloud workload clustering using the QoS metrics. In the initial phase, workload models are created by clustering incoming user requests, after filtering out abnormal requests deemed unsuitable for training. Similar clusters compared to the current user request can then be determined by comparing both the training and test workloads. In order to identify optimal scaling decisions for efficient resource provisioning, the gray wolf optimizer (GWO) metaheuristics technique is employed. Simulation results obtained under real workloads indicate that their proposed approach is efficient in terms of CPU utilization, elasticity, and response time predictions, as compared with other approaches.

Belgacem [20] presented an analysis for the problem of dynamic resource allocation in the cloud environment in which he extracted key research challenges in this area. The work offered several taxonomies for classifying dynamic resource allocation, scheduling approaches, and optimization metrics. The study revealed that metaheuristic methods provide efficient solutions in this problem domain.

Pahlevan *et al.* [21] integrated a novel hyper-heuristic and the K-means machine learning algorithm that dynamically and optimally allocates VMs to servers in cloud data centres. The sampled CPU utilization as well as the memory traces of the VMs are classified with the K-means machine learning algorithm in addition to a set of heuristics used to determine the VM classes. In the last step of the ML algorithm the VMs' patterns and features are extracted with the last-value predictor method and the reinforced learning technique determines which VM should be assigned to a particular class in the form of a finite set of actions and states of the virtual machines.

The K-means classification-based approach demonstrates an improvement of up to 24% in server-to-server network traffic. For large scale data centers, their approach is able to reduce workload execution time by 480 times compared to the state-of-the-art. In particular, when their approach is compared to correlation and network-aware [22] state-of-the-art schemes, the ML and Heuristic methods significantly improve the network communication overheads as the number of VMs provisioned increases. In comparison with the state-of-the-art, the application of the K-means technique helps achieve a reduction in violations in terms of server overutilization and network traffic overhead especially in VMs' off-peak loads management.

Our approach applies the boosted decision tree regression algorithm in building predictive models that can be used for analyzing and forecasting cloud resource allocation and consumption. We benchmarked and compared this approach with state-of-the-art shallow machine learning algorithms such as the ordinary least squares (OLS), Kalman filtering technique (KFT) and stochastic gradient descent (SGD) algorithms. In section 6 below, in order to demonstrate the benefits of our approach, we will compare it against state-of-the art methods that use reactive methods for planning capacity by distributed allocation and redirection [8], ensemble methods [17], genetic algorithm (GA) methods [18], and stochastic gradient descent [23], [24].

Deep learning algorithms are known to deliver top-quality results and come with their own preprocessing layers that allow application directly to unstructured data. The main drawback is that deep learning approaches require massive sets of data to flawlessly train a model. We compare our approach of using the BDT algorithm with deep learning approaches such as diffusion convolutional recurrent neural networks (DCRNN) [25], deep belief networks (DBN) [26], traditional neural networks (TNN) [27] and apolyadic canonical decomposition autoencoder models (CP-SAE) [28] in section 6. Table I provides a summary of key previous work related to our study in terms of the evaluation techniques, performance metrics, datasets, advantages and disadvantages.

3 Problem Definition

This section describes the boosted decision tree (BDT) linear regression algorithm as it is applied in making predictions on the server KPIs by building sub-functions within the hypothesis space of an inductive ensemble learning system. The problem of cloud resource provisioning and consumption prediction can be formulated in the language of least squares regression as having the overall aim of building

models that can make predictions on z-manifold observations. The goal of teaching the model to predict a server application KPIs like bandwidth fluctuations, latencies or %CPU utilization using a functional hypothesis space can be achieved through the minimization of the mean squared error.

3.1 Conceptual framework

Our monitoring and adaptation framework has four main building stacks as depicted in Fig. 1.

1: The *monitor stack* consists of a dashboard that showcases the different purposes for conducting monitoring: The pay-per-use monitor depicts metrics that may be relevant to how resources are consumed and how much the consumer may be required to pay for them. For example, the amount of memory, bandwidth or %CPU utilization can be used here as metrics for evaluating resource consumption, used for billing the client. For the purpose of enforcing an SLA contractual agreement, the SLA monitor here can display metrics such as the availability of resources provisioned within the cloud. The fail-over/infrastructure monitoring stack is used to charaterize transient and general network issues. This component is generally required to detect failures and anomalous behaviors, so they can be mitigated before the virtual network and the application server become unavailable. The interactions between the components of the conceptual framework are shown in the sequence diagram in Fig. 2.

2: The *adaptation stack* contains the filtering or machine learning algorithm that is implemented to learn from behavioral patterns displayed by the virtual infrastructure network and the KPIs of the application server. For instance, the implementation of an ensemble learning algorithm (BDT) helps predict future resource consumption patterns. In this case an adaptation strategy such as elastic load balancing, auto-scaling of pooled resources or the migration of a DB workflow can be enforced.

3: The third block of components consists of our application server and the virtual infrastructure network nodes to be monitored. We implemented a webservice platform that mimics eBay or Amazon, in the sense that it can allow a huge number of robot users to browse and make purchases from the application. The content management and the metrics polling techniques (push or pull) all constitute our application stack as shown in Fig. 1.

4: The fourth stack is the admin user console that is interfaced with the application in observing the different metrics of the framework. The admin terminal administers all the databases and storage required for the operation of the system as shown in Fig. 1.

3.2 The BDT algorithm

The *boosted decision tree* learning methods first derived by Friedman belong to an ensemble group of inductive learning methods [34], [35]. In these algorithms, the idea is to teach a model how to make predictions on a training set example by sequentially constructing a set of hypotheses in each iterative step. The individual hypotheses in the final set of functions or hypothesis space are combined in determining the predicted output [36], [37]. Compared to

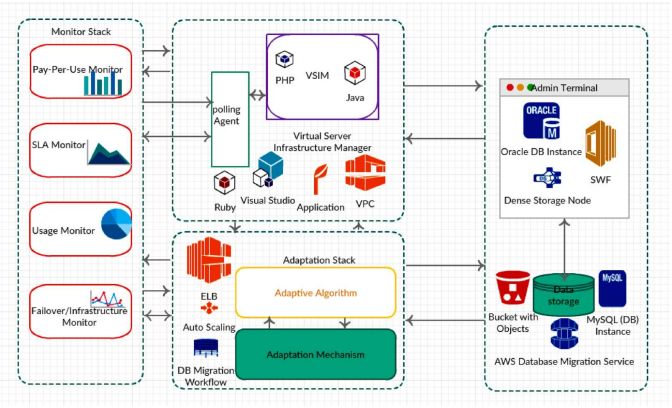


Figure 1. Block diagram of of the conceptual framework.

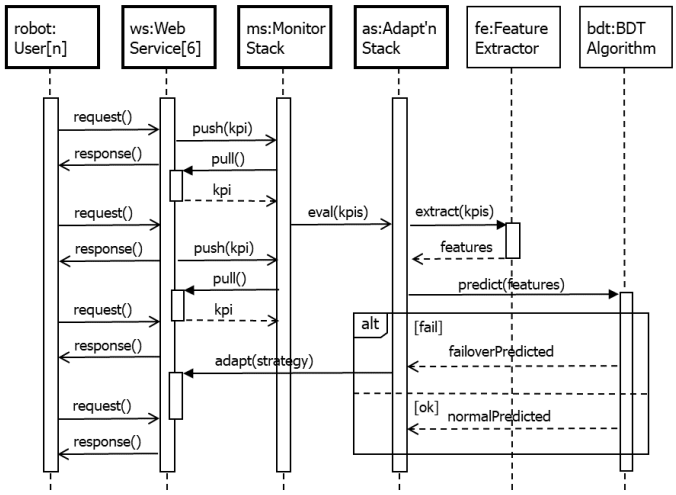


Figure 2. Sequence diagram showing the interactions of the components in the conceptual framework.

other training algorithms, BDT is fast, efficient and requires less effort in training unstructured data; it is seen here as a good choice for our problem definition.

Consider a finite hypothesis space M. For each iteration, a newly constructed function at the time interval is weighted to determine its contribution to the output value of the final hypothesis. Assigning weights to each hypothesis added to the predictive model is called *boosting* [35], [38] (e.g. boosting is normally initialized with a constant value greater than 0). In the next iteration, these weights are increased for results with weaker predictions while the hypotheses with the correct results have their weights decreased. These iterations continue until the final number of functions are generated.

Assuming a training set $\{x_i, z_i\}_{i=1}^{N}$ is obtained by measuring the server KPIs of a virtual infrastructure network or an application server provisioned in the cloud, the goal here is to teach a model with the boosted decision tree base-learner to make predictions on the input variable. For instance, if the variable of interest is the bandwidth fluctuation, or the percentage CPU utilisation, then data sets of these kinds can be used to train the decision tree model objective function, to make predictions on these KPIs. The

rest of this section presents the mathematical background behind the construction of the boosted decision tree as a predictive model according to [34], [38].

Let the hypothesis space be defined as one comprising the functions generated at each iteration on the training set example $\{x_i, z_i\}_{i=1}^{N}$: the parameters for building the functional space include the tree structure, score labels for each leaf and the total number of trees to be generated. Since each step generates its own subtree (also referred to as a *sub-hypothesis*), the objective function constitutes the sum of all the different hypotheses as defined in equation (1):

$$H = \{h_1, h_2, h_3, ...h_T\} \tag{1}$$

where T is the total number of trees and $h_1, h_2, ...h_T$ are the sub-hypotheses added to the hypothesis space during each running of the training experiment. The boosted decision tree can be constructed from a logical expression by forming the disjunction of all the sub-hypotheses as shown in equation (2) (see, e.g., [35]).

$$H \Leftrightarrow (h_1 \vee h_2 \vee h_3 \vee \cdots \vee h_T) \tag{2}$$

If the decision tree can be constructed through the combination of the sub-hypotheses in (2), the next steps illustrate how each of these sub-functions can be constructed to fill the defined hypothesis space H.

In building the regression tree from sampled observations on a server's KPI at different time intervals, one immediate step is to construct the loss function that minimizes the least-squares error. Thus the loss on making inaccurate predictions on the training set data is the sum of the residuals between the predicted and the actual observations, given as follows:

$$Loss_{(m)} = \sum_{j=1}^{m} Loss(z_j, \hat{z}_j) \tag{3}$$

This error function measures how well the model fits on the training set. According to the formulation of the least square error, if we take this to be the minimization objective function, then the squared loss is given as follows:

$$Loss = (z_j - \hat{z}_j)^2 \tag{4}$$

In order to characterize the complexity of the new model, we define an additive L_2 regularization hyperparameter (see [38], [39] for the definition of the L_2 regression) as a component of our objective function. This component includes a learning rate factor, λ, which ranges from $0 < \lambda \leq 1$ and can be used to tune the model. As a rule of thumb, the model learns faster, the closer the learning rate is to unity, but according to [37], this parameter should be tweaked carefully to avoid overfitting. For the j^{th} training set $\{x_j, z_j\}$, the estimate of $\hat{z}_j$ at sample interval j is the sum of all the hypotheses generated from the previous estimates. These estimates in the hypothesis space form the decision trees base stumps which are evaluated on *if-then* clauses in building the tree.

If a total of T regression trees is desired, then the model function can be summarized (equation (5)) as the complete objective function. The model characterized by the sum of the heuristic hypotheses can be split from the base-learner into sub-trees consisting of the sub-functions within the ensemble defined in (1) by applying the information gain (see definition in [34], [38], [39]). The complete objective function can be stated as follows:

$$Loss = \sum_{j=1}^{m} Loss(z_j, \hat{z}_j) + \sum_{T} \Psi(h_T) \tag{5}$$

where $\sum_T \Psi(h_T)$ defines how complex the tree can be ($h_T \in H$) summing all sub-hypotheses at the different iterations to generate the ensemble, and T is the total number of trees to be grown from the sub-hypotheses. Once the objective function has been completely stated, a prediction at a defined period is simply the sum of the estimation at the previous and current iterations. This can be expressed mathematically as

$$\hat{z}_m = \hat{z}_{j(m-1)} + h_m(x_j) \tag{6}$$

where $h_m(x_j)$ is the new function added to the growing tree at the m^{th} round of running the training experiment on the model in predicting the values of z. It is this addition of $h_m(x_j)$ to the training loss function that is referred to as boosting [34], [38]. Applying the steep gradient approach, the sum of squares loss can be reconstructed from (4). If the first and second partial derivatives are applied to equation (4), then this leads to the following equation. If we consider the total loss by taking the difference between the actual and the predicted estimate as shown in (4), then the new loss error square function can be rewritten as in (7):

$$Loss = (z_j - (\hat{z}_{j(m-1)} + h_m(x_j)))^2 + \Psi(h_m(x_j)) + C \tag{7}$$

where C is a constant. The goal of combining equation (4), (5), and (6) to form equation (7) is to allow the objective function to be derived by applying the gradient descent method. To this end, applying the first and second partial derivatives to equation (7) and using the Taylor expansion series for linearizing polynomials, the error square objective function can be rewritten as:

$$Loss = \sum_{j=1}^{m} [\Delta h_m(x_j) + \frac{1}{2}\Delta^2 h_m^2(x_j)] + \Psi(h_m) \tag{8}$$

where Δ is partial derivative taken on the loss square error function with respect to the previous estimates $\hat{z}^{(m-1)}$.

$$\Delta = \partial_{\hat{z}^{(m-1)}} Loss(z_j, \hat{z}_j) \tag{9}$$

$$\Delta^2 = \partial^2_{\hat{z}^{(m-1)}} Loss(z_j, \hat{z}_j) \tag{10}$$

The component $\Psi(h_m)$, called the regularization parameter, characterizes the complexity of the model. There are several regularization methods in the literature (see [34], [38], [39] for further reading) that can be employed for measuring the complexity of a decision tree model. For this article, the L_2 norm, which is defined as the standard Euclidean distance, is used in combination with the learning rate to define the structure of the tree. Thus the model complexity $\Psi(h_m)$ contains the total number of leaves and the L_2 sums the optimal weights within the ensemble. This formulation can be expressed as:

$$\Psi(h_m) = \lambda \|\omega_m\|^2 + \gamma L \tag{11}$$

where ω_m is the optimal weight assigned to each leaf in a particular round of the training experiment and γL is the

total number of leaves from the tree. Algorithm 1 below presents the pseudo code for the boosted decision tree algorithm (following [34]).

3.3 Pseudocode and time complexity analysis

Algorithm 1 shows the steps in the application of the BDT algorithm in making a prediction to a cloud server KPI. The algorithm accepts as input, the server KPI metric as a training set example (e.g. server throughput KPI). By defining a finite set of hypotheses, the algorithm constructs an error function and the base-learner. The read stage of the algorithm computes the negative gradient and includes this to the hypothesis space. The gradient descent is optimized to minimize the square of the loss error objective function.

In terms of time complexity analysis, the boosted decision tree (BDT) algorithm has $O(N * D^2)$ running time, where D is the number of features. For each level of the decision tree it has $O(ND)$ running time when constructing each function in a hypothesis space with D features and a total number of N hypotheses.

Algorithm 1 The Boosted Decision Tree Algorithm

Inputs: the sampled server KPIs, $\{x_i, z_i\}_{i=1}^{N}$; the loss function, $Loss(z_i, \hat{z}_i)$; and the number of trees, m

procedure BDT($\{x_i, z_i\}, Loss(z_i, \hat{z}_i), m$)
  Initialize the model with a constant value:
    $f_0(x) = \arg\min_{\hat{z}} \sum_{i=1}^{N} Loss(z_i, \hat{z}_i)$
  read current state
  for $i \leftarrow 1, m$ **do**
    Compute the pseudo-residuals $for\ i = 1, ..., N$
      (the negative gradient is optimized):
      $residual_{im} = -\left[\frac{\partial Loss(z, f(x_i))}{\partial f(x_i)}\right]$
      Construct the base-learner: $\hat{h}_m \leftarrow constant$
      with $\{x_i, residual_{im}\}_{i=1}^{N}$ as new inputs
      Compute the multiplier $\hat{z}_m$ to optimize the
      following one-dimensional problem:
      $\hat{z}_m = \arg\min_{\hat{z}} \sum_{i=1}^{m} Loss(z_i, f_{m-1}(x_i)) + \hat{z}_i h_m(x_i)$
    Update the model with the estimate:
      $f_m(x) \leftarrow f_{m-1}(x) + \hat{z}_i h_m(x_i)$
  end for
  return Output $f_m(x)$
end procedure

4 Experimental design

This section describes the various experimental methods and tools set up in order to critically evaluate our conceptual framework. This is a real experimental testbed designed in the Microsoft Azure cloud with resources distributed across different geolocations.

We designed and implemented a webservice platform (*http://mytwg.azurewebsites.net*) that mimics a shopping application like Amazon or eBay, designed to accept large numbers of hits from simulated virtual users. In Azure we provisioned six logically separate servers, running in three distinct geographical regions (US East Coast, West Coast, and Europe), and migrated the application to each of these servers, to provide a high availability service with a failover mechanism. For instance, if the main site being remotely hosted in these data centers is shut down for any reason, a load-balancer can redirect traffic to one of the six replicated servers. These six servers are networked to communicate with each other with one server controlling the entire domain of the virtual infrastructure.

We then programmed robot virtual users using the JMeter server tool to distribute concurrent virtual users which are driven by scripts under experimental control, and scheduled them to execute concurrently as clients of the sales application. The users come in the form of Java threads that are programmed to send requests to a server. We also provisioned a generic load balancer in Azure that regulates the network and user traffic that are directed to the application servers.

In addition to the virtual infrastructure network, the web service platform is interfaced with Azure application Insights [40] in order to monitor live server KPIs as the experiment is being run. Instrumenting Google Analytics [41] with the applications also allows the measure of metrics on the remote procedure calls, user behaviour and navigation patterns as they open sessions to the web platform. The techniques of load balancing ensure that users requests are uniformly distributed on the server without overburdening a particular resource. These resources implemented within the Azure cloud form the real testbed in order to measure the key performance indicators characterizing the virtual infrastructure network and the application servers.

Below is a description of the steps taken when conducting the experiments on the virtual infrastructure network and the application server of the web service platform.

4.1 The experiments

For the purpose of our data collection in building our predictive framework with the boosted decision tree algorithm, we performed four main experiments simulating different user scenarios on the webservice platform. The results of these experiments are shown in Fig. 3.

We aligned the streaming of the workload characteristics of the virtual machines provisioned in Azure to be homogeneous in terms of CPU and RAM, and adopt the approach in [8], [42]. A homogenous workload influx assumed here can also be applied to heterogenous VMs and cloud resources. In a co-located environment such as Azure, multiple VMs do run in parallel and in cases of high resource demands, workload is evenly distributed among multiple virtual machines [8].

In experiment One, the server was immediately loaded with a high number, N, of virtual users from the start of the experiment. The ramp-up period was set to zero, in which case an idle time was set for the server to prepare and process the load influx. The experiment was run for 180 minutes before decreasing the load to zero. For this experiment we employed the standard JMeter thread sampler for the load distribution onto the server.

In experiment Two, the objective was to simulate a real user behavior by employing the concepts of pacing and think time. In a real world web platform, users do not usually execute all their actions at once (in quick succession) when browsing a web application but there is usually some delay in the series of activities referred to as the user "think time". To simulate the behavior of the virtual users in a

realistic manner, the stepping thread group in JMeter was employed. The parameter settings for this experiment can be described as follows:

A maximum constant number, N, of virtual users was set on the thread group with a delay of 65 seconds before starting the threads on the application server. After the 65 seconds the experiment began by adding a constant number of threads every 20 seconds with first a ramp-period of 5 seconds. After reaching the maximum load of N virtual users, then the server was programmed to delay the execution of the load for five seconds (the server was kept idle for this period of time). The application then allowed the virtual users to browse the platform for 60 minutes and then it started decreasing the load by 10 virtual users every second.

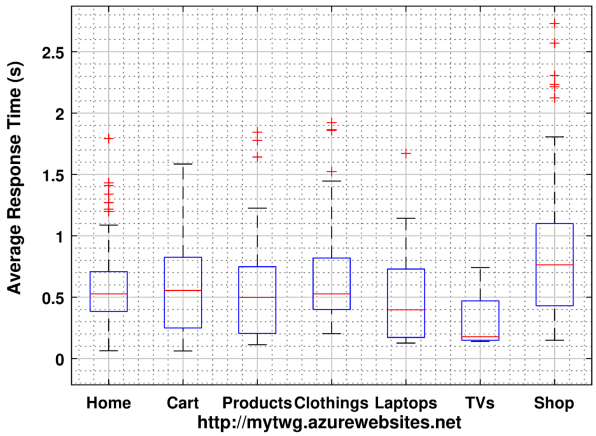
In Experiment Three, instead of defining a constant maximum load influx, the uniform random timer in JMeter was used so that a random number of virtual users could be added on the application server. This experiment was intended to depict a real-world scenario in which one does not know the rate of user load influx on the application server. It was also intended to show how a server responds to highly randomized user activities.

Experiment Four combined both a constant load influx for a period of time and then the load was decreased to a minimum value and then a random number of load influx was added again. Thus we increased the load and decreased the maximum load randomly to near zero and then increased the load for a constant load before bringing the experiment to an end. The main goal with this experiment is to show a mixture of random and a predetermined user behavior – for instance, a server may be designated to process a constant workload but for one reason or another some additional load can be redirected to this server in a random manner. For this experiment, we employed the ultimate thread groups (see [43] for more on JMeter ultimate thread group) from JMeter in distributing the load across the server.

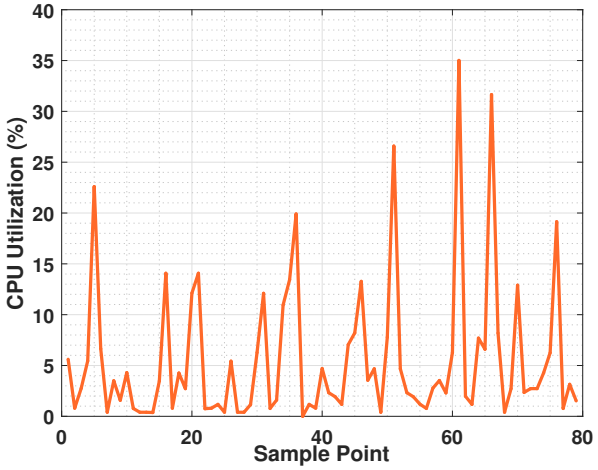
These four main experiments were conducted repeatedly for four weeks to generate and measure the average values of the key performance indicators characterizing the application server and the virtual infrastructure network. In addition to the response times from these experiments, the remaining server KPIs measured included the percentage of CPU utilization, latency, bandwidth fluctuation, server hits per second, and throughput (requests per second) for building the predictive models.

These KPIs are of fundamental importance in cloud data center management especially in designing SLAs, billing, and load admission control. The work presented in [3] describes how consumers use these KPIs when selecting a suitable cloud service provider.

Fig. 3a shows the aggregate results of the average response times presented as box plots on the main experiments conducted for the data collection. As shown in Fig. 3a sampling of the TVs page has the lowest mean and median response time indicated by the red line on the box plots. The shop page has high response times, with a larger number of outliers (indicated by red plus) in the composite plot of Fig. 3. The rest of the pages indicate similar mean response times as shown in the remaining six sampled sites. The



(a) The sampled average response times of the application server under normal workload ($vU < 1000$) fluctuation at different sites.



(b) The sampled % CPU utilization under normal workload ($vU < 1000$) fluctuation.

Figure 3. Experimental results for the average response times and % CPU utilization of the application server and the virtual infrastructure network under normal load ($vU < 1000$) influx.

mean response times from all the sampled sites indicate an overall predictable average response except for the few outliers highlighting some potential volatilities under a normal workload characteristic. We defer a discussion of the high number of outliers sampled at the shop site and some of the sampling sites until section 7.1.

Fig. 3b shows the average % CPU utilization that is quite predictable with the mean and maximum utilization of 5.01% and 35.01% respectively. These percentage utilizations correspond to a normal workload influx simulated with slightly more than 1000 virtual users. The maximum utilization of 35.01% is a little above the normal CPU characterization (20%) under normal data center workload as reported in [44]. The volatilities shown in the response time plots (indicated in the outliers) are in line with the % CPU utilization and we present a detailed case study on the root cause of the high degradation of the application performance (see section 7.1).

4.2 Data collection procedure

We rolled out live the web application designed for this experiment on the Microsoft Azure cloud hosting environment for virtual users (vU) to interact with the dynamic contents of the application. Clients are required to generate events through the web pages (e.g. by browsing and clicking on a product image or a button) to create an HTTP request object. Simulating client-server activities with a large number of real users simultaneously interacting with the application poses a huge challenge since it is difficult to attract large number of real users on the website over a specified time period.

To simulate realistic user loads, we used the JMeter client-server emulator, which is capable of generating thousands of HTTP requests, something that would not be possible with human test subjects. For a system with 8 GB memory and 4 vCPU cores, JMeter can generate as many as 1000 virtual users per server. Using JMeter, we simulated the number of virtual users with Java thread samplers that allow concurrent users to browse the web application according to the four experiments described in section 4.1. The JMeter samplers send the HTTP requests to the web page or server and the ramp-up period determines the frequency with which each virtual user accesses a particular page of the application (e.g. 10 vU configured on a ramp-up time of 10 seconds means that JMeter has 10 seconds to get all 10 vU threads up and running. Each thread has access to the server 1 second (10 vU/10) after the previous thread executed its requests). In order to guarantee that we obtain accurate measurements of the metrics we are investigating, we subjected the web application to a prior functional testing regime.

The main task then is to incrementally load all our application servers concurrently with 1000s of HTTP requests and then measure the load-capacity and server performance metrics. Setting up our virtual infrastructure network and the web application with these server configuration parameters allows us to monitor and collect the data needed for building our predictive models.

For instance, based on our simulated data, we want to address such research questions as:

1) *Can we realistically overload the shopping application with users until we see different resource-bounded failure modes?*
2) *Can we train predictive models that react in time before failure occurs?*
3) *Can we provision alternative resources within the time window given by the predictions?*

These are some of the questions we desire to concretely answer and discuss through the experiments described in the previous sections. The next section presents the critical analyses and the evaluation of the machine learning model built with the boosted decision tree algorithm.

5 Critical Analysis and Evaluation

From the experiments described in sections 4.1 and 4.2, we sampled data on the key performance indicators characterizing the web service application server and the virtual infrastructure network, including the server hits per second, latencies/response time, throughput, CPU utilization, and bandwidth consumption. Each dataset was saved as a CSV file that would serve as the input for performing the machine learning experiment.

In Azure machine learning [44], the basic program unit for performing an experiment on a category of dataset is known as a module. Each module is bounded by both input and output ports that enable the flow of information from one module to the next during processing.

The fundamental model features selected from the dataset consist of the number of virtual users (simulated here as the JMeter thread groups), the average response time (which has a direct linear correlation with the latencies), CPU utilization, average throughput, bandwidth, and memory consumption. By training the boosted decision tree regression model using the Azure machine learning studio, the goal here is to be able to produce predictive analyses on these features (server and VIN KPIs).

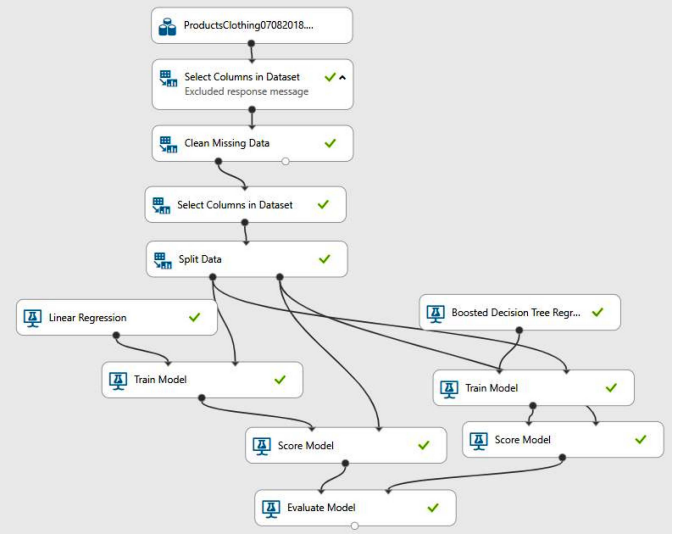


Figure 4. These flow diagrams illustrate the machine learning experiments conducted with the boosted decision tree algorithm.

We applied the data cleaning module, available in the experimental environment (the Azure machine learning experiment canvas), after uploading the sampled data in order to clean columns and rows with missing and redundant data. We also applied data manipulation queries to transform and clean the raw data into a suitable format that would increase prediction accuracy.

5.1 Inputs and outputs in the prediction tasks

Fig. 4 illustrates the machine learning experiments conducted in Azure to train the models with the boosted decision tree regression algorithm. The figure shows the initial settings and all the modules selected for the complete machine learning experiments for training the dataset. The validation experiment is conducted in a similar fashion, where the results of the entire experiment are converted into a webservice output platform for external users to consume.

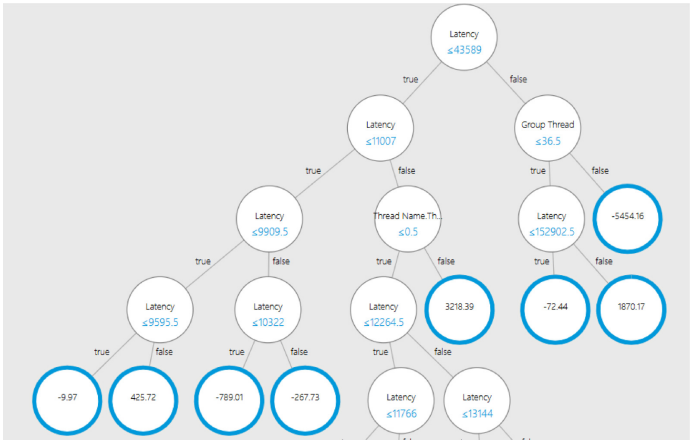
The inputs into the components of the framework are the server KPI metrics sampled at constant time intervals

from the monitoring components which include the mean response time, number of requests per second, percentage CPU utilization, bandwidth, throughput and server hits per second. The output of the component is the predicted value for the cloud service that is offered to the provider. The predicted value is then used as an input into the adaptation phase (e.g horizontal or vertical scaling, load balancing or admission control).

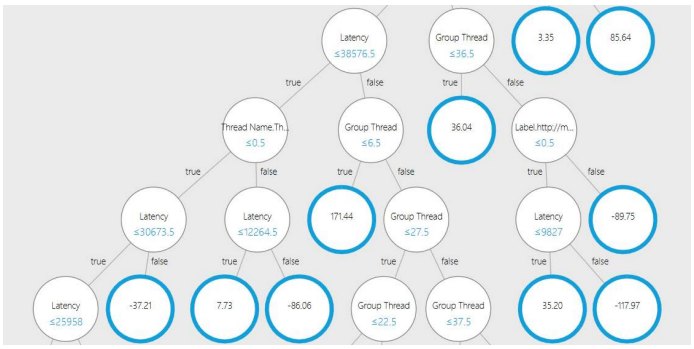
To illustrate the construction of the tree with the given inputs, as presented in Algorithm 1, the prediction tasks initially take the server KPIs metric at different time intervals, the loss function and the number of trees as inputs. We then define a base model with the average of the KPI metric as the output to the base model. The first order derivatives (gradients) of the loss function are used to minimize its value. After fitting the base model, we then use the predicted KPI metric again as the input and the output will be the residual. These steps are repeated sequentially together with the learning rate as a multiplicative factor in adding a new subtree until the final value is predicted.

We designed the boosted decision tree regression experiment using a single model parameter for the training mode with a fixed learning rate of 1.0. We then experimented with 20-maximum leaves on each tree and a default value of 10 samples per node leading to a total construction of 100 trees per leaf on each run of the experiment. We did not use a random seed number and the webservice parameters were configured to average on the final hypothesis on the evaluated model. Fig. 5 shows the 1^{st} and 100^{th} iterations of the sub-hypotheses in filling the hypothesis space. The iteration trees depicted in these figures are sub-hypotheses of the hypothesis space illustrated in equation (1). As described in section 3.1, for example, the 100^{th} iteration tree shown in Fig. 5b represents the new function $h_m(x_j)$ that is added to the growing tree depicted by equation (6) at $j = 100$. The 100^{th} iteration of the decision trees indicates the final hypothesis constructed for the hypothesis spaces of 100 trees at $j = 100$ as shown Fig. 5.

Further, as explained in section 3.1, for the design of the boosted decision tree model we set out a finite hypothesis space ($M = 100$) that averages over the final hypothesis space as indicated in equations (1) and (2). Our parameter λ is 0.001 for the L_2 regulation with a maximum learning rate of 1.0. Both training and validation datasets predict quite well after a learning rate of 0.2 and there is not much variation in the performance of the model as we further increase the learning rate with the same λ parameter tuning. Tuning the hyperparameters of the model from equations (7) and (11) gives the final results of the model with the boosted decision tree algorithm. Fig. 5a and Fig. 5b show the iterative steps (from 1 to 100) of the different trees generated within the functional hypothesis defined in equation (2) but we are not able to display all of them.



(a) 1^{st} iteration tree.



(b) 100^{th} iteration tree.

Figure 5. The figures illustrate the constructed trees at the 1^{st} and 100^{th} stages of the iteration with the BDT algorithm.

6 Model Evaluation and Interpretation

To evaluate the performance of the models built with the boosted decision tree algorithms, we measured the mean absolute error (MAE), root mean squared error (RMSE), absolute error (RAE), and relative squared error (RSE), the standard statistical metrics suitable for describing the performance of a model for regression [45].

The results are presented in Table II. Averaging the MAE values at their respective learning rates gives a result of 177.29 which is less than 180 for a model to considered a good one [46]. From the results in Table II, we recorded the best R-squared value at a learning rate of 0.2 and as we further increased the learning rate, there was not much variation in the coefficient of determination (CoD) values.

TABLE II. Statistics indicating the performance of the boosted decision tree algorithm.

	Learning Rate						
Model metric	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Mean Absolute Error	145.35	154.34	167.12	185.70	185.70	200.60	202.23
Root Mean Squared Error	496.91	474.08	476.15	523.89	523.894452	545.69	542.14
Relative Absolute Error	0.020917	0.02221	0.024049	0.026723	0.026723	0.028866	0.029102
Relative Squared Error	0.000907	0.000825	0.000833	0.001008	0.001008	0.001094	0.001079
Coefficient of Determination	0.999093	0.999175	0.999167	0.998992	0.998992	0.998906	0.998921

The results in Table II clearly indicate that the chosen algorithm completely replicates the training and testing dataset in the machine learning experiments. Similar results (e.g. dataset on bandwidth, throughput, and server hits etc.) are achieved with the application of the algorithm to the other server KPIs mentioned in this paper but for the sake of brevity, their plots are not included. We can generalize our results and state that the boosted decision tree regression is a suitable algorithm for predicting cloud server KPIs on cloud application servers and the virtual infrastructure networks.

Our comparisons in the next sections demonstrate the superiority of the BDT algorithm over some state-of-the-art shallow and deep learning algorithms. The BDT used here as a base learner has the desirable capability that it can both linear and nonlinear effects which cannot be captured by linear techniques such as the Kalman filtering technique (KFT), Bayesian information criterion (BIC), and ordinary least squares (OLS). Also the synergy of non-linear and threshold effects among inputs can best be handled inherently in the construction of the BDT algorithm in comparison to the state-of-the-art deep learning approaches.

6.1 Comparison with the state-of-the-art shallow machine learning algorithms

We benchmark and compare the application of the BDT approach to two standard state-of-the-art models [23], [24], [45] : the ordinary least square linear regression (OLS) and the non-linear stochastic gradient (SGD) algorithms in making predictions on the cloud server KPIs measured in section 4. For this purpose, we followed the approach in [23] with the L2 regularization.

In order to compare our results to these standard ML techniques we ran extensive machine learning experiments with the non-linear stochastic gradient descent from the range 0 to 1 learning rate. We achieved 0.33 as the best coefficient of determination on both training and test results at a learning rate of 0.7 and any further increase of the learning rate results in a continuous decay of the predicted signal. Based on the mean absolute error and the coefficient of determination, the SGD performs poorly compared to the OLS and the BDT training algorithm. In addition to the metrics shown in Table III, the SGD requires a longer training time versus accuracy compared to the OLS and BDT in attaining its best training and testing results.

Running the experiments with the ordinary least squares method improves the model accuracy of prediction with a 0.9989 coefficient of determination at the final learning rate of 1 as shown in Table III. Comparing standard OLS and BDT, both algorithms achieve convergence on the models at a very low learning rate with little variation in model performance as we further increase the learning rates. BDT achieves its best coefficient of determination 0.9991 at a learning rate of 0.3 while the ordinary least squares achieves its best coefficient of determination at a learning rate of 1.0 to closely match the BDT results.

In addition to comparing the performance of the BDT regression algorithm to the two standard state-of-the-art machine learning algorithms (SGD and OLS) we further compare this approach with the reactive framework presented in [8]. Their experimental results indicate that a maximum percentage error on the average response time estimation is less than 20%. Specifically, their approach is able to provide an accuracy prediction quality that in terms of the mean square error is always less than 10%. The VM cost optimization technique integrated in this framework is one of its main advantages. The mean percentage error of our approach varies between 2% to 5% even at a very low learning rate using the boosted decision tree regression technique as against the total of 10% prediction quality mean square error achieved in the work from [8]. The BDT regression algorithm described in this paper outperforms the adaptive method presented by Ardagna *et al.* [8] in terms of the mean percentage-errors of less than 5% on both the training and testing examples. The models trained and tested achieved a predictive accuracy of 98% and BDT outperforms the well-known state-of-the-art ordinary least squares (OLS) and the stochastic gradient descent algorithms (SGD).

TABLE III. This table compares the metrics of the three regression algorithms.

Model Algorithm	MAE	RMSE	RAE	RSE	CoD	L2
Boosted Decision Tree	177.29	569.70	0.032033	0.001192	0.998808	0.001
Ordinary Least Squares	192.09	700.29	0.013252	0.000109	0.989145	0.001
Stochastic Gradient Descent	6057.18	16336.04	0.871653	0.980027	0.332885	0.001

Further to the evaluation of our model we compared its performance with other shallow learning machine algorithms. Specifically we compare our work with the approaches using the Bayesian information criterion (BIC) [17] and the Kalman filtering technique (KFT) [32]. BDT outperforms BIC in terms of average prediction accuracy. The performance of the OLS just beats our BDT approach in the RAE and RSE metrics, but BDT is more accurate overall. Fig. 6a shows a comparison of the different state-of-the-art shallow machine learning algorithms with the BDT algorithm.

6.2 Comparison with deep learning algorithms

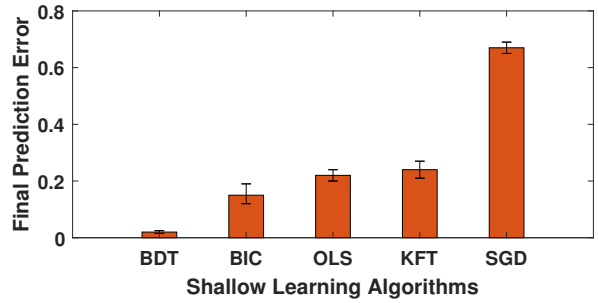
In addition, we compare our approach with deep learning algorithms to further evaluate the benefits of the boosted decision tree algorithm in such a non-linear environment. Specifically we compared our approach with the diffusion convolutional recurrent neural networks (DCRNN) [25], the deep belief neural networks (DBN) [26], the traditional neural networks (TNN) presented in [27] and the apolyadic canonical decomposition autoencoder model (CP-SAE) [28]. As shown in Fig. 6b, the DCRNN outperforms both the DBN and the TNN but its performance falls a little below that of the BDT algorithm.

6.3 Comparison with genetic and ensemble algorithms

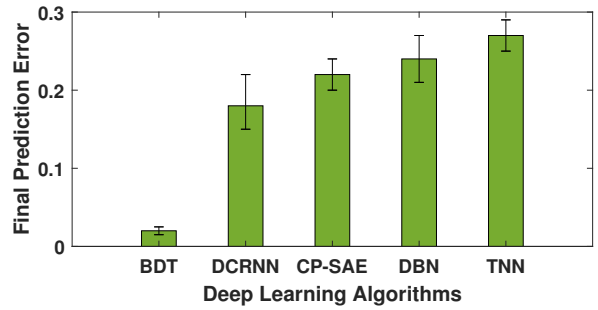
Finally we evaluated our work against state-of-the-art ensemble (EN) and genetic algorithms (GA) as applied in cloud resource prediction. Comparing the proposed model with previous work in [17], as shown in Fig. 6c, our approach outperforms both GA and EN in final prediction accuracy.

6.4 SLA violations and mitigation

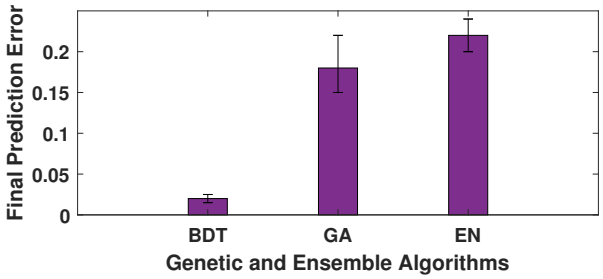
We demonstrate in this section, taking as an example the setting of a service level objective (SLO), how SLA violations can be detected by the proposed framework using real world datasets of sampled response times and server throughput. As a service level objective we set for our online shopping service application an average response time of 1 second. Other previous approaches have set a mean response time of 200ms to a maximum of 1200 seconds. Our evaluation here seeks to demonstrate the detection and violation of SLA through the proposed monitoring framework. For our web application we set a total of 470



(a) Comparison with shallow learning algorithms.



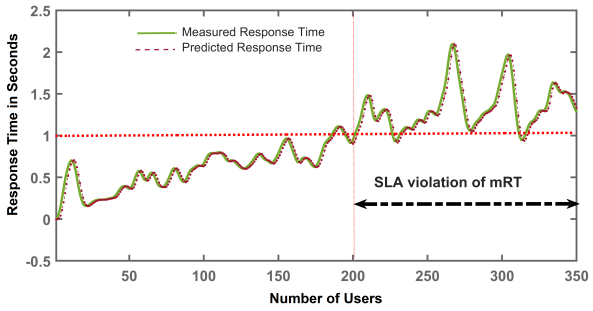
(b) Comparison with deep learning algorithms.



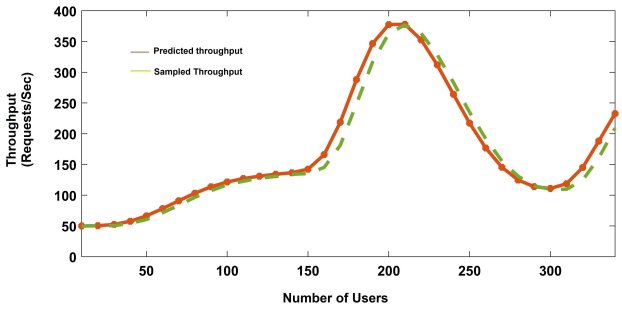
(c) Comparison with genetic and ensemble algorithms.

Figure 6. Comparison of the different algorithms with BDT in terms of performance degradation (smaller values mean better performance).

workload. The server throughput begins to degrade at the same period that the response time was violating our set target of 1 second. The monitoring stack described in Fig. 1 simply automates the process of a non-disruptive service by scaling out a cloud resource or live migrating the application to an idle or a less busy server. This process leads to a settling of the response time back to either below or within the mean response time with the throughput starting to increase.



(a) Performance of the server response time.



(b) Performance of the server throughput.

Figure 7. Prediction of the response time and server throughput.

pages per second as a throughput service level objective. Consider a typical HTTP GET request to the web server of about 256 bytes of data, running on 100 Mbps Ethernet. Meeting this throughput SLA, when transmitting typical 200kb pages using a standard TCP/IP protocol (180 Bytes), amounts to moving 470 pages every second (calculated as: $100Mbps/214292\ bits\ per\ page$).

To evaluate the SLA violations we streamed workload on our application service and monitored the server throughput and the performance of the mean response time. From Fig.7a, the response seems quite predictable as it increases linearly with the streaming of users sending requests to the server to be processed.

However after 200 seconds, the point at which 200 users were issuing concurrent requests, a continuous degradation of server performance with a doubling of the mean response time was observed. This suggests congestion within the server due to our aggressive transmission of workload onto the server. At the same time, monitoring the server throughput as shown in Fig.7b, indicates that the server throughput was pretty good before the 200 seconds of processing the

7 Potential Applications and Threats to Validity

We seek to provide a framework for monitoring and predicting service performance in the cloud. The related benefits are different for service consumers and service providers. A service consumer may wish to conduct a comprehensive analysis of Service-Level Agreements, to see under what conditions any of the Quality of Service (QoS) promises may be broken. A service provider may wish to gain *a priori* knowledge about how their applications are likely to perform when deployed in a given cloud environment, before the service is offered live, so they can make informed choices about storage and bandwidth offered by the cloud host. Finally, the cloud host or broker may operate the predictive engine in real time, in order to detect when failure is imminent and so take suitable corrective action.

The work presented in [47] details the importance of the metrics we have used in our experiments in comparing different cloud service providers. Our predictive model analysis can help in conducting these comparisons

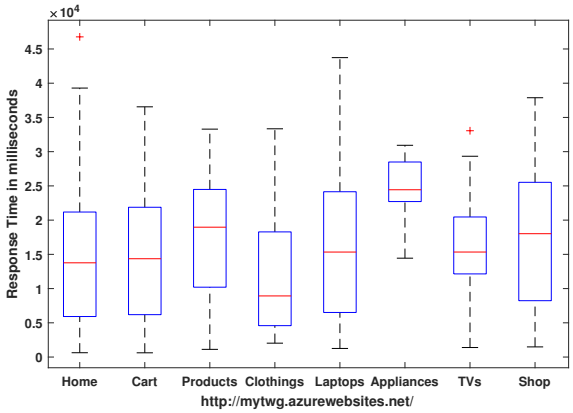
in a more comprehensive way. Modeling and being able to predict response time, bandwidth or latency is essential in leveraging the control of workload fluctuations, managing resource contention and making optimal decisions. For instance, having accurate predictions about the performance characteristics of an application can help in deciding how to manage the arrival rates of workload in data centers, especially during peak periods. Adagna *et al.* [8] in a related work underscored the relevance of these metrics especially in a realistic estimate of the QoS model parameters (which include bandwidth variations, response time and network latencies).

7.1 Case study

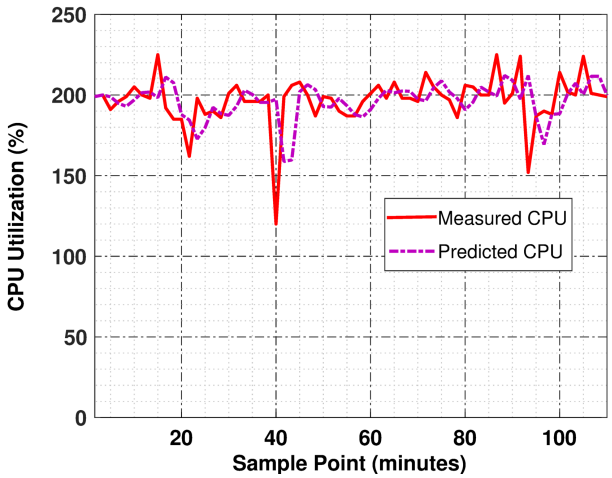
The graphs in Fig. 8 show a box plot of the sampled response time and the percentage CPU utilization measured against a high influx of virtual users opening sessions with the application. With respect to the three training algorithms, the response looks predictable with both BDT and OLS. As a result of the poor performance of the SGD algorithm, it is unable to detect some of the volatilities that have been captured by BDT and OLS (see Table III). As shown in the plots, the first half of the experiments indicate a stable response time for the resource but these tend to be volatile in the second half of the experiment. These spikes averaging more than 10 seconds could suggest the limiting factor of the capacity of the provisioned A1-series VM. This could also be due to a virtualization contention or high demands placed on the resources at the backend. The server degrades quickly as we further increase the number of users beyond 1000 even though CPU utilization dynamically scales up to more than 200% as shown in Fig. 8b. Contrary to normal desktops or servers, Azure virtual machines are designed to be highly elastic and scalable, which allows them to dynamically adjust the CPU utilization to as high as 600% [40]. This indicates why as we simulated extremely high workloads, the A-series virtual machine adjusted the CPU utilization to more than 200% which is in tandem with the spikes experienced as shown in the response time box plots in Fig. 8a.

In order to determine the real cause of the abnormal latencies we contacted the Microsoft Azure technical team, who reviewed the platform from the server side and found that there was no additional latency incurred from the application infrastructure. This led to the conclusion that the latency delays could be arising from the application code. Together with the Azure technical team we scanned and analyzed 620 slow requests and identified that module(s) consuming most of the time are the FastCgiModules (94.07%). Requests are spending most of the time in the CGI module which means that the underlying application code (PHP, NodeJS etc.) is taking a longer time.

Our goal with this case study is not only to demonstrate the efficacy of the BDT and the selected benchmarking training algorithms but to also draw meaningful conclusions that could possibly help in a decision-making process. We observe a linear increase in response time and CPU utilization corresponding to an increase in virtual users. A suitable recommendation here could be a review of the architectural properties of the applications such as scripts for server instantiation or logins to the database that could result in requests queuing.



(a) The sampled average response times of the application server under extreme workload ($vU > 1000$) fluctuation at different sites.



(b) The sampled and predicted % CPU utilization under extreme workload ($vU > 1000$) fluctuation.

Figure 8. The average response times and the % CPU utilization of the application server and the virtual infrastructure network under extremely high workload influx.

These discussions lead us to conclude, with regard to the questions in section 4.2 that with a suitable algorithm (e.g. the BDT) it is possible to project in advance the performance metrics (e.g. response time, CPU, latency etc.) of an application server before an application can be deployed onto the cloud environment. This analysis is helpful in comparing different cloud service providers for the type of resources they offer. We have also been able to demonstrate that our application performed optimally only under normal workload ($vU < 1000$). Under an extremely high workload ($vU > 1000$), resources degraded quickly where the requests were queuing at the FastCGI module.

The potential threat to validity in the application of this regression algorithm is that, in the presence of outliers and noise, the minimization of the sum of squared errors can have negative performance effects on the algorithm. This constitutes a potential threat to internal validity especially where measurements are performed in a highly dynamic and noisy setup.

The number of trees and tree depth do pose a constraint to performance improvement. Growing excessive trees can cause over-fitting leading to both internal and external threats to validity. We handled this by stopping at the point where the loss value converges and we limited the tree depth to shorter ones in the experiments.

In addition, the problem of data peeking can surface when applying this algorithm to a training set example. Peeking is said to occur when data intended for the testing phase of the model building is somehow leaked to the algorithm before its performance is validated. We mitigated this problem by having a totally different set of training and testing data.

8 CONCLUSIONS AND FUTURE WORK

We employed a realistic cloud testbed to investigate different algorithms for use in proactive monitoring and adaptation. These used a collection of realistic KPIs measuring a virtual infrastructure network and an application service. To this end we designed a web service platform and remotely hosted this at different geolocations. We aimed to simulate real user behavior by programming robot users that open sessions and consume our cloud resources in Microsoft Azure. We employed JMeter as our client-server emulator for distributing a huge amount of workload streamed from different geolocations onto our application server and the virtual infrastructure network. We further interfaced our webservice platform with Google Analytics and the Azure Application Insights for live server-metrics monitoring and sampling.

Our framework applied the BDT machine learning algorithm in training and evaluating models on the KPIs. The application of the boosted decision tree regression method yielded predictions of the cloud server KPIs with accuracies intervals of 98.57% which completely replicates the input signal. The high confidence intervals from the training and testing evaluations are strong indications of how well the model fits to the dataset.

A comparison with some state-of-the-art reactive solutions indicate that the ML approach with the BDT algorithm outperforms these techniques. A further comparison with the well known standard OLS and the non-linear SGD shows that the BDT algorithm has the best performance in terms of prediction accuracy. The framework is suitable for conducting fundamental analyses on cloud application servers before deploying resources to the cloud environment.

The maturity of IoT, cloud/fog/edge and autonomous computing has seen most research activities in the last decade directed towards system performance optimization. Most research activities have applied AI, machine learning, and deep learning algorithms with successful results in increasing system performance. In addition, the emerging trends in next generation computing are directed towards the application of AI/ML - integrated applications particularly in IoT, cloud, fog, edge, serverless and Quantum Computing [48]. A very exciting field is evolving to explore the future of deep learning through the use of transformers, transfer learning, meta-learning, recurrent independent mechanisms in building neural nets that enable a quick adaptation to a new environment.

REFERENCES

[1] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of Cloud monitoring tools: Taxonomy, capabilities, and objectives," in *Journal of Parallel and Distributed Computing*, vol. 74, No. 10, Article No. 10, pp. 2918–2933, 2014

[2] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication*, vol.10, pp. 800–845, Jan. 2011.

[3] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 72, pp. 1-14, 2010.

[4] C. Delimitrou and C. Kozyrakis, "HCloud: resource-efficient provisioning in shared cloud systems," in *Proceedings of the 21st Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Atlanta GA, April 2016.

[5] M. Colajanni, M. Pietri, S. Tosi, and M. Andreolini, "Adaptive, scalable, and reliable monitoring of big data on clouds," in *Journal of Parallel and Distributed Computing*, vol.79–80, pp. 67–79, 2015.

[6] M. Colajanni, M. Pietri, S. Tosi, and M. Andreolini, "Real-Time adaptive algorithm for resource monitoring," in *9th International Conference on Network and Service Management 2013 (CNSM 2013), Zuerich, Switzerland* , vol.8226, No. 1, pp. 67–74, Oct. 2013.

[7] N. Mohammed and J. Al-Jaroodi, "Real-Time big data analytics: Applications and challenges," in *2014 International Conference on High Performance Computing and Simulation (HPC)*, pp. 305–310, July 25, 2014.

[8] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," in *Journal of Parallel Distribution Computing*, 72, pp. 796-808, 2012.

[9] M. S. Aslanpour, S. E. Dashti1, M. Ghobaei-Arani, and A. A. Rahmanian, "Resource provisioning for cloud applications: a 3-D, provident and flexible approach," in *Journal of Supercomput 74*, pp. 6470–6501, 2018.

[10] D. Menascé, V. Almeida, and L.Dowdy, "Capacity planning and performance modeling: from mainframes to client-server systems," *Prentice-Hall*, Inc. NJ, USA, 1994.

[11] J. Rolia, and VVetland, "Correlating resource demand information with ARM data for application services," in *Proceedings of the 1st international workshop on Software and performance. ACM*, Santa Fe, New Mexico, USA, pp 219–230, 1998.

[12] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proceedings of the 4th ICAC Conference*, Jacksonville, Florida, USA, pp 27–27

[13] J. Kumar and A. K. Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution," in *Future Generation Computer Systems, Elsevier B.V*, Vol. 81, pp. 41–52, 2018.

[14] T. Ban, R. Zhang, S. Pang, A. Sarrafzadeh, and D. Inoue, "Referential kNN regression for financial time series forecasting," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol.8226, No. 1, pp. 601–608, Oct. 2013.

[15] A. Eddahech, S. Chtourou, and M. Chtourou, "Hierarchical neural networks based prediction and control of dynamic reconfiguration for multilevel embedded systems," in *Journal of Systems Architecture, Elsevier B.V.*, Vol. 59, No. 1, pp. 48–59, 2013.

[16] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," in *Future Generation Computer Systems, Elsevier B.V.*, Vol. 28, No. 1, pp. 155–162, 2012.

[17] S. Tofighy, A. A. Rahmanian, and M. Ghobaei-Arani, "An ensemble CPU load prediction algorithm using a Bayesian information criterion and smooth filters in a cloud computing environment.," in *Softw Pract Exper. 48, John Wiley & Sons, Ltd.*, pp. 2257–2277, 2018.

[18] N. Chauhan, and R. Agrawal, "Probabilistic Optimized Kernel Naive Bayesian Cloud Resource Allocation System," in *Wireless Personal Communications, Springer Science+Business Media, LLC, part of Springer Nature 2022*, https://doi.org/10.1007/s11277-022-09493-5.

[19] M. Ghobaei-Arani and A. Shahidinejad, "An efficient resource provisioning approach for analyzing cloud workloads: a metaheuristic-based clustering approach," in *The Journal of Supercomputing (77)*, pp. 711-750, 2021.

[20] A Belgacem, "Dynamic resource allocation in cloud computing: analysis and taxonomies," in *ACM DL Computing*, Vol. 104, No. 3, pp. 681–710, 2022. https://doi.org/10.1007/s00607-021-01045-2
[21] A. Pahlevan, X. Qu, M. Zapater, and D. Atienza, "Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 1667-1680, August 2018.
[22] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud system," in *Proceedings of IEEE/ACM Intenational Symposium Cluster Cloud Grid Computing (CCGrid)*, Ottawa, ON, Canada, pp. 498–506, 2012.
[23] MLBench: "Distributed machine learning benchmark," 2019, Available online: https://mlbench.readthedocs.io/en/latest/index.html
[24] Y. Liu, H. Zhang, L. Zeng, W. Wu, and C. Zhang, "MLBench: benchmarking machine learning services against human experts," in *Proceedings of the VLDB Endowment*, 11(10), 1220-1232, Rio de Janeiro, Brazil 2018.
[25] M. S. Al-Asaly, M. A. Bencherif, A. Alsanad, and M. M. Hassan " A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment," in *Neural Computing & Applic (2021)*, https://doi.org/10.1007/s00521-021-06665-5.
[26] W. Zhang, L. T. Yang, P. Duan, F. Xia, Z. Li, Q. Lu, W. Gong, and S. Yang, "Resource requests prediction in the cloud computing environment with a deep belief network," in *Softw Pract. Exper. 47(3)*, pp. 473-488, 2017.
[27] F. Qiu, B. Zhang, and J. Guo,, " A deep learning approach for VM workload prediction in the cloud," in *2016 IEEE/ACIS 17th Int Conf Softw Eng Artif Intell Netw Parallel/Distributed Comput SNPD*, 319–324, 2016.
[28] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, " An efficient deep learning model to predict cloud workload for industry informatics," in *IEEE Trans Ind Inform 14:*, pp. 3170–317, 2018.
[29] D. Yang, J. Cao, J. Fu, and J. Guo, "A pattern fusion model for multi-step-ahead CPU load prediction.," in *The Journal of System Software, 86(5)*, pp. 1257-1266, 2013.
[30] G. K. Shyam and S. S. Manvi, " Virtual resource prediction in cloud environment: a Bayesian approach," in *The Journal of Network Computer Applications (65)*, pp. 144-154, 2016.
[31] F. Dong, J. Luo, A, Song, J. Cao, and J. Shen, " An effective data aggregation based adaptive long term CPU load prediction mechanism on computational grid.," in *Future Generation Computing Systems. 28(7)*, pp. 1030-1044, 2012.
[32] T. W. Gyeera, A. J. H. Simons, and M. Stannett, "Kalman filter based prediction and forecasting of cloud server KPIs," in *IEEE Transactions on Services Computing*, 2022, doi: 10.1109/TSC.2022.3217148.
[33] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, " Host load prediction with long short-term memory in cloud computing.," in *Journal of Supercompu*, pp. 1-15, 2017.
[34] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," in *The Annals of Statistics, Institute of Mathematical Statistics*, vol.29, No. 5, pp. 1189–1232, Oct. 2001.
[35] S. J. Russell, "Artificial intelligence : a modern approach," *Prentice Hall series in artificial intelligence, Prentice Hall*, 2nd ed, International ed., Upper Saddle River, N.J., isbn. 0130803022
[36] T. Dietterich, A. Ashenfelter, and Y. Bulatov, "Training conditional random fields via gradient tree boosting," in *ACM International Conference Proceeding Series; Vol. 69: Proceedings of the twenty-first international conference on Machine learning; 04-08 July 2004* vol. 7, December 2013.
[37] M. Schmid and T. Hothorn, "Flexible boosting of accelerated failure time models," in *BMC Bioinformatics*, vol. 9, No. 1, pp. 269–269, 2008.
[38] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery*, isbn 9781450342322, vol. 13-17, 2016.
[39] A. Enatekin and A. Eknoll, "Gradient boosting machines, a tutorial," in *Frontiers in Neurorobotics, Frontiers Media S.A*, vol. 7, December 2013.
[40] D. Stephens and B. Wren, "Azure monitor application insights documentation," *Azure, Microsoft Research Academic*, 2017, Available on: "https://docs.microsoft.com/en-us/azure/azure-monitorl/" Accessed: 2020-02-02.
[41] Google Analytics: "All web site data (audience, behaviour, events and conversions)," 2017, Available on: "https://analytics.google.com/analytics//" Accessed: 2020-02-02.
[42] L. Barroso and U. Hoelzle, "The Datacenter as a computer: An introduction to the design of Warehouse-Scale machines," *MC Publishers*, 2009.
[43] Apache JMeter - User's manual, *The Apache Software Foundation*, 2017.
[44] Microsoft Azure: "Machine learning studio documentation," *Azure, Microsoft Research Academic*, 2020.
[45] J. Gareth, W. Daniela, H. Trevor, and Robert Tibshirani, "An Introduction to statistical learning : with applications in R," *New York: Springer, 2013.*
[46] Y. Liu, H. Zhang, L. Zeng, W. Wu, and C. Zhang, "MLBench: How good are machine learning clouds for binary classification tasks on structured data?," in *PVLDB, arXiv preprint arXiv:1707.09562, Microsoft Academic* 11(10), 1220-1232, 2017.
[47] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 72, pp. 1-14, 2010.
[48] Gill et al, "AI for next generation computing: Emerging trends and future directions," in *Proceedings of the Internet of Things (IoT)*, vol. 19, pp. 100514, August 2022.

Dr Thomas Weripuo Gyeera is an Assistant Professor at the University of Massachusetts Dartmouth. He received a PhD degree in computer science from the University of Sheffield UK in 2019 and an MS in computer and network engineering with distinction from Sheffield Hallam University, UK in 2014. He received a BSc degree in computer science and communications engineering from the University of Duisburg in 2005. He has worked for Thales Group and Ford motor company as an application development engineer. He has been working on using machine learning and adaptive algorithms for proactive cloud computing resources monitoring and adaptation. His major interests and work are in AI, Deep and Machine learning, cloud computing, application development, network engineering and Big Data.

Dr Anthony J.H. Simons is a Senior Lecturer and member of the Testing Research Group in the Department of Computer Science at the University of Sheffield. His current research is in model-based testing and model-driven engineering, coming out of work in object-oriented testing, type theory and precise notations. His early research was in speech and language processing. Previously he served as departmental director of undergraduate admissions and director of teaching.

Dr Mike Stannett is a Senior Lecturer and member of the Verification Research Group in the Department of Computer Science at Sheffield University. Originally trained as a mathematician, his recent work focusses mainly on machine-verification of physical theories, though he has also worked in macroeconomic forecasting. He is a member of several professional societies, including the London Mathematical Society, the Royal Economic Society and the Association for Symbolic Logic.