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

Thomas Weripuo Gyeera [1]

[1]The University of Sheffield

May 14, 2021

Abstract

The National Institute of Standards and Technology defines the fundamental characteristics of cloud computing as: on-demand computing, offered via the network, using pooled resources, with rapid elastic scaling and metered charging. The rapid dynamic allocation and release of resources on demand to meet heterogeneous computing needs is particularly challenging for data centres, which process a huge amount of data characterised by its high volume, velocity, variety and veracity (4Vs model). Data centres seek to regulate this by monitoring and adaptation, typically reacting to service failures after the fact. We present a real cloud test bed with the capabilities of proactively monitoring and gathering cloud resource information for making predictions and forecasts. This contrasts with the state-of-the-art reactive monitoring of cloud data centres. We argue that the behavioural patterns and Key Performance Indicators (KPIs) characterizing virtualized servers, networks, and database applications can best be studied and analysed with predictive models. Specifically, we applied the Boosted Decision Tree machine learning algorithm in making future predictions on the KPIs of a cloud server and virtual infrastructure network, yielding an R-Square of 0.9991 at a 0.2 learning rate. This predictive framework is beneficial for making short- and long-term predictions for cloud resources.

Regression Analysis of Predictions and Forecasts of Cloud Data Centre 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—The National Institute of Standards and Technology defines the fundamental characteristics of cloud computing as: on-demand computing, offered via the network, using pooled resources, with rapid elastic scaling and metered charging. The rapid dynamic allocation and release of resources on demand to meet heterogeneous computing needs is particularly challenging for data centres, which process a huge amount of data characterised by its high volume, velocity, variety and veracity (4Vs model). Data centres seek to regulate this by monitoring and adaptation, typically reacting to service failures after the fact. We present a real cloud test bed with the capabilities of proactively monitoring and gathering cloud resource information for making predictions and forecasts. This contrasts with the state-of-the-art reactive monitoring of cloud data centres. We argue that the behavioural patterns and Key Performance Indicators (KPIs) characterizing virtualized servers, networks, and database applications can best be studied and analysed with predictive models. Specifically, we applied the Boosted Decision Tree machine learning algorithm in making future predictions on the KPIs of a cloud server and virtual infrastructure network, yielding an R-Square of 0.9991 at a 0.2 learning rate. This predictive framework is beneficial for making short- and long-term predictions for cloud resources.

1 Introduction

Cloud computing service providers seek to deploy vast quantities of shared resources that are managed in an optimal way for all consumers. *Monitoring* describes a collection of techniques for measuring cloud resource consumption, performance, and for detecting anomalies [1]. *Adaptation* describes a collection of techniques for effecting changes to a cloud configuration in order to improve performance, or to counter anomalous behavior.

The state-of-the-art in cloud monitoring mostly generates statistics about Infrastructure-as-a-Service (IaaS) [2] which is used privately to inform providers about the health of the service. Consumers see other aspects, such as service latency and response times, affected by the number of tenants on a given platform [3], [4]. However, the relationship between low-level resource provision and high-level consumer experience is ill understood. While cloud providers offer Service Level Agreements (SLAs), it is often not possible to tell whether these are being honoured. Providers want to predict performance, plan capacity and detect anomalies well in advance; and consumers need guarantees that their SLAs are being honoured.

Many attempts have been made in recent years to address this issue. For example, Kalyvianaki *et al.* [5] considered this problem on the basis of reactive monitoring and adaptation of virtualized server applications, arguing that data centre resources should be provisioned and managed *reactively*. As a concrete example, they derived mathematical relations between the allocation and utilization of the server CPU and based on these relations, showed that appropriate dynamic adjustment of its percentage CPU allocation and utilization could be carried out on-demand. We argue, however, that modern data centres have become increasingly dynamic, complex and heterogeneous, and that the reactive approach may not, therefore, be wholly suitable for managing modern data centre servers and applications. Our approach focusses instead on *proactive* monitoring and adaptation, where Key Performance Indicator (KPI) patterns for application servers are analyzed in advance.

It has often been observed that both small and large organizations are rapidly moving to the cloud because of the immediate cost benefits in adopting this technology, and in so doing they increase the amount of data managed per unit time interval in the supporting data centres. The multi-shared environment of modern cloud data centres enables great flexibility with which cloud computing resources can be provisioned or deallocated. But this approach results in the generation of huge amounts of data that are generally characterized by the '4Vs' (volume, velocity, variety and veracity) [6], [7]. With these huge volumes of data from data centres, we argue that the behavioral patterns and KPIs characterizing virtualized servers, networks and database applications can best be studied and analyzed with predictive models such as the linear boosted decision tree regression algorithm. This approach allows both long- and short-term predictions to be used as a guide for the provisioning and deallocation of cloud data centre resources, and for cloud data centre resource management more generally. A key direct benefit of the approach would be the reduction of over and under provisioning and utilization of cloud computing resources.

We trained, tested, and evaluated predictive models using the boosted decision tree regression algorithm and bench-marked this approach with two state-of-the-art algorithms (the ordinary least squares (OLS) and the stochastic gradient descent (SGD)) [8]–[10]. Like other decision tree algorithms, this is capable of handling outliers in the training

set; it is preferred over other linear regression algorithms, such as the ordinary least squares (OLS) algorithm, because of its speed during training and its ability to scale well on non-parametric data. We considered the accuracy of the prediction algorithm, together with its linearity, the number of features, and other parameters needed when using it. For instance, a problem with high dimensional training set example will require the application of principal component analysis (PCA) or mixture discriminant analysis (MDA), and a classification problem may require the application of multiclass logistic regression to solve complex problems [11].

Service level agreements (SLAs) are designed to establish a bond between a cloud provider and a consumer. These documents outline the quality-of-service (QoS) features that must be maintained and respected by all parties involved. The quality-of-service needs to be described in clear measurable performance metrics and the limitations of the services or IT resource offered to the consumer [12]. In addition to managing SLA, most application load and performance testings at the design, integration, and operational stages need to measure and predict the performance of these metrics in order to support a comprehensive decision-making process (e.g. load balancing, billing, and controlling resource usages) [13].

For these purposes we sampled data relating to the key performance indicators of a real experimental cloud testbed with a virtual infrastructure network (VIN) constructed in Microsoft Azure. This framework enabled us to monitor and measure the following server KPIs: Server hits per second, throughput (average requests per second), bandwidth consumption in Mbps, percentage CPU utilization, average response time/latency in seconds, and Bytes received per second.

Extensive supervised machine learning experiments conducted with the boosted decision tree regression algorithm on our generated data from the application server and the virtual infrastructure network achieved a performance with an R-square value of 0.9991 at a learning rate of 0.2 compared to two state-of-the-art (OLS and SGD).

1.1 Structure of the Paper

The rest of this paper is organized as follows. We present previous work related to this investigation in section 2. Section 3 presents the mathematical constructs and the problem definition in terms of the boosted decision tree (BDT) supervised machine learning algorithm. Section 4 covers the experimental testbed 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 linear 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 precious works. The potential applications, benefits and threats to the validity of our work are discussed in section 7. We summarise our main conclusions in section 8, and include a number of topics for further investigation.

2 Related work

We extend our discussion on the related work to cover some key machine learning algorithms that have been employed in solving the problem of workload prediction in cloud data centres.

The work in Sackl *et al.* [14] presented five models in capturing the effects of bandwidth consumption fluctuations on user's quality of experience. The five models (LTD, SLTD, TJ, AREA, and double models) identified suitable techniques for the observation of bandwidth fluctuations in assessing the overall quality of the network. Their models constructed new KPIs for characterizing the impact of bandwidth fluctuations on the user experience. The LTD model defines a fraction of time for which the bandwidth throughput is said to be observed below the network connection downlink referred to as the BDW. This time range allows a mapping function to be modeled in order to relate the mean opinion score to the fluctuating bandwidth. The selective throughput duration model considers that the duration for which a bandwidth fluctuations drops is unobserved by the user whereas the TJ model is constructed based on the principle of the moving average and avoids the high sampling frequency in order not to obfuscate drops of the bandwidth reference value. Both the AREA and the double models work in a similar way as the LTD.

Menascé *et al.* [15], [16] 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 the 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 control 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 [17] 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.* [18] applied an artificial neural network and the adaptive differential evolution algorithm in predicting cloud data centres workload. They performed experiments on some HTTP server benchmark datasets from NASA and the predicting 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.* [19] proposed the k–nearest neighbor (kNN) approach for making predictions on financial time series dataset. This algorithm was again applied by Eddahech *et al.* [20] 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.* [21] in

predicting workload variations in data centres. 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 widow framework.

The technique of the boosted decision tree method in our approach presents predictions with confidence bounds of 98 % compared to the standard state-of-the-art ML (OLS and SGD) algorithms. A prediction accuracy of 98 % means that we are able to match the utilization signal and therefore we are able to detect potential violations, over, and underutilizations of the server resources (e.g. CPU utilization, bandwidth traffic).

Chen *et al.* [22] presented combined techniques based on both neural network and the steepest descent algorithms in making predictions on workload fluctuations in data centres. Even though the technique suffers from high prediction errors, it turns to improve on the prediction accuracy with time delay.

Fan-Hsun *et al.* [23] presented a framework based on a multi-objective genetic algorithm (GA) for dynamically predicting and allocating cloud data centre resources. The proposed GA predicts the resource requirements of the next step ahead time slot based on the historical patterns of the previous time slot. A similar approach of using the stochastic model and the neural network was presented by Prevost *et al.* [24] in predicting workload fluctuations in cloud data centres. The approach exploited the benefits of back propagation of neural networks and the high prediction accuracy of the stochastic model.

Hu *et al.* [25] proposed a framework based on statistical learning theory in constructing models using the Kalman smoother and the support vector regression algorithms. Prediction accuracy with their approach were evaluated to be higher than the techniques that employ auto regression, the back propagation neural network, and the canonical support vector regression algorithms.

Pahlevan *et al.* [26] 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% 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 the correlation and network-aware [27] 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

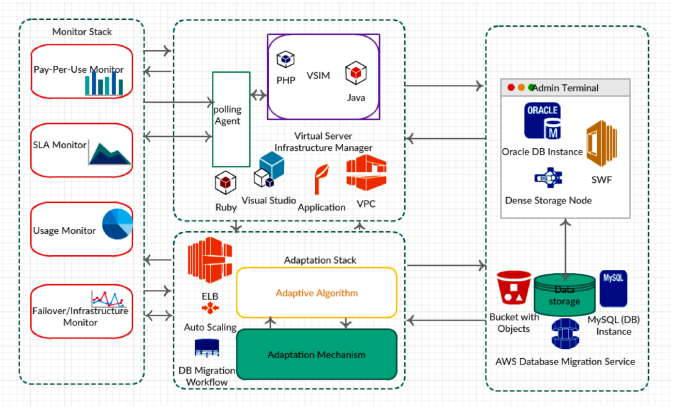


Fig. 1. Block diagram of of the conceptual framework.

in terms of server overutilization and network traffic overhead especially in VMs off-peak loads management.

Our approach applies the linear 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 the state-of-the-art ordinary least squares and the stochastic gradient descent regression machine learning algorithms. Our approach is further compared with the work presented in [13], a reactive framework that concentrates on cloud systems workload capacity planning, allocation, and redirection. In order to state the benefits of our framework, extensive discussions on comparing our approach to some state-of-the-art techniques such as [10], [13] are covered in section 6.

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 stated in this approach as one that 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 KPI such as bandwidth fluctuations, latencies or the percentage 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 a metric for evaluating and billing the client as specified within the SLA. For the purpose of enforcing an SLA

contractual agreement, the SLA monitor here can display metrics such as the availability of the resources that were provisioned within the cloud. The fail-over/infrastructure monitoring stack is quintessential in characterizing transient and general network issues. This component is generally required in detecting failures and anomalous behaviors, and how they can be mitigated before the virtual network and the application server becomes unavailable.

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 key performance indicators 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 can be enforced such as elastic load balancing, auto-scaling of pooled resources or the migration of a DB workflow.

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 terminals administer 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 [28], [29]. In these algorithms, the idea is to teach a model how to perform 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 hypotheses space are combined in determining the predicted output [30], [31].

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* [29], [32] (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 from 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 bandwidth fluctuations or the percentage CPU utilization constitutes the training set example from the designed experiment, then it is computationally realizable using the decision tree model objective function to make predictions on the server KPIs. The rest of this section presents the mathematical background behind the construction of the boosted decision tree as a predictive model according to [28], [32].

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., [29]).

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

If the decision tree can be constructed through the combination of the sub-hypothesis 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 square errors. 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:

$$Error(E) = \sum_{j=1}^{m} e(z_j, \hat{z}_j) \tag{3}$$

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:

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

In order to characterize the complexity of the new model, we define an additive L_2 regularization hyperparameter (see [32], [33] for the definition of the L_2 regression) as a component of our objective function. This component combines the 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 [31], 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 to the current estimates. These estimates in the hypothesis space form the decision trees based stumps on *if-then* clauses evaluation.

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 which is characterized by the sum of the heuristic hypotheses can be split from the

base-learner into sub-leaves consisting of the sub-functions within the ensemble defined in (1) by applying the information gain (see definition in [28], [32], [33]). The complete objective function can be stated as follows:

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

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 [28], [32]. 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):

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

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, then the error square objective function can be rewritten [12] as:

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

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

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

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

The component $\Psi(h_m)$, called the regularization parameter, characterizes the complexity of the model. There are several regularization methods in the literature (see [28], [32], [33] 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 the 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 [28]).

Algorithm 1 The Boosted Decision Tree algorithm [28]

procedure BDT

initialization
Consider an N training set example $\{x_i, z_i\}_{i=1}^N$
Define a finite set, $\hat{H}$, of hypotheses
Construct the error function
Construct the base-learner
$\hat{h}_m \leftarrow constant$

read current state
for $i \leftarrow 1, N$ **do**
Compute the negative gradient Δ
Add a new base-learner to the hypothesis space
Optimize the gradient descent Δ
Update the ensemble with the hypothesis estimate:

$$\hat{z}_m \leftarrow \left(z_j^{(m-1)} + h_m(x_j)\right)$$

end for

return results
end procedure

3.3 How to Split the Leaves of the BDT

In this section, the information gain used in splitting the leaves of a decision tree algorithm are given (following [32]). In section 3.2, we presented details of how the base-learner hypothesis and sub-hypotheses can be constructed to fill the hypothesis space.

By defining the base function, the leaves of the tree can be optimally constructed through equations (1) to (7) with the L_2 regularization loss function. According to [28], [32] the sub-leaves can be greedily added to grow the tree using the information gain explained as follows.

$$weight_{optimal} = -\frac{\Delta}{(\Delta^2 + \lambda)} \tag{12}$$

$$\Delta = \partial_{\hat{z}^{(m-1)}} e(z_j, \hat{z}_j) \tag{13}$$

$$\Delta^2 = \partial^2_{\hat{z}^{(m-1)}} e(z_j, \hat{z}_j) \tag{14}$$

$$Error(E) = -\sum_{j=1}^{T} \frac{\Delta^2}{(\Delta^2 + \lambda)} + \gamma T \tag{15}$$

If we consider the leaves on the stump of a tree to be partitioned as left and right sub-leaves, then the deviations from the partial derivatives on the error loss function can be determined to greedily grow the tree. This leads to definition of the information gain as:

$$\frac{1}{2}\left[\frac{\Delta_l^2}{(\Delta_l^2 + \lambda)} + \frac{\Delta_r^2}{(\Delta_r^2 + \lambda)} - \frac{(\Delta_l + \Delta_r)^2}{(\Delta_l + \Delta_r) + \lambda)}\right] - \gamma T \tag{16}$$

where Δ_l and Δ_r are computations of the partial derivatives of the left and right sub-leaves on the tree. Equation (16) is the information gain that can be used to optimally split and grow the tree. Usually no further splitting is necessary if the gain is negative [32]. Further reading on information gain can be found in [28].

4 Experimental design

This section consists of the various experimental methods and tools set up in order to critically evaluate the conceptual framework outlined for this research activity. 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 the East US data center is shut down for one reason or another, 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 server. In addition, the load balancer redirects the workload to a new server if the target server is overloaded.

In addition to the virtual infrastructure network, the web service platform is interfaced with Azure application Insights [34] in order to monitor live server KPIs as the experiment is being run. Instrumenting Google Analytics [35] 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 server.

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 in simulating different user scenarios on the webservice platform. The results of these experiments are shown in Fig. 1.

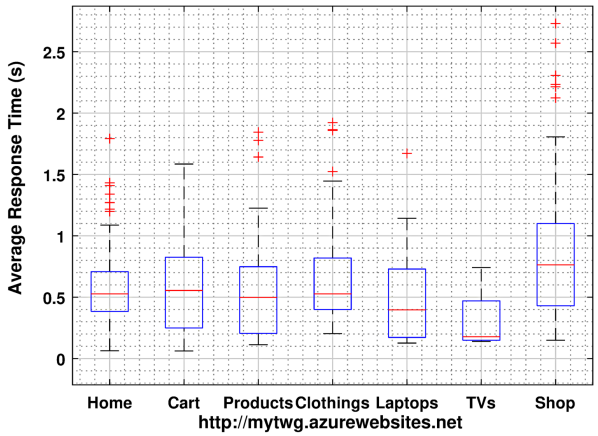
We aligned the streaming of the workload characteristics of the virtual machines provisioned in Azure to be homogeneous in terms of CPU, RAM, and adopt the approach in [13], [36]. A homogenous workload influx assumed here can also be applied to heterogenous VMs and cloud resources. In a co-located environment such as Azure, multiple VMs do run in parallel and in cases of high resource demands, workload is evenly distributed among multiple virtual machines [13].

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.

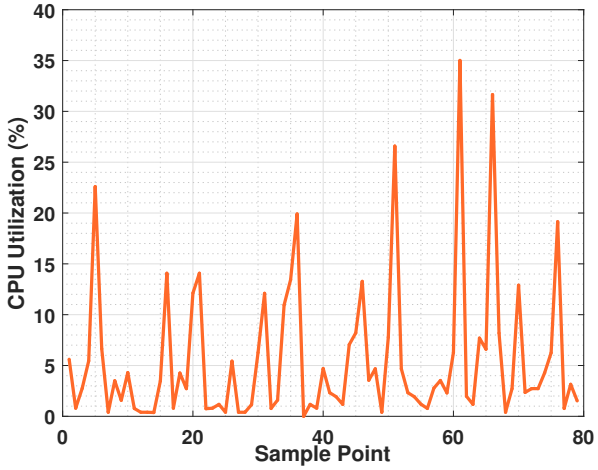
In experiment Two, the objective was to simulate a real user behavior by employing the concepts of pacing and think time. In a real world web platform, users do not usually execute their actions on browsing an application all at once but there is usually some delay in the series of activities referred 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 with 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 group (see [37] for more on JMeter ultimate thread group) from JMeter in distributing the load across the server.



(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.

Fig. 2. Experimental results of the average response times and the % CPU utilization of the application server and the virtual infrastructure network under normal load ($vU < 1000$) influx.

These four main experiments were conducted repeatedly for four weeks in generating and measuring 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 include 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 centres management especially in designing SLAs, billing, and load admission control. The work presented in [3] explains in detailed how these KPIs are relevant in selecting a suitable cloud service provider by a consumer.

Fig. 2a 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. 2a sampling on the TVs site has the lowest mean and median response time indicated by the red line on the box plots. The shop site indicates high response time with additional high number of outliers (shown in red plus) shown as a composite plot in Fig. 2. The rest of the sites do indicate similar mean response times as shown in the remaining six sampling sites. The mean response times from all the sampling sites indicate an overall predictable average response except for the few outliers highlighting some potential volatilities under a normal workload characteristic. We defer the discussion on the high number of outliers sampled at the shop site and some of the sampling sites until the case study in section 7.1.

Fig. 2b 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 at a slightly 1000 virtual users. The maximum utilization of 35.01% is a little above the normal CPU characterization (20%) under normal data centre workload as reported in [36], [38]. The volatilities shown in the response time (indicated in the outliers) plots 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 on case study).

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 at a defined space of time.

With the JMeter application, a Java API seen as an ideal client-server emulator with the capability of generating 100s to 1000s of HTTP user requests per server (e.g. for a system with hardware requirements of 8 GB, 4 vCPU Cores, JMeter can generate as many as 1000 virtual users on a particular server) something that would be unimaginable to simulate with real users at a defined time period. On JMeter, we simulated the number of virtual users (vU) with Java thread samplers implemented in JMeter 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 (vU) 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) [37].

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 objective here is to have a test plan that performs assertion tests on the functional blocks of the application. There are

several assertion tests that can be performed on a web application to ascertain its functional correctness some of which include the HTML, response, MD5Hex, XML, and duration assertions. Having verified the correct functionality of the application we then deployed it live onto the different servers geolocated across the East US Microsoft Azure cloud data centres with our client emulator (JMeter API) sited at the Microsoft European data centre in Ireland.

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 configurations parameters allows us to monitor and collect the data needed for building our predictive models the linear boosted decision tree regression algorithm.

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

1) *For a given number of users, can we forecast in advance if the to-be-provisioned bandwidth would be adequate for the set of servers in the virtual infrastructure network?*
2) *For a given application, would the response time, latency or throughput from the servers be adequate or not?*
3) *For some of the metrics described in section 3.3, we selected suitable features to build the predictive models based on the simulated data. For instance, given a certain number of virtual users (vU), how much variation of the response time do we expect for a mission critical application (e.g. a flight radar tracking software.) like the one built for this experiment?*

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 [39], 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).

For instance, the question a cloud consumer may want to ask is whether the allocated memory or any of the KPIs characterizing the server application would be enough for the deployment of a desired application. Thus the results of predictions from these models can be used as a baseline in proactively determining how a new application or resource will perform when deployed on the cloud.

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 dataset. We also applied data manipulation queries to transform and clean the raw data into a suitable format that would increase prediction accuracy.

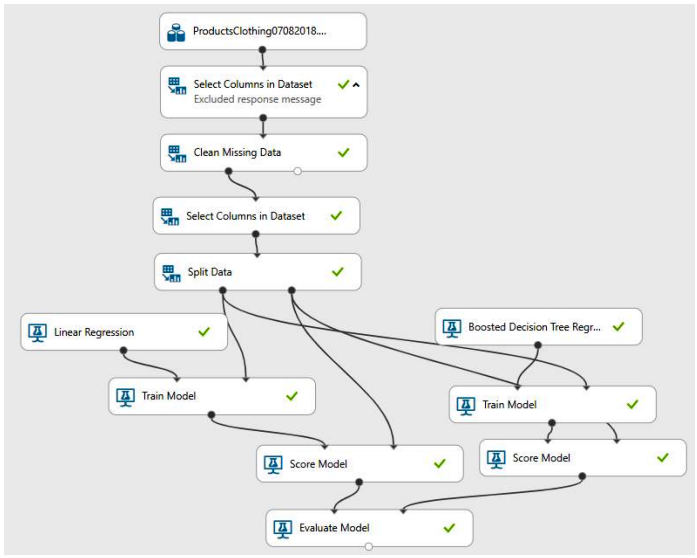
Fig. 3 illustrates the machine learning experiments conducted in Azure to train the models with the boosted decision tree regression algorithm. Fig. 3a shows the initial settings and all the modules selected for the complete machine learning experiments for training the dataset. The validation experiment is shown in Fig. 3b where the results of the entire experiment are converted into a webservice input platform for external users to consume.

We designed the boosted decision tree regression experiment on 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. 4 shows the 1^{st} and 7^{th} iterations of the sub-hypotheses in filling the hypothesis space. The 100^{th} iteration of the decision trees indicates the final hypothesis constructed for the hypothesis spaces of 100 trees Fig. 5. The iteration trees depicted in these figures are sub-hypotheses of the hypothesis space illustrated in equation (1). As described in section 3.2, for example, the 9^{th} iteration tree shown in Fig. 5a represents the new function $h_m(x_j)$ that is added to the growing tree depicted by equation (6) at $j = 9$.

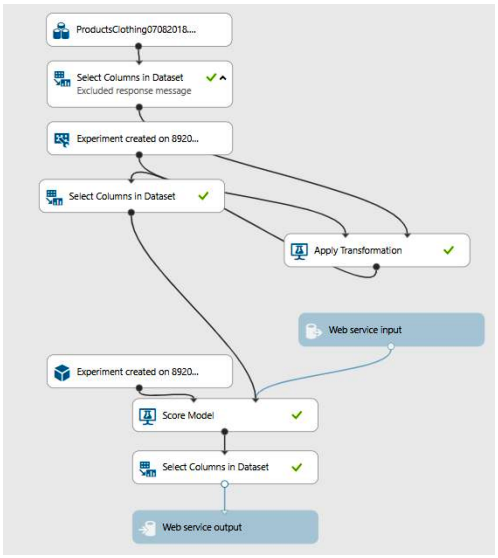
Further, as explained in section 3.2, for the design of the boosted decision tree model we set out a finite hypothesis space ($M = 100$) that averages on the final hypothesis space as indicated in equations (1) and (2). Our parameter λ is 0.001 for the L_2 regulation with a maximum learning rate of 1.0. Both training and validation dataset 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 in these figures. Fig. 4 and Fig. 5 show the iterative steps of the different trees generated within the functional hypothesis defined in equation (2).

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), relative absolute error (RAE), and relative squared error

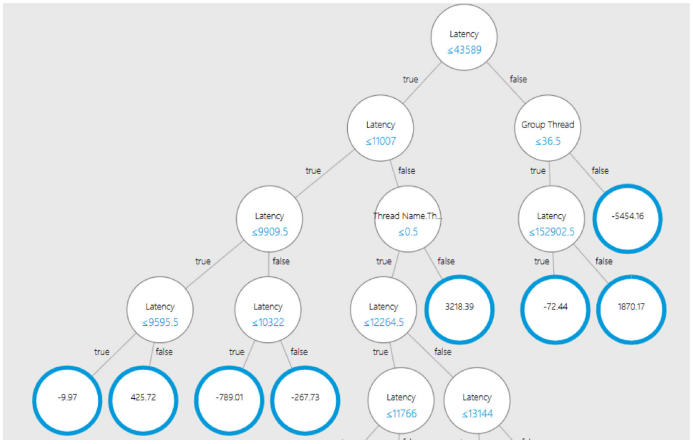


(a) The training experiments with the BDT algorithms.

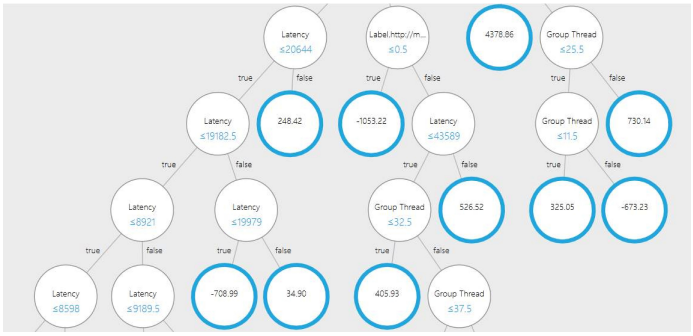


(b) Flow diagram of the testing and validation experiments to deploy the model as a web service.

Fig. 3. These flow diagrams illustrate the machine learning experiments conducted with the boosted decision tree algorithm.

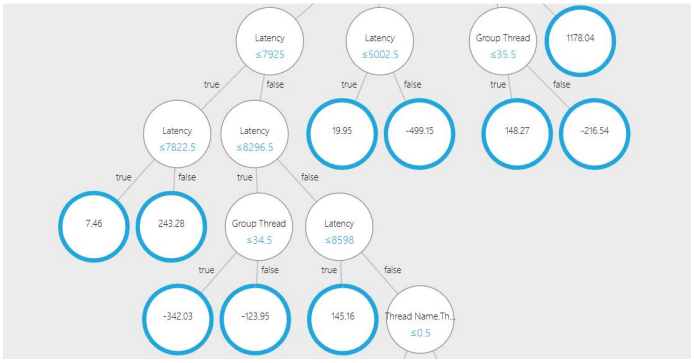


(a) 1^{st} iteration tree.

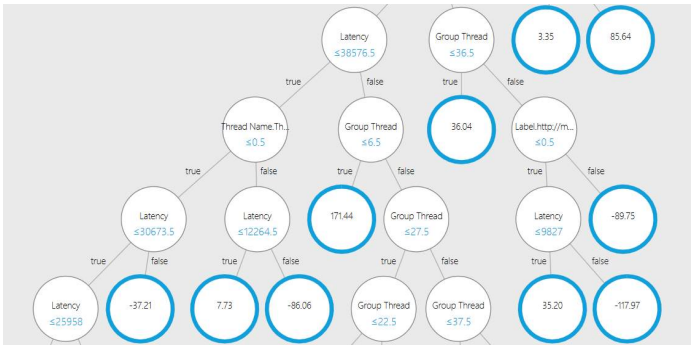


(b) 7^{th} iteration tree.

Fig. 4. The figures illustrate the constructed trees at the 1^{st} and 7^{th} stages of the iteration with the BDT algorithm.



(a) 9^{th} iteration tree.



(b) 100^{th} iteration tree.

Fig. 5. The figures illustrate the constructed trees at the 9^{th} and 100^{th} stages of the iteration with the BDT algorithm.

(RSE), the standard statistical metrics suitable for describing the performance of a model for regression [39].

For the mean absolute error, we computed the difference between the actual and the predicted observations and defined this as an error on the performance of the model. We also computed the RMSE as the error arising by taking the average square roots of the predicted errors of the model fitted to the dataset. Another metric of statistical significance is the mean of the absolute errors computed in relation to the absolute difference between the measured and actual

value referred here as the relative absolute error (RAE) and the square of this yielded an additional metric referred to as the relative squared error.

The results of these metrics selected for evaluating the performance of the model are presented in table I. 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 [11]. From the results in table I, we recorded the best R-Square 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. The observable trend has been that all values at the differently tweaked learning rates are near to unity, a strong indication of how well this algorithm fits to the model.

TABLE I. 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

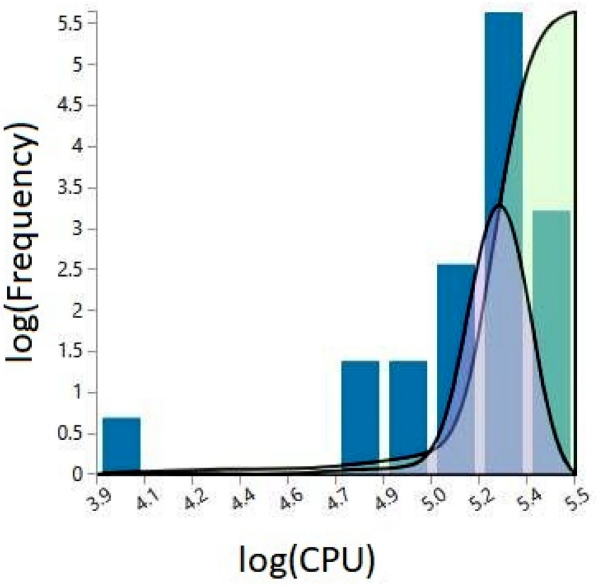
Fig. 6a shows a log scale plot of the measured dataset on the percentage CPU consumption which indicates the linear Gaussian distribution characteristics of the sampled dataset. The figure displays the histograms, plots of the cumulative, and the probability density functions (CDF and PDF) on a log scale of the first 10 bins of the sampled dataset. The plots of the predicted values with the boosted decision tree algorithm on the percentage CPU utilization and network latencies are discussed in section 6.1.

These results in table I 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.

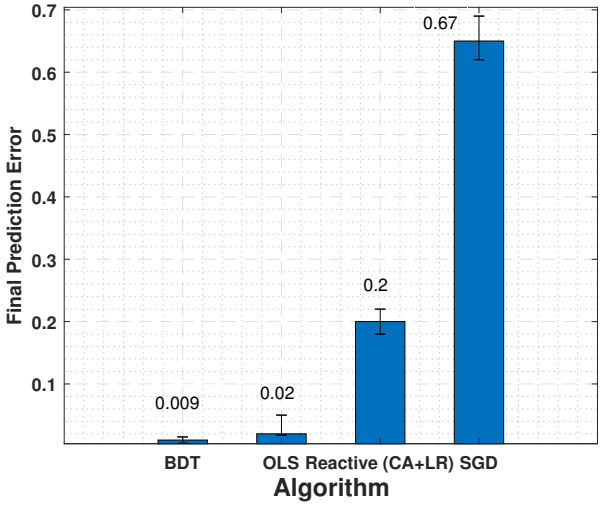
6.1 Benchmarking and comparison with the State-of-the-art

We benchmark and compare the application of the boosted decision tree (BDT) to two standard state-of-the-art models [9], [10], [40] : 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 [10] 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



(a) Percentage CPU utilization (Histograms, CDF and PDF)

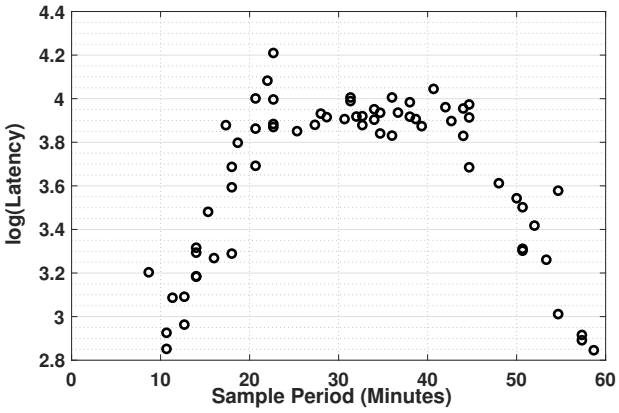


(b) Comparison of the four different algorithms in terms of performance degradation (the smaller the value the better the performance).

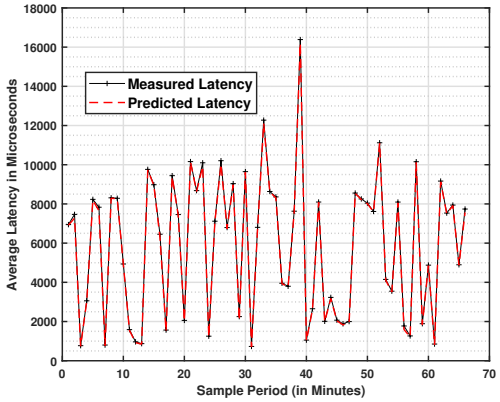
Fig. 6. These figures illustrate the CDF and PDF of the % CPU utilization as well as a comparison plot fo the four algorithms.

the OLS and the BDT training algorithm. In addition to the metrics shown in table II, 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 II. Comparing the standard OLS and the BDT, both algorithms achieve convergence on the models at a very low learning rate with little variations in the model performance as we further increase the learning rates. As shown in table II the boosted decision tree achieves its best coefficient of determination 0.9991 at a learning rate



(a) Scatter plots of the measured average latency on a log scale.



(b) The average latency predicted output.

Fig. 7. These figures illustrate the measured and predicted average latency. The actual predictions are shown in 7b with a prediction accuracy of 98% in the training phase.

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 results of the Boosted decision tree.

In addition to comparing the performance of the boosted decision tree 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 [13].

Ardagna *et al.* [13] 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 request per second without violating the average response time in the SLA document. The LR on the other hand determines at every time the total rate of executions of the web service requests at a particular site and attempts to redirect workload from overburdened VMs which seem to violate the mean response time.

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 the main advantages of the proposed resource management framework.

The machine learning approach focused in this work with the boosted decision tree regression algorithm outperforms the adaptive method presented by Ardagna *et al.* [13] 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 equally outperforms the well-known state-of-the-art ordinary least squares (OLS) and the stochastic gradient descent algorithms (SGD).

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. 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 [13].

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

Model Algorithm	MAE	RMSE	RAE	RSE	CoD	L2
Stochastic Gradient Descent	6057.18	16336.04	0.871653	0.980027	0.332885	0.001
Boosted Decision Tree	177.29	569.70	0.032033	0.001192	0.998808	0.001
Ordinary Least Squares	92.09	172.29	0.013252	0.000109	0.999891	0.001

Fig. 6b shows a comparison of the different algorithms in terms of their performance degradation. The BDT with a final prediction error of 0.009 outperforms all the others whereas the non-linear SGD with a 0.67 final prediction error has the worst performance.

Fig. 7a shows the sampled average latencies plotted on a log scale. A composite plot of the sampled and predicted output on the average response latencies is shown in Fig. 7b. The detected peaks show a simulated effects of load spikes that indicate a potential violation of the QoS with the response latency that is desired to be lower than 100 milliseconds. We present a detailed review of the cause of the high latencies in section 7.

7 Potential Applications and Threats to Validity

We seek to provide a framework that helps consumers to directly project resource performances especially for consumers who want to have *a priori* knowledge about how their applications are likely to perform when migrated into

the cloud platform. For instance, the latency and throughput metrics measurable at the consumer side can help determine the best storage services offered by the provider. This further supports not only the projection of performance characteristics of applications to be deployed in the cloud environment, but also a prediction framework that can be used to conduct a comprehensive analysis on the quality-of-services (QoS) that must be guaranteed within the cloud environment.

The work presented in [41] details the importance of these metrics 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 the value of the response time, bandwidth or latency is quintessential in leveraging the control of workload fluctuations, managing resource contention and making optimal decisions. For instance, having a well predicted information about the performance characteristics of an application can help in deciding how to manage the arrival rates of workload in data centres, especially during peak periods.

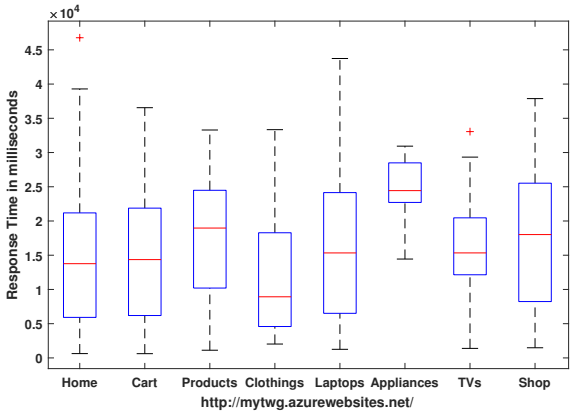
Adagna *et al.* [13] in a related work underscored the relevance of these metrics especially in a realistic estimate of the QoS model parameters (which include bandwidth variations, response time and network latencies).

7.1 Case Study

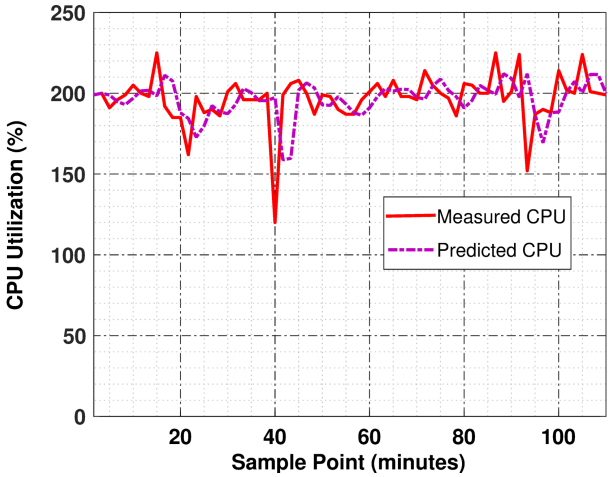
As shown in Fig. 8, these graphs contain a composite box plots of the sampled response time and the percentage CPU utilization as against a high influx of virtual users opening sessions with the application. With respect to the three training algorithms, the response looks predictable with both the BDT 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 the BDT and the OLS (see table II). As shown in the plots, the first half of the experiments indicate a stable response time for the resource but tend to be volatile in the last 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 the 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 that allow them to dynamical adjust the CPU to as high as 600% [34]. 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 delay in latencies we contacted Microsoft Azure technical team and 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. We resolved this issue by debugging the application code which helped eliminate the additional latency delays that we incurred.

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 the response time, and the 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.

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

The discussions by far lead us to conclude on the answers 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 quite optimal only under normal workload ($vU < 1000$). Under extremely high workload ($vU > 1000$) influx, resources degraded quickly where the requests were queuing at the FastCGI module. We had to debug the application in order to improve the performance metrics of the application server.

7.2 Threats to validity

According to the Gauss-Markov theorem, the least means square regression presents the best linear estimator with unbiased linear coefficients [29]. 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 performance of the boosted decision tree regression algorithm is characterized with very high confidence intervals indicating how suitable this algorithm fits to the dataset. The problem of data peeking is a general problem that 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 [28], [29]. We mitigated this problem by having a totally different set of training and testing data.

In all types of mathematical and machine learning problems the phenomenon of model overfitting can occur if one is not careful especially by placing emphasis on meaningless and irrelevant data. We highlight this as a threat to external validity. The problem of overfitting can lead to wrong predictions (e.g. if emphasis is placed on outliers) and the way this was handled with the boosted decision tree algorithm was through pruning of the leaves grown from the hypothesis generated as subfunctions for the stumps of the tree.

8 Conclusions and Recommendations

We proposed a real cloud testbed for proactive cloud resource monitoring, adaptation, and information gathering on cloud virtual infrastructure network and application server KPIs. 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. The sampled server KPIs then served as inputs for building our predictive analytic framework.

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 confidence 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. We aim in future research activities to compare this linear algorithm with other adaptive filtering techniques (Kalman filters and state space models).

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] E. Kalyvianaki, T. Charalambous, and S. Hand, "Adaptive resource provisioning for virtualized servers using Kalman filters," in *LACM Transactions on Autonomous and Adaptive Systems, ASSOC COMPUTING MACHINERY*, vol. 9, No. 2, Article No. 10, pp. 1–35, 2014.

[6] 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.

[7] 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.

[8] 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.

[9] D. Das, S. Avancha, D. Mudigere, K. Vaidyanathan, S. Sridharan, D. D. Kalamkar, B. Kaul, and P. Dubey, "Distributed deep learning using synchronous stochastic gradient descent," in *CoRR, arXiv preprint arXiv:1602.06709, Microsoft Academic, 2016.*

[10] MLBench: "Distributed machine learning benchmark," 2019, Available online: https://mlbench.readthedocs.io/en/latest/index.html

[11] J. Gareth, W. Daniela, H. Trevor, and Robert Tibshirani, "An Introduction to statistical learning : with applications in R," *New York: Springer, 2013.*

[12] T. Earl, Z. Mahmood, and R. Puttini, "Cloud computing concepts, technology, and architecture," *The Prentice Hall service technology series from Thomas Earl. Prentice Hall.*, UPPER SADDLE RIVER, NJ, BOSTON, INDIANAPOLIS, AND SAN FRANCISCO, PP. 359-415, 2014.

[13] 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.

[14] A. Sackl, P. Casas, R. Schatz, L. Janowski, and R. Irmer, "Quantifying the impact of network bandwidth fluctuations and outages on web QoE," in *IEEE, 2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*, pp. 1–6, 2015.

[15] D. Menascé, V. Almeida, and L.Dowdy, "Capacity planning and performance modeling: from mainframes to client-server systems," *Prentice-Hall,* Inc. NJ, USA, 1994.
[16] 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.
[17] 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
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] Z. Chen, Y. Zhu, Y. Di, S. Feng, and J. Geng, "A high-accuracy self-adaptive resource demands predicting method in IAAS cloud environment," in *Neural Network World, Czech Technical University in Prague, Faculty of Transportation Sciences*, vol. 25, No.5 pp. 519–539, 2015.
[23] T. Fan-Hsun, W. Xiaofei, C. Li-Der, C. Han-Chieh, and V.C. M. Leung, "Dynamic resource prediction and allocation for cloud data centre using the multiobjective genetic algorithm," in *IEEE Systems Journal*, Vol. 12, No. 2, pp. 1688–1699, 2018.
[24] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, "Prediction of cloud data centre networks loads using stochastic and neural models," in *IEEE 2011 6th International Conference on System of Systems Engineering*, pp. 276–281, 2011.
[25] R. Hu, J. Jiang, G. Liu, and L. Wang, "Efficient resources provisioning based on load forecasting in cloud," in *The Scientific World Journal, ScientificWorld Ltd.*, Vol. 2014, No. 2, pp. 3212–3231, 2014.
[26] 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.
[27] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud systems," in *Proceedings of IEEE/ACM Intenational Symposium Cluster Cloud Grid Computing (CCGrid),* Ottawa, ON, Canada, pp. 498–506, 2012.
[28] 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.
[29] 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
[30] 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.
[31] M. Schmid and T. Hothorn, "Flexible boosting of accelerated failure time models," in *BMC Bioinformatics*, vol. 9, No. 1, pp. 269–269, 2008.
[32] 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.
[33] A. Enatekin and A. Eknoll, "Gradient boosting machines, a tutorial," in *Frontiers in Neurorobotics, Frontiers Media S.A,* vol. 7, December 2013.
[34] 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.
[35] Google Analytics: "All web site data (audience, behaviour, events and conversions)," 2017, Available on: "https://analytics.google.com/analytics//" Accessed: 2020-02-02.
[36] L. Barroso and U. Hoelzle, "The Datacenter as a computer: An introduction to the design of Warehouse-Scale machines," *MC Publishers,* 2009.
[37] Apache JMeter - User's manual, *The Apache Software Foundation,* 2017.
[38] C. Delimitrou and C. Kozyrakis, "Quality-of-Service-Aware scheduling in heterogeneous Datacenters with Paragon," in *IEEE Micro, Special Issue on Top Picks from Architecture Conferences,* vol. 34, No. 3, pp. 35–45, May 2014.
[39] Microsoft Azure: "Machine learning studio documentation," *Azure, Microsoft Research Academic,* 2020.
[40] 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.
[41] 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.
[42] C. Hankendi and A. K. Coskun, "Scale and cap: Scaling-Aware resource management for consolidated multi-threaded applications," in *ACM Transaction on Design Automation of Electronic Systems,* vol. 22, No. 2, pp. 35–45, March 2017.

Dr Thomas Weripuo Gyeera received a BSc degree in computer science and communications engineering from the University of Duisburg in 2005 and a Master of Science degree in computer and network engineering with distinction from the Sheffield Hallam University, UK in 2014. He received a PhD degree in computer science from the University of Sheffield UK in 2019. He has been working on using machine learning and adaptive algorithms for proactive cloud computing resources monitoring and adaptation. He has worked in the industry for Ford Motor company and Thales Group as an application engineer from 2006 before going for a postgraduate study. His major interest 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 Director of Teaching Quality in the Department of Computer Science, University of Sheffield, where he leads object-oriented research in verification and testing, type theory and language design, development methods and precise notations. He is also the director of the undergraduate computer science program at the University of Sheffield.

Dr Mike Stannett is a Senior Lecturer, and a member of the Verification and Testing Research Group, in the Department of Computer Science at Sheffield University. He has a wide range of interests, including: the verification and testing of unconventional and heterotic computing systems; autonomic cloud computing platforms; the use of formal modelling techniques to generate new understandings of physical systems; and computational modelling of macroeconomic systems. He is a member of Computing in Europe (CiE) and the European Association for Theoretical Computer Science (EATCS), and has previously served as a member of the London Mathematical Society's Computer Science Committee.