Kalman filter based prediction and forecasting of cloud server KPIs

Thomas Weripuo Gyeera [1], Anthony J.H. Simons [2], and Mike Stannett [2]

[1]The University of Sheffield
[2]Affiliation not available

October 30, 2023

Abstract

Cloud computing depends on the dynamic allocation and release of resources, on demand, to meet heterogeneous computing needs. This is challenging for cloud data centers, which process huge amounts of data characterised by its high volume, velocity, variety and veracity (4Vs model). Managing such a workload is increasingly difficult using state-of-the-art methods for monitoring and adaptation, which typically react to service failures after the fact. To address this, we seek to develop proactive methods for predicting future resource exhaustion and cloud service failures. Our work uses a realistic test bed in the cloud, which is instrumented to monitor and analyze resource usage. In this paper, we employed the optimal Kalman filtering technique to build a predictive and analytic framework for cloud server KPIs, based on historical data. Our k-step-ahead predictions on historical data yielded a prediction accuracy of 95.59%. The information generated from the framework can best be used for optimal resources provisioning, admission control and cloud SLA management.

Kalman filter based prediction and forecasting of cloud server KPIs

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

Abstract—Cloud computing depends on the dynamic allocation and release of resources, on demand, to meet heterogeneous computing needs. This is challenging for cloud data centers, which process huge amounts of data characterised by its high volume, velocity, variety and veracity (4Vs model). Managing such a workload is increasingly difficult using state-of-the-art methods for monitoring and adaptation, which typically react to service failures after the fact. To address this, we seek to develop proactive methods for predicting future resource exhaustion and cloud service failures. Our work uses a realistic test bed in the cloud, which is instrumented to monitor and analyze resource usage. In this paper, we employed the optimal Kalman filtering technique to build a predictive and analytic framework for cloud server KPIs, based on historical data. Our k-step-ahead predictions on historical data yielded a prediction accuracy of 95.59%. The information generated from the framework can best be used for optimal resources provisioning, admission control and cloud SLA management.

Index Terms—Kalman filtering, machine learning, adaptive filters, virtual infrastructure network, and cloud computing

1 Introduction

Historically (from the 1960s onwards), IT businesses were run on mainframe computers, but in the 1990s both mini/personal computers and x86 servers became major competitors. The x86 server dominated until the early 2000s when the technique of virtualization began to offer more flexibility in running thousands of workload from data centers. The cloud concept, itself traceable to John McCarthy in the early 1960s, only became popular in the mid-2000s. McCarthy envisaged that computers would in the future be organised like utilities forming the backbone of big businesses [1]. Today, modern businesses have adopted the model due to its low initial investment requirements for IT infrastructure (both soft- and hardware) and the flexibility it offers to consumers, both in terms of on-demand provisioning of resources and also the number of IT professionals required for start-up [2].

Nearly half a century after this vision of cloud computing, the evolution of the IT industry has seen widespread adoption of the cloud business delivery model. Over 58% of big businesses across the world now utilize the cloud. According to a recent Gartner report, companies were spending annually up to $175bn on cloud business services, with this figure projected to double by 2020 [3]. The operating profit generated by AWS (S3) alone in provisioning corporate cloud space exceeds $7.3bn [4].

The adoption of cloud business services is driven largely by cost, together with the flexibility with which cloud resources can be provisioned and deallocated. These huge benefits cannot be overemphasized except insofar as no technology is without limitations. As shown in Table I, recent outages of major cloud service providers (CSPs), including AWS, Gmail, Yahoo! and Microsoft Azure have had significant consequences for both providers and consumers.

The idea of monitoring and adaptation, which is the main focus of our research, is motivated by consideration of these outages and disruptions. We argue that it should be possible for cloud service providers to offer proactive monitoring and adaptation in real time, which predicts a potential problem before it occurs, rather than reacts to it after the fact. To give just one example, adaptation-oriented monitoring might potentially have allowed the prediction of the 6-hour AWS outage of March 2017, which resulted in many East Coast businesses remaining offline for a whole day [5].

We regard the problem of monitoring and adaptation as one of prediction, detection, synthesis/analysis and adaptation. Forecasting enables the root cause of a potential problem to be determined proactively and mitigated. The prediction of resource allocation or consumption, for example, can allow IT resources to be dynamically provisioned while avoiding over- and under-provisioning. Here, over-provisioning means that resources are made available for consumption but are not used to capacity because of limited requests or workload fluctuations. Under-provisioning means that more IT resources are requested than can be allocated by the service provider. An efficient and effective predictive tool can warn a provider of the potential of this scenario developing. The capability of detection can enable a proactive tracing and identification of a potential problem, as well as the planning required for its appropriate mitigation.

Our approach involves using Kalman filters as a training algorithm on the relevant data sets. In making this choice we considered the accuracy of prediction, linearity, and the number of features and parameters of this vs. alternative approaches. The Kalman filter was first described in the 1960s [6]. Originally designed for navigation of spacecraft, it has gained wider application in many fields of engineering, computing and economics, and is widely considered

TABLE I. Downtime, impact and root cause of various outages.

Provider	Date of Outage	Downtime	Impact in terms of Cost	Users Affected	Root Cause
Amazon Web Services	03/2017	6 Hours	$\approx$ \$1B	148,213	Command line error
Yahoo! Mail	04/2011	6 Hours	$\approx$ \$1B	1M	Server upgrade
Google Cloud Gmail	02/2011	2 days	$\approx$ \$1B	150,000	A software bug
Amazon EC2 & RDS	04/2011	4 days	$\approx$ \$1M	$\approx$ 1M	Problems with EBS replication
Microsoft Hotmail	12/2010	4 days	$\approx$ \$1B	17,000	Load balancing issues
Amazon Web Services (AWS) S3	07/2008	8 Hours	$\approx$ \$1B	$>$ 100000	corrupted X-server messages and under-provisioning
WMware Cloud Foundry	03/2011	2 days	$\approx$ \$1M	$>$ 100000	loss of controller connectivity

a suitable model for linear system dynamics. An extended version of the filter was later developed for non-linear system properties [6] [7]. For non-Gaussian problems, so-called Markov and Hidden Markov models are appropriate, while for problems with high dimensional data sets other algorithms (e.g., MDA and PCA) are generally suitable [8]. However, the Kalman filter has the advantage in that it provides high quality estimates with relatively low computational overheads [6] [9].

1.1 Contributions

For the work reported in this paper we sampled data on 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 fundamental server KPIs that are usually defined as measurable metrics in an SLA document:

1) Server hits per second
2) Throughput (average requests per second)
3) Bandwidth consumption in Mbps
4) CPU consumption in percentage
5) Average response time in seconds
6) Bytes received per second
7) Average latency

We conducted rigorous training and validation experiments with the optimal Kalman filtering algorithm on our generated data set yielding a good performance with a prediction accuracy of 95.51%. The approach also presents the capabilities of being able to forecast into the future of the key performance indicators characterizing virtualized servers and the applications deployed on them. Our approach presents a unit $k-$step ahead prediction and forecasting of cloud resources based on past and current observations using the optimal Kalman filtering techniques. We defer the discussion emphasizing the justification and relevance of these metrics in the application design, implementation, testing and integration process until the section on related work.

1.2 Structure of the paper

The rest of the paper is organized as follows. We present previous research work close to our approach in section 2. In sections 3 and 4 we present our conceptual framework and define the underlying problem using the mathematical constructs of the Kalman filtering algorithm. Section 5 provides a description of the experimental testbed, tools and procedures we used in sampling the datasets for training and validating the model. Section 6 covers a critical review of our experimental results. A practical application of our framework is discussed in a case study in section 7 and our main conclusions are presented in section 8.

2 Related work

This section presents previous research activities conducted in the area of monitoring and adaptation of cloud computing resources related to our work.

Nilabja *et al.* [10] developed a model-predictive algorithm for efficiently forecasting and autoscaling of workloads in the cloud. Their techniques predict workload in advance in order to ensure that resources are optimally provisioned and deallocated based on the volume of workload influx. In their approach the second order autoregression moving average algorithm (ARMA) attempts to forecast workload for a future time horizon and determines the response time based on the current workload. The predicted workload, available hardware, service demand as well as the user think time in executing their actions serve as the inputs for determining the overall response time for the application. The next stage is to determine an optimal resource provisioning strategy through the solution of a utility cost function where resources are increased if the predicted workload is high and less resources are provisioned if there is a reduction in the workload.

Our approach is quite similar to this work in the sense of the step-ahead prediction horizon that determines how much of the workload the system expects in the future without violating the SLA parameters. We focus on using the Kalman estimator which is robust against noisy sampling and for non-linear properties the unscented Kalman filter allows the linearization of the mean and covariances [11]. This makes the Kalman filter a suitable choice especially for both linear and non-linear data with Gaussian distributions.

The technique of the autoscaling method is only suitable for systems with simple linear properties and this may not be suitable in highly dynamic environment like cloud data centers. In addition, the approach with the Kalman filtering technique is a good choice since it can be applied to both discrete and continuous value problem.

Colojani *et al.* [12] [13] presented a real-time adaptive framework toward adaptive, reliable and scalable cloud data gathering and monitoring. The adaptive algorithm defines model parameters at varying sampling intervals with the overall goal of maintaining a low-cost function on communication overhead while guaranteeing reliable data quality. In this algorithm, if the dynamics of the monitored system are generally stable, data sampling is done with larger sampling intervals and in a highly volatile systems' behavior the algorithm samples data much faster. This dynamic approach of real time big cloud data gathering and monitoring with variable sampling intervals is aimed at capturing all transient and long-term window of events characterizing the system.

Our approach exploits the benefits of predictive algorithms in defining baselines for cloud resources provisioning and monitoring. The reactive approach presented by Colojani *et al.* overburdens the system with high computational overheads as it keeps adjusting the sampling interval. Our approach determines the systems behaviour well in advance from previous resource usage patterns in order to optimally provision the amount of resources that the system would need in the future.

Kalvianaki *et al.* [14] [15] presented self-adaptive and self-configuring virtualized servers' CPU resource provisioning using Kalman filters. Based on feedback control theory the Kalman filter is integrated into the controllers depending on the system state inputs and outputs for tracking and updating resources utilization under variable workload conditions. In their work the Kalman basic controller is designed as a Single Input, Single Output (SISO) model which dynamically allocates and adjusts the percentage CPU utilization on the virtual machines. This method is further extended to the Multiple Input, Multiple Output (MIMO) principles of feedback theory in which the noise covariances between multiple VMs is exploited as a tuning parameter for adapting multitier applications provisioned on the virtual machine. The adaptive MIMO controller (APNCC) from this work integrates the self-configuring capability in addition to the dynamic allocation and adjustment of multi-tier application. Experimental results show that these controllers are very effective at tracking the percentage CPU allocation and utilization.This current work is closest to this work.

The key difference in our approach is that we focus more on proactive monitoring and adaptation in which the performance patterns of the application server KPIs can be analyzed in advance.

Reactive adaptation to the constantly changing demand for resources is no longer effective, considering the volume, veracity, velocity and variety (4Vs model) of data processed by data centers. We argue that the behavioural patterns and KPIs characterizing virtualized servers, networks and database applications can best be studied and analyzed with predictive models such as the LMS regression and the optimal Kalman filter estimators. With an approach using predictive analysis both long- and short-term predictions can be used as a guide for the provisioning and deallocation of cloud data center resources or for the general purpose of cloud data center resources management. Another direct benefit would be the prevention of over- and under-provisioning as well as the prevention of over- and under-utilization of cloud computing resources. A disadvantage of the reactive approach presented in [15] is that the system is overburdened with computational overheads as it adapts to new changes.

Sackl *et al.* [16] developed the LTD, SLTD, TJ, AREA and doubles models for characterizing bandwidth consumption fluctuations for determining user quality-of-experience (QoE). The models define new KPIs in order to establish the impact on bandwidth fluctuations on the user experience. In the LTD model, a fraction of time is set to be able to observe the bandwidth below the downlink connection of the network called BDW. This time period is used to map a new function between the mean opinion score and the fluctuating bandwidth. The selective throughput duration (SLTD) model takes into consideration the time frame the bandwidth drops are unnoticed by the user whereas the TJ model uses the moving average with low frequency sampling so that the bandwidth reference value remains noticeable. Both the AREA and the double models are constructed in the same way as the LTD. A threshold on the downlink bandwidth is defined as a gap that accounts for larger bandwidth fluctuations.

A model driven engine for cloud resources scaling based on the Amdahl's law is presented by Gandhi *et al.* [17]. Their work employs the optimal Kalman estimator to dynamically predict workload fluctuations and adapt by either horizontally or vertically scaling the provisioned resources. The model characterizes the workload fluctuations and determines the optimal scaling methods using the Kalman filtering techniques. Finally, Hu *et al.* [18] presented a framework based on statistical learning theory in constructing models using the Kalman smoother and the support vector regression algorithms. Prediction accuracies with their approach are evaluated to be higher than those using techniques that employed auto regression, back propagation neural networks, and canonical support vector regression algorithms [19] [20] [21] [22].

Our approach in this work applies the state space version of the Kalman models in building predictive models that can be used for analyzing and forecasting cloud resources allocation and consumption. To further state the benefits of our approach, we present in section 6.4 a detailed comparative analysis using two machine learning algorithms (stochastic gradient decent (SGD) and boosted decision tree) from previous work in [23] and a reactive approach developed by [24].

3 Conceptual Framework

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

(1) The *monitoring stack* provides a dashboard that showcases various aspects of monitoring. The pay-per-use monitor depicts metrics showing 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 service level agreement (SLA). For the purpose of enforcing an SLA contractual agreement, the SLA monitor can also display metrics such as the availability of resources provisioned within the cloud. The fail-over/infrastructure monitoring stack is quintessential to characterising transient and general network issues. This component is generally required for detecting failures and anomalous behaviours so that they can be mitigated before the virtual network or application server becomes unavailable.

(2) The *adaptation stack* implements the filtering/machine learning algorithm used to learn from the behavioral patterns of the virtual infrastructure network and application server KPIs. For instance, the implementation of an adaptive filtering algorithm or an ensemble learning algorithm (e.g.

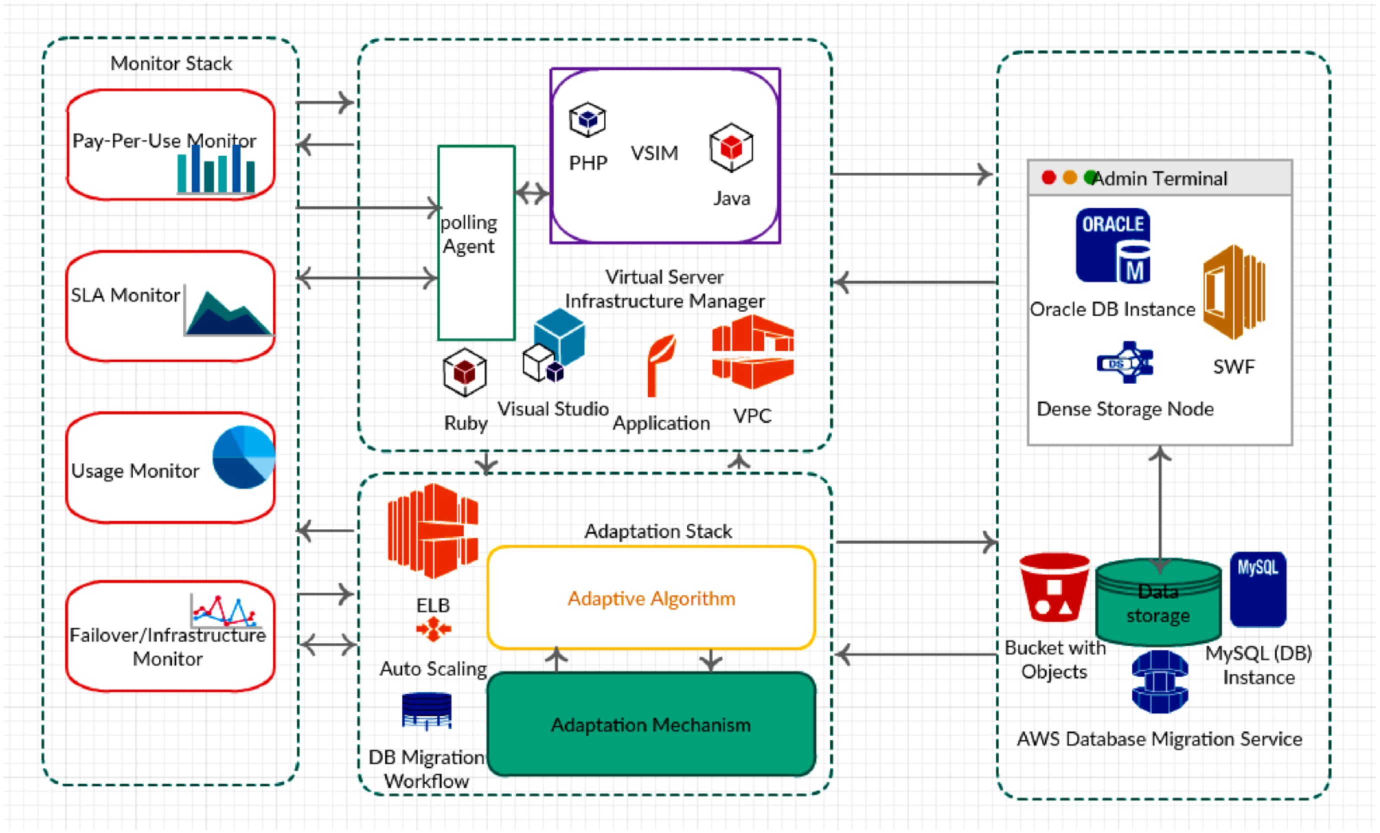


Fig. 1. Conceptual framework.

BDT) helps predict future resource consumption patterns. It is this which allows a suitable adaptation strategy to be enforced, e.g., elastic load balancing, auto-scaling of pooled resources, or the migration of a DB workflow.

(3) The third stack contains our application server and the virtual infrastructure network nodes to be monitored. We implemented a web service platform that allows a huge number of robot users to browse and make purchases from the application. The content managements and the metrics polling techniques (push or pull) are all part of the application stack.

(4) The fourth stack provides the admin user console that interfaces with the application for observing the different metrics of the framework. The admin terminals administer the databases and the storage required for the operation of the system.

4 Problem Definition According to the Kalman Filter Algorithm

We illustrate in this section how the problem of cloud service data and business activities can be formulated as a Kalman filtering problem in conjunction with subsequent tracking of the allocated and consumed resources' signals.

We can think of a cloud service platform as a dynamic system which allocates resources in response to signals concerning the consumption of, e.g., bandwidth, CPU, resources, storage, and memory. We can also assume that the population of consumer services is sufficient to generate significant moment-to-moment variations in resource consumption (the distribution of which, by the Central Limit Theorem, can be considered essentially Gaussian). Under these conditions it is reasonable to model the consumption of resources using the output signals from random processes (though these may also contain unwanted noise) [6]. This allows us to model the cloud allocation system in terms of a Kalman filtering problem involving the filtering, prediction and forecasting of cloud resource provisioning.

For a zero-mean random Gaussian process the following random input and output signal vectors and matrices are defined for all iterations of the filtering and forecasting processes, under the generalized state space model [6] [7] [25]:

$$\begin{cases} x_{k+1} & = T_k x_k + R_k \omega_k + \delta_k \\ y_k & = H_k x_k + \nu_k \end{cases} \quad (1)$$

for all $k \geq 0$.

The variable x_k denotes the state information of the system within a finite-dimensional vector space p. The known initial input signal of the system is defined as ω_k. The observed or measured output signal at time-step k is denoted y_k. T_k, and H_k represent state and measurement transition matrices respectively while R_k is the matrix that derives the dynamics of the input to the system.

The last terms in (1) represent the contributions of noise, where δ_k is the process noise and ν_k the observation noise.

Derivation [6] [25]: Assume the state estimate at time interval $k+1$ is desired, given $(y_0, y_1, ...y_k)$ manifold measurements, then based on Bayes' rule, the step-ahead equation for the state estimate x_{k+1} can be derived from first principles as follows. The prediction of the states at

$k+1, k+2...k+n$ based on the condition that a measurement value has been made at the previous time interval of k is computed as the expectation x_{k+1} conditioned on y_k observations:

$$\begin{aligned} \hat{x}_{k+1|k} &= E[x_{k+1}|y_1, y_2, ...y_k] && (2) \\ &= E[x_{k+1}|y_k] && (3) \end{aligned}$$

Substituting from (1) into (3), which is obtained by applying Baye's rule, the predicted state is computed as follows:

$$\begin{aligned} \hat{x}_{k+1|k} &= E[T_k x_k + R_k \omega_k + \delta_k | y_k] && (4) \\ &= T_k E[x_k|y_k] + R_k \omega_k + E[\delta_k|y_k] && (5) \end{aligned}$$

Given that the input signal ω_k is known and the noise factor δ_k has zero mean, the derivation can be completed by reducing (4) to the same form as (1), viz.:

$$\hat{x}_{k+1|k} = T_k \hat{x}_k + R_k \omega_k \quad (6)$$

The H_2-norm transfer matrix characterizes the transient effects of the input signal on the measured output signal. The H_2-norm and the optimal Kalman filter are considered to be similar and are used interchangeably in this research. In the state space model, the standard equations for both the state variables x_k and observation processes (output signal) y_k are described as in (1). However, a greater challenge is that the means and covariance matrices are unknown parameters from the start of the observations. To circumvent this problem, the maximum likelihood estimation (MLE) model is employed to determine the best fits for these parameters. Derivations of the MLE model's mathematical foundations can be found in [26].

4.1 *A priori* filter design

Consider the state space model of a zero-mean random variable obtained from the standard state equation (1), which includes uncertainty associated with the state estimation and the noise resulting from the measured output. Given a sequence of measured outputs $(y_0, \ldots, y_{k-1})$ the states of the input variable can be projected in advance. This therefore allows the formulation of the equation for the predicted optimal state estimates as follows. If the noise in the initial measurement is zero, then using (3) allows the expectation of the observed signal to be computed as in (7) – this is the first term of the observation equation in (1). The "innovation", e, is defined to be the difference between the observed output and the expectation of the measured output.

$$e = H_k x_k \quad (7)$$

The covariance of the expectation is computed from the transition matrices characterizing the observations made and the state estimation covariance (P). The expectation E for the states covariance is given in (8)

$$E = HPH^T \quad (8)$$

To derive the equation for the covariance matrix P for the states estimation, assume that a step-ahead estimate is made at time interval $k+2$ for y_k manifold measurements.

If the state estimates at $k+2$ and the noise are further assumed to be uncorrelated, the mean-squared error between the two random variables x and $\bar{x}$ then the covariance is computed using Bayes' rule as follows [25]. Recalling that

$$E\tilde{x}_k = E(x - \bar{x}) \quad (9)$$

we have

$$\begin{aligned} &\hat{P}_{k+2|k+1} = \\ &= E[(x_{k+2} - \hat{x}_{k+2|k+1})(x_{k+2} - \hat{x}_{k+2|k+1})^T] + E[\delta_{k+2}|y_k] && (10) \\ &= T_{(k+1)} E[(x_{k+2} - \hat{x}_{k+2|k+1})(x_{k+2} - \hat{x}_{k+2|k+1})^T | y_k] && (11) \\ &= T_k E[(x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})^T | y_k] T_k^T + E[\delta_k \delta_k^T | y_k] && (12) \end{aligned}$$

$$P_{k+2|k+1} = T_k P_{k|k} T_k^T + \Delta_k \quad (13)$$

where $\hat{P}$ is the estimated covariance of the states. The total uncertainty resulting from the observation processes can be obtained by summing the weights of the covariance of the expectation and the covariance due to the measurement noise derived from (7), (8) and (12). From the derivations made for (7), if the model is assumed to be a discrete Reccati equation (DRE) [25], then the *a priori* (predicted) mean value of the state estimates x_{k+1} can be given as

$$\hat{x}_{k+1} = T_k \hat{x}_k + K_{g,p} (y_k - H_k \hat{x}_k) \quad (14)$$

From (7) and (8)

$$\begin{aligned} Z &= R + E && (15) \\ K_{g,p} &= PH^T Z^{-1} && (16) \end{aligned}$$

where R is the covariance of the measurement noise and it is assumed that the initial values of the quantities $\{T_k, R_k, H_k, \Delta_k\}$ are known or can be obtained through *a diffuse prior* using the MLE model. $K_{g,p}$ is defined here as the predicted Kalman gain that can be computed as follows. Assume the covariance matrix of the *a priori* state estimation is represented as P for k manifold observations:

$$K_{g,p} = T_k P_k H_k Z_{e,k}^{-1} \quad (17)$$

Modelling the system in such a way that it obeys the recursive discrete equation in (8), then $Z_{e,k}$ the predicted estimate of the covariance P, is based on the estimation of the instantaneous error in the *a priori* state estimates.

The steps involved in the *a priori* (predicted) of the mean states and the covariance estimates can be summarized as follows:

1) For a *diffuse prior* estimate, the values of the quantities $\{T_k, R_k, H_k\}$ can be determined from the Maximum Likelihood Estimator (MLE).
2) Compute the expectation of the states as a product of the state transition matrix and the random variable.

3) The covariance of the expectation is then computed from (3) after determining the covariance matrix from the MLE.
4) The total uncertainty associated with the predicted mean signal variable is the sum of the covariance expectation and the noise resulting from the dynamics of the system.
5) Once the total uncertainty of the system is modeled, the predicted Kalman gain can be calculated from (16)) as a function of the covariance matrix, the state transition matrix and the Reccati equation.
6) Having computed the predicted Kalman gain, the mean value of the next state estimate prediction can be determined from (13).
7) The final step in the predictive stage is to repeat the process to calculate the mean value of the predicted covariance from equation (12).

4.2 *A posteriori* (filtered) state estimates filter design

In the predictive model described above, all the computational steps can be summarized in two major operations. In the first iterative step, for a given sequence of observations $(y_0...y_{k-1})$, the states of the system can be predicted in advance. The errors arising as a result of the prediction can then be computed in order to correct the projected states estimates. The predicted states and the covariances form the basis for the measurements to be corrected and updated (corrective measure). In other words, based on the projections on the random state variable x_{k+1}, how much do these predictions differ from the actual measurements? The desired goal in the *a posteriori* stage (filtering) is to find a better estimate for the current state, based on the latest measurement. It is always then the case that given the current state of the random variable, it is feasible to predict the next state based on the known distribution at the given time. The update equations are stated here without proof(see [6] for proof) as follows:

First the corrected (filtered Kalman gain) K_{gf} is computed as

$$K_{gf} = P_{k|k-1} H_k^T S_k^{-1} \tag{18}$$

where S_k is the covariance arising from the observations as given in (19).

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{19}$$

With the filtered Kalman gain, the *a posteriori* updated state estimate, $x_{k|k}$ is given by

$$x_{k|k} = x_{k|k-1} + K_{gf} \bar{y}_k \tag{20}$$

The updated estimate covariance (*a posteriori*) is then given by

$$P_{k|k} = (I - K_{gf}) P_{k|k-1} \tag{21}$$

where I is the identity operator.

The steps involved in the *a posteriori* estimation are thus relatively few in number:

1) Compute the residual from the observations which is defined as the innovation (7).
2) The covariance resulting from the measurements is computed using (19).
3) The filtered Kalman gain is then calculated from equation (18) and used as a tuning parameter of the measurement innovation as defined in $\bar{y}_k$. The measurement innovation here is defined as the difference between the forecast output measurement and the actual measured output at the time interval k.
4) Equation (20) can then be used to compute the *a posteriori* state estimate.
5) The final step is to update the covariance estimate using (21).

```
INITIALIZATION
    1)  $\hat{x}_{0|0} = x[0] = 0$
    2)  $\hat{P}_{0|0} = x[0]x[0]^T = [0]$
PREDICTION
Do while $k \geq 1$
    1)  $\hat{x}_{k+1} = T_k \hat{x}_k + R_k \omega_k$
    2)  $P_{k+1|k} = T_k P_{k|k} T_k^T + \Delta_k$
    3)  $K_{g,p} = T_k P_k H_k Z_{e,k}^{-1}$
CORRECTION
    1)  $\hat{x}_{k+1} = T_k \hat{x}_k + K_{g,f} (y_k - H_k \hat{x}_k)$
    2)  $P_{k|k} = (I - K_{gf}) P_{k|k-1}$
    3)  $K_{gf} = P_{k|k-1} H_k^T S_k^{-1}$
```

Fig. 2. Summary of the Algorithm

5 EXPERIMENTAL SETUP AND PROCEDURE

We present a detailed description of the experimental set up and the implementation of the conceptual framework presented in section 3. We designed a real-world test bed in Microsoft Azure cloud having six virtual servers distributed in the US East Coast, US West Coast, and European regions. We then implemented a web service platform merchandising a selection of products. The goal here was to simulate a high-volume of concurrent virtual users browsing for products on the platform, including subsequently proceeding to make purchases and then exit the platform.

We migrated the application into each of the six logically separated virtual servers in order to enforce a fail-over mechanism; these servers were then networked in Azure so they could communicate with each other using a common domain controller. For monitoring purposes, we interfaced the web platform with both Google Analytics [27] [28] and Azure Application Insights [29] for live observation of the KPIs (for example, we can observe the dynamic navigation patterns of the virtual clients using Google Analytics, while Azure Application Insights are suitable for observing live server metrics).

We distributed the virtual load influx on the application server and the entire infrastructure network using JMeter [30] [31] in the form of Java threads to concurrently browse the web service platform and purchase the products on display. The next section gives a detailed description of the key experiments conducted on the application server and the virtual infrastructure network for collecting data sets for the purposes of training and testing the models. Detailed configuration steps for these virtual servers on Azure are left out in this work for the sake of brevity.

5.1 The experiments and data collection procedure

We performed four main experiments in simulating different user scenarios on the web service platform in order to generate and collect data for our model. A summary of the statistics (mean, median, 90% line, 95% line, 99% line, min and max) of these experiments are shown in Fig. 3.

Experiment 1 (constant server workload influx): We submitted a constant number of N clients workload to be processed by the application server and the virtual infrastructure network to emulate client-server interaction. The server was configured with no ramp-up period by setting the start time to zero. We ran the experiment on JMeter for 12 hours before decreasing the server load influx to zero. Fig. 3a displays the results of this experiment, showing the average response time for the server and the virtual infrastructure network to construct responses and send them to the JMeter client.

Experiment 2: (variable server workload influx): The objective in this experiment was to use the concepts of pacing and think-time in simulating real time user behavior. Web clients' requests in the real-world usually have some delay in executing their actions when browsing a web application.

"Think-time" refers to the the time it takes a customer to navigate or perform an action on the page; we simulated it by applying the JMeter Stepping Group controller. To configure the parameters of JMeter for this experiment, we employed a constant number N of virtual users with a ramp-up period of 65 seconds to delay the processing of clients' request for the specified period. The server ran the experiment for 6 hours by constantly increasing the workload in every 15 minutes until the maximum load capacity is reached. On reaching the maximum load, the server was configured to delay the experiment for 5 seconds while keeping the server idle. After the expiration of the ramp-up period virtual users were then allowed onto the application for a period of 5 hours before decreasing the server load influx. For every minute we decreased the load by removing 10 virtual users from the server until the load reached zero. Fig. 3b displays the results of this simulation indicating a graph of the server response time that built-up at constant rate to the maximum load before it gradually dropped at a constant rate back to zero.

Experiment 3: (random server workload influx): The scenario simulated here mirrors the use of a server designated for random work, in which case the load to be processed may not be known in advance. We employed the JMeter random timer to simulate this user behavior. The uniform random timer allows the workload to be added to the server in a random fashion, and these loads are processed in any order they arrive. The intention here is to demonstrate how a server responds to random user activities. Fig. 3c presents the results of the simulation with the random workload influx.

Experiment 4: (a mixture of random and constant server workload influx): In this experiment we simulated a mixture of both a constant workload and then a random server workload influx. The idea here is that a server could be designated for a constant workload processing but may be required to handle a random amount of workload influx.

For this experiment we initially distributed a constant number N of virtual users to the web platform to be processed. After reaching the N-user peak load, the number was decreased to minimum value 100 virtual users before a random workload load was added to the server. We employed the JMeter ultimate thread group to emulate this client- server interaction. As shown in Fig. 3d, for the constant load influx, the experiment was run for 4 hours. Then for another 4 hours, a low 200 but constant head of workload was processed and then the server was again configured to process virtual users that were distributed randomly across the server node. The response time graphs are shown in Fig. 3d.

These four main experiments were conducted repeatedly for 10 runs each in generating and measuring the KPIs characterizing the application server and the virtual infrastructure network. In addition to the response times from these experiments, we also measured latency, bandwidth fluctuation, server hits per second, %CPU utilization and throughput (requests per second) as part of the process for building the predictive models.

6 Critical analysis and evaluation

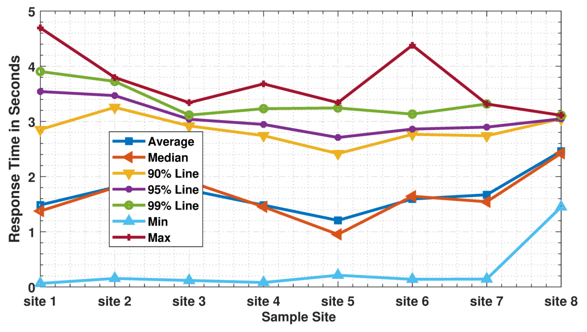
We present in this section detailed empirical results based on the evaluations described above. From the experiments described in section 5, time series data sets on the key performance indicators (e.g. the average response time in seconds ,server hits per second and %CPU utilization) characterizing the server application and the Azure cloud virtual resources were recorded using the JMeter application.

This section also includes k-step ahead predictions and forecasts on the server KPIs (e.g the average response time and percentage CPU utilization). The question here is that based on the time series data on the metrics characterizing the application server, how well can the Kalman filtering techniques estimate the performance of the application server and the cross layer virtual infrastructure hosting the web service platform? In order to guarantee accurate results in the predictions and analysis, all the time series data sets were preprocessed and filtered as described in the next section.

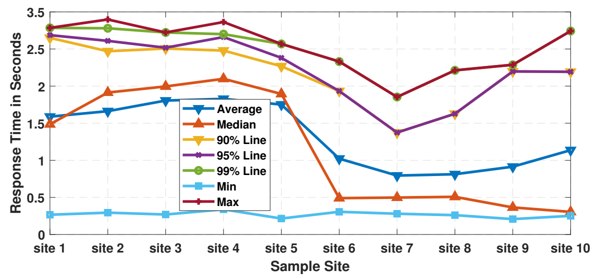
6.1 Data preprocessing and filtering

The preprocessing and filtering of sampled signals are fundamental steps in trying to reduce the noisy components that could potentially affect the accuracy of the models. In general, the noise attenuating a sampled signal turns out to have undesirable effects on the measured signal depending on the dynamics that drive the input to the output. There are a number of standard methods for data-filtering depending on the problem domain.

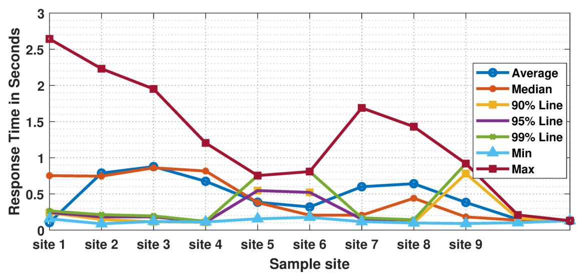
Specifically, we designed a low pass IIR filter [32] [33] [34] for use under different parametric configurations of



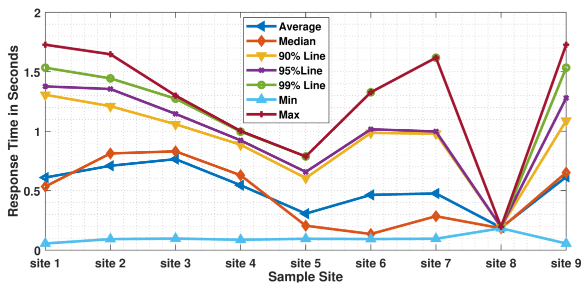
(a) Expt. 1: Constant workload



(b) Expt. 2: Steady increase



(c) Expt. 3: Variable load



(d) Expt. 4: Mixed

Fig. 3. These figures illustrate the measured response times of the application server and the virtual infrastructure network for different server load characteristics and parameter configurations used in experiments 1–4.

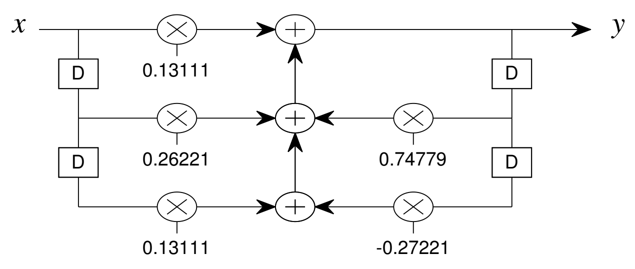
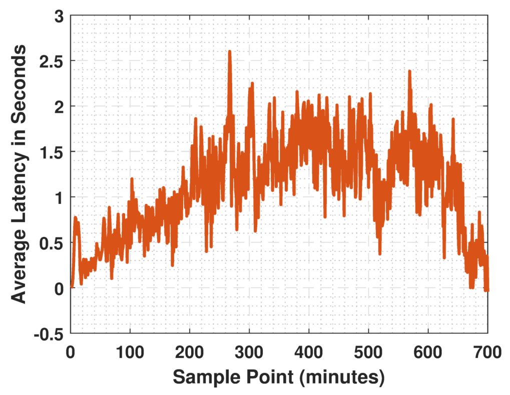
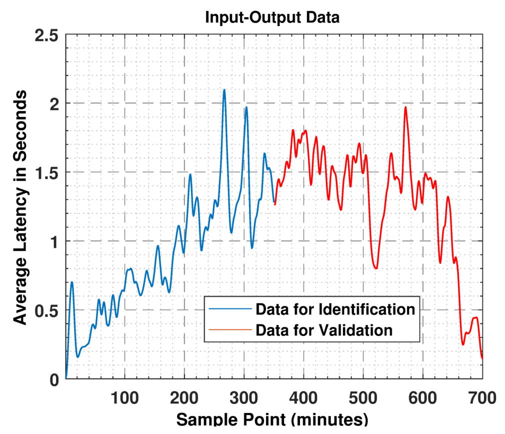


Fig. 4. Signal flow diagram depicting the filter coefficients $[a, b]$ of a 20th order low pass Butterworth filter at a 0.3 Hz corner frequency.

$$\begin{aligned}\Gamma_{a,b}(z) &= \frac{(b_0 + b_1 z^{-1} + \cdots + b_N z^{-N})}{(1 + a_1 z^{-1} + \cdots + a_M z^{-M})} \\ &= \frac{(0.131 + 0.262 z^{-1} + 0.131 z^{-2})}{(1 + 0.748 z^{-1} - 0.272 z^{-2})} \\ &= \frac{(0.131 z^2 + 0.262 z + 0.131)}{(z^2 + 0.748 z - 0.272)} \qquad (22)\end{aligned}$$



(a) Plot of the sampled raw signal of the average latency in seconds.



(b) Final filtered output using the 20th order low pass filter.

Fig. 5. These figures illustrate the sampled raw signal of the average latency in Fig. 5a and the evolution of the filtered signal as shown in Fig. 5b.

filter coefficients, $[a, b]$, while tweaking the order from 2 to 20 at a corner frequency of 0.3 Hz. A plot of the signal flow diagram showing the evolution of the filter coefficients $[a, b]$ at 0.3 Hz cut-off frequency is shown in Fig. 4. The values of the coefficients of the IIR Butterworth low pass filter can be read off from the signal flow diagram depicting the evolution of the filter coefficients, yielding the following transfer function [34] [35] as shown in (22):

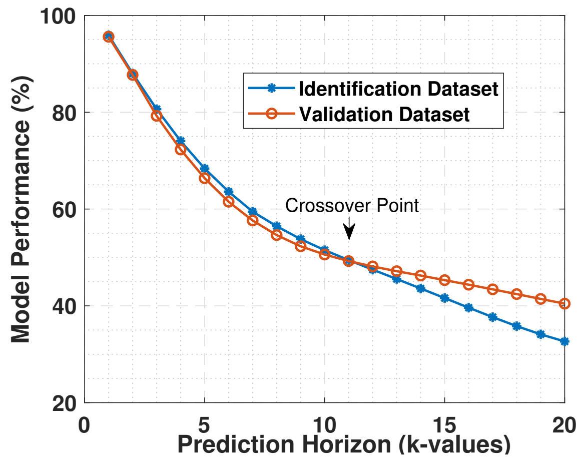


Fig. 6. Performance of the states space model with increasing prediction horizon. This indicates an exponential decay with a continuous rate of -0.085 per unit time of the model's performance as k increases.

Fig. 5a shows plots of the raw signal before filtering. In order to determine the noise inherent within the sampled signal we applied a half-bin discrete Fourier transform (DFT) to reveal the magnitude spectrum of the signal. The magnitude spectrum clearly reveals the first frequency components of the measured signal centered at zero of the DFT bin axis while the noise components can be seen spread along the rest of the DFT bins but due to lack of space we are unable to display plots of the frequency spectrum analysis. These noise components seen from the frequency spectrum analysis strongly suggest that the effects of noise components will be noticeable when applying Kalman filters to the make k-ahead predictions.

We initially passed the raw data sets through different filtering techniques (e.g. low pass, bandpass, stopband and the elliptical filters) to suppress the noise components and the various output signals were compared before settling on a 20th order Butterworth low-pass filter with a 0.3 Hz cut-off frequency. Even though this filter with a high order is generally characterized as computationally intensive, the accuracy of the predictive experiments depends much on the total elimination of noise and outliers from the data set. A final output plot of the filtered signal with the aggregated coefficients which serves as the input signal for the predictive model is shown in Fig 5b.

6.2 Model description and parameterization

In the experiments described in section 5 and the mathematical constructs defined in section 4, we used recorded past data on the KPIs to predict and forecast the performance of the application server up to a finite future time horizon.

This section presents the predictive analytic framework based on the optimal Kalman filter estimators described in section 4. The key performance metrics measured on the virtual infrastructure network and the application server include: average response time, %CPU, server hits per second, average throughput, latency, bandwidth consumption and the number of bytes per transaction. The historical data from these observations served as the input for building the model to fit the data set after the filtering process described in the previous section.

From the experiments designed for the identification of the state space model covered in section 4, half of the data set was used for identifying the model and its parameters, and the remaining data was reserved for the system validation. From the start, the system and its parameters T_k, H_k, and γ are unknown. The observations at different sampling intervals, $\{\ y_0,\ y_1,\ y_2,$ and $y_k\ \}$ are the only known parameters for the design of the model after the noise components of the sampled signal have been removed, where y_k is the measurement taken at the time interval k.

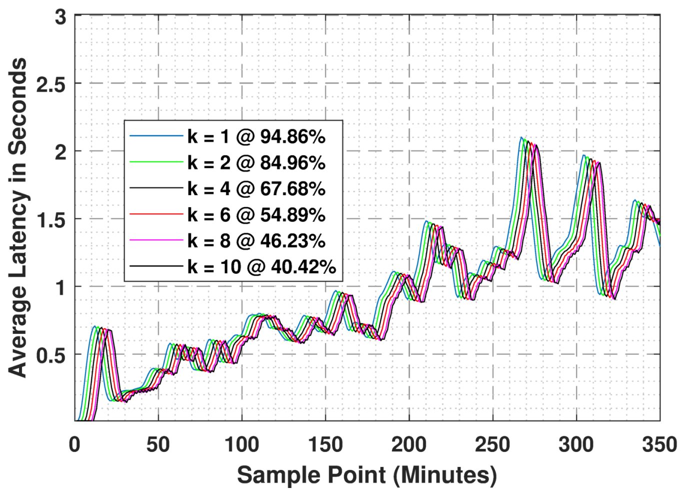
A *diffuse prior* based on the maximum likelihood estimation on the system's parameters and its initial states, conditioned on the previous observations, helped establish the general model for fitting the data as described in section 4.

Feeding the server response time into the system identification toolbox in MATLAB, the following state space model has been identified as fitting the data set. The model equation (23) obtained based on the empirical observations of the server response time, and all the other metrics, satisfy the theoretical formulation of the problem domain in section 4 with the model parameters summarized in table II.

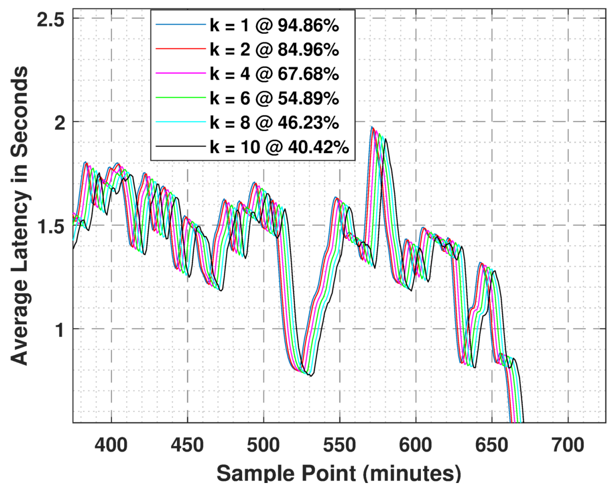
Equation (23) is a slight modification of the original formulation of the state space model (1). The original definition of the state space requires a multiplicative input variable with its transition matrix as a linear combination with the state's variable and its multiplicative transitions matrix as characterized by (1). The model built on the time series training set example does not require an input vector to drive the dynamics of the output variable. Therefore (23) describes a modification with emphasis on the noise as a result of the state estimates multiplied with a Gamma (γ) constant factor. The noise of the observations is characterized by only delta (δ_k) which is consistent with the definition of the observation equation (1).

Thus, the first states prediction in a unit time interval is a product of the state transition matrix, T_k and the current state x_k with additive noise components $\gamma\delta_k$, while the noise component of the observation signal is characterized using only δ_k. The observation equation is characterized by the design matrix H_k and the current state estimate x_k with δ_k additive noise. For the model to be used for predictions, the observation equation must satisfy the conditions $k \geq 0$ and the samples are Gaussian distributed with uncorrelated white noise.

In table II, the model parameters based on the observations of the time series data, depict the values of T, H and γ which remain constant once the initial conditions have been determined from the maximum likelihood estimation model as explained in section 4. The table also depicts the evolution of the covariance matrix determined for each state of the prediction and both the mean squared error (MSE) and the final prediction error (FPE) quantify the overall uncertainty of the estimation and measurement processes. The prediction efficiency expressed as a percentage determines how well the model fits the data sets. As shown in the table, the data set on the response time generated the best prediction with an average of 95.91% while the average percentage CPU utilization is predicted with at least an accuracy of

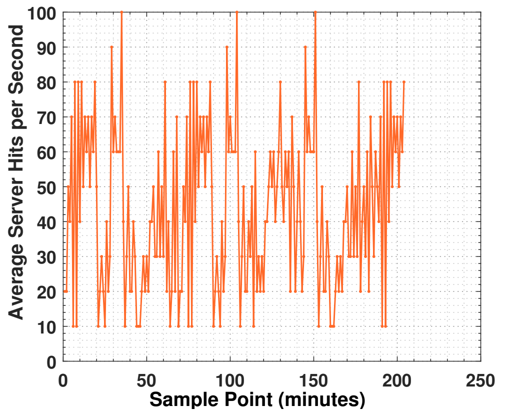


(a) Latency predictions at $k = 1, 2, 4, 6, 8, 10$ for the system identification.

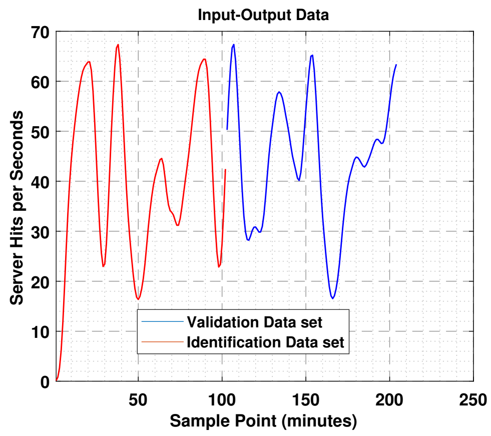


(b) Latency predictions at $k = 1, 2, 4, 6, 8, 10$ for the system validation.

Fig. 7. These figures illustrate the predictions at different future time horizons for both the system identification in Fig. 7a and validation data sets Fig. 7b respectively.



(a) Plot of the sampled raw signal of the server hits per second.



(b) Final filtered output of the server hits per second.

Fig. 8. These figures illustrate the sampled raw signal of the server hits per second in Fig. 5a and the evolution of the filtered signal as shown in Fig. 5b.

85.18%. The different measurements of the model focus are largely determined by the size of the data set, the larger the data set the better the prediction.

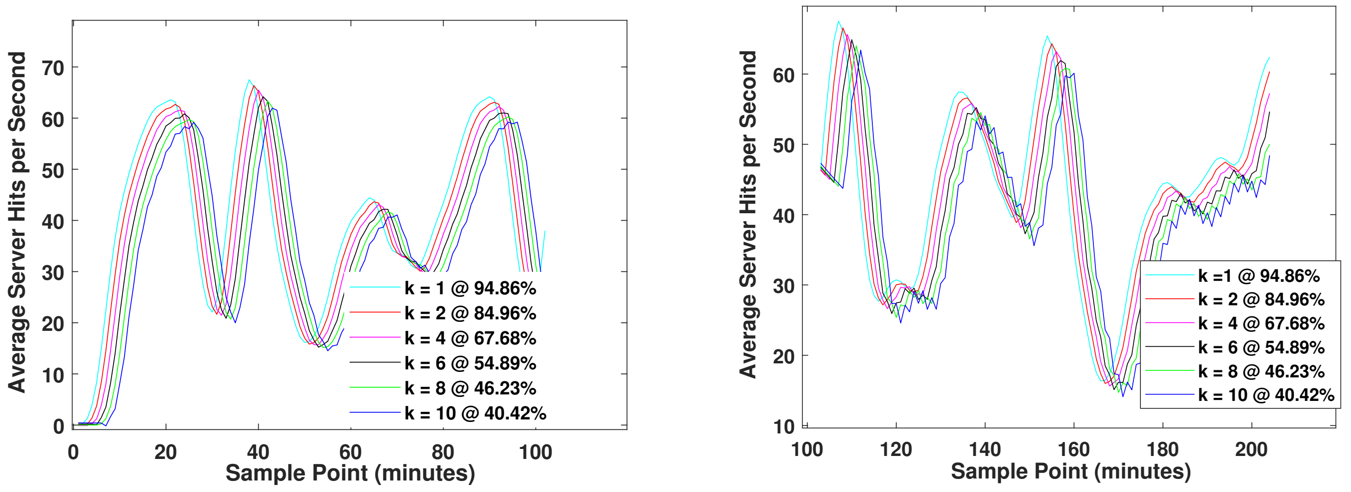
$$\begin{cases} x_{k+1} &= T_k x_k + \gamma \delta_k \\ y_k &= H_k x_k + \delta_k, k \geq 0 \end{cases} \tag{23}$$

TABLE II. Parameters for fitting the state space model after training

KPI	T	H	γ	Covariance Matrix	FPE	MSE	Efficiency
Response Time	0.9943	1	1.994	$\begin{bmatrix} 0.0000 & 0 & -0.0000 \\ 0 & 0 & 0 \\ -0.0000 & 0 & 0.0025 \end{bmatrix}$	3.62	3.611	95.91%
%CPU	0.9885	1	1.988	$\begin{bmatrix} 0.0001 & 0 & -0.0002 \\ 0 & 0 & 0 \\ -0.0002 & 0 & 0.0356 \end{bmatrix}$	17.19	16.58	85.18%

To effectively evaluate and score the performance of the state space model, predictions on different sampling time intervals were measured with the corresponding confidence bounds.

A plot of these iterations shows a continuous rate intercept exponential function cutting on the vertical axis at 100% with a continuous rate of decay at -0.085 per unit time. From Fig. 6, the function generalizing the model performance can be written as $P = 100e^{-0.085k}$, where P represents the confidence bounds in one run of the experiment, and k is the prediction or forecasting horizon. The value 100% in the equation represents the best prediction accuracy of the model. This can be observed from Fig. 6 as both curves cut the vertical axis at 100%. The prediction accuracy of 100% which also coincides with the value of $k = 0$ (is referred to as filtering), if $k > 1$ the signal is being predicted and when $k < 1$, a smoothing filter is being applied to the sampled signal. Our results confirm the definition of the terms filtering, smoothing and prediction as in system theory (see [6] [8] [25] for the definitions of these terms). The rule of thumb on scoring the state space model is that for larger data sets, the model behaves more stably and the predictions match both the training and validation data set.



(a) Server hits predictions at $k = 1, 2, 4, 6, 8, 10$ for the identification. (b) Server hits predictions at $k = 1, 2, 4, 6, 8, 10$ for the validation.

Fig. 9. These figures illustrate the predictions at different future time horizons for both the system identification in Fig. 9a and validation data sets Fig. 9b respectively.

The next section describes the application of the state space model to the time-series data sets measured for the server application running on the virtual infrastructure network.

6.3 k - Step Ahead Predictions with the Kalman Filter

The model described in the previous section was applied to the observation data sets to predict the bandwidth fluctuations, server hits per second, average response time, latency and average bytes per transaction characterizing the application server and the cross layer virtual infrastructure network. Details of the predictive analyses are given in this section.

In the experiments conducted for building the model, half of the signal length on the data set was separated for training the model while the remaining half for the model validation. In order to guarantee accuracy on the predictions, the noise component was filtered out with the 20th order low pass Butterworth filter. The plots shown in Fig. 7 display the predictions made on different k-step ahead time unit of the average response latency of the server application. As shown in Fig. 7a, the experiment started with a constant influx of 20 virtual users accessing the application concurrently as the server response time is being recorded with the JMeter application.

The predictions on the response times shown in Fig. 7a clearly indicate that as the number of virtual users increases there tends to be a lot of dynamics generated in the entire system leading to a steep increase in the average response time. A strong inference here is that as more virtual users simultaneously open sessions to the server application, the workload on the server tends to increase leading to a correspondence increase in the average response time. The plots shown in Fig. 7a and Fig. 7b indicate also that the predictions strongly follow the measured and filtered time series data but what these two plots display is the non-uniformity in the way the server processes the requests from the virtual users also suggesting that as more users queue with their requests, there appears to be a point where the rate of processing is non-deterministic. The rugged nature of both curves are direct indications of the dynamics that are generated within the server due to the high influx of work load.

Fig. 7b illustrates the predictions on the validation data set with the model using the remaining half length of the sampled server latency as virtual users concurrently open sessions to the application. The plots further reveal the validations of the model at different k values. The actual predictions indicate the validations at different $k(1, 2, 4, 6, 8, 10)$ future time horizons. The figures show that the predictions on the validation data set at different k values follow the measured signal but close to the end of the experiments there is a steep reduction in the average response time, indicating the workload on the server has been reduced.

The plots in Fig. 8 illustrate the evolution of the sampled server hits per second as a constant number of virtual users are deployed on the application server. The raw sampled time series server hits per second data set reveals the noise in the signal in Fig. 8a and by applying a 20^{th} order low pass Butterworth filter with a cut-off frequency at 0.3 Hz, the final filtered output is shown in Fig. 8b. The first half of the signal length was applied for the model training with the first k time unit prediction on the server hits per second at 95.59% accuracy.

The illustrations in Fig. 9a and Fig. 9b depict the server hits per second predictions at different time intervals. To identify the system we applied the first half length of the filtered signal to make the k-step predictions for different future time horizons. Fig. 9a indicates that the prediction follow the input signal quite for $k = 1, 2, 4, 6, 8, 10$ steps into the future.

The validation plots for the model are shown in Fig. 9b. The general trend is that the validation data sets give a higher prediction accuracy strongly suggesting that the model learns faster after the training phase. Fig. 9b shows a composite plot of different $k = 1, 2, 4, 6, 8, 10$ values of the server hits per second which do not deviate much from the plots from the system identification data set. We defer the application and usefulness of these predictions until the section 7 on case study.

6.4 Comparison of models and techniques

In order to quantify and state the benefits of our approach with the Kalman filtering techniques, we compared this work with previous works using two machine learning algorithms (the boosted decision tree(BDT) and stochastic gradient descent (SGD)) [23] and the reactive approach in [24].

For the approach with the boosted decision tree algorithm, we ran extensive machine learning experiments with the L2 regularization in characterizing the complexity of the model with the BDT algorithm. For this experiment a boosting with 100 trees in the subfunctional hypothesis space was sufficient to achieve an accuracy prediction of 98% at a learning rate of 0.2 and any further increase in the learning rate does not result in any performance improvement of the model. As shown in Fig. 10, the BDT performs slightly better than the Kalman filtering technique on the same data set since the KFT achieves it best performance of 95.51% with a final prediction error of 0.0443. Even though the prediction accuracy of the BDT algorithm is slightly better, the algorithm is most suitable for a data set with linear characteristics.

In a similar approach, comparing the Kalman filtering technique and the stochastic gradient descent approach, we ran the same machine learning experiments on the training and testing set examples. The SGD achieves its best performance with a 33.37% prediction accuracy at a learning rate of 1.0 with the Kalman filter having superior performance in terms of the prediction quality. The main advantage of the Kalman filtering technique is that it has a high degree of accuracy on data with Gaussian distribution and a finite mean.

In addition to comparing our approach with the above two machine learning algorithms, we again selected a reactive framework for VM capacity planning to compare with the Kalman filtering technique. Ardagna *et al.* [24] presented a two-phase framework in which the VM capacity allocation (CA) phase identifies the properties of VMs that are required to process the requests arriving from clients for every second while guaranteeing the response time in the SLA. The reactive model is set to dynamically adapt the VM resources to optimize the mean response time without violating the parameters of the SLA document. The load direction (LR) phase processes the total rate of executions of the web service requests and redirects workload influx from highly degraded resources to idles servers without increasing the mean response time.

Experimental results of predictions on the mean response time achieved a maximum prediction error of less than 20%. Thus, the two-phase techniques can achieve a prediction accuracy with the mean square error of less than 10%. The reactive method outperforms the stochastic gradient descent by far but its performance falls below the Kalman filter and the BDT algorithm with both techniques yielding 95.57% and 98% respectively. Another advantage of the Kalman filtering techniques is that for non-linear data extended Kalman filter [11] is most suitable. The extension allows the linearization of the current mean and the covariance of the estimates. Another disadvantage of the reactive models is that they require extra computational overhead as the system adapts to transient changes.

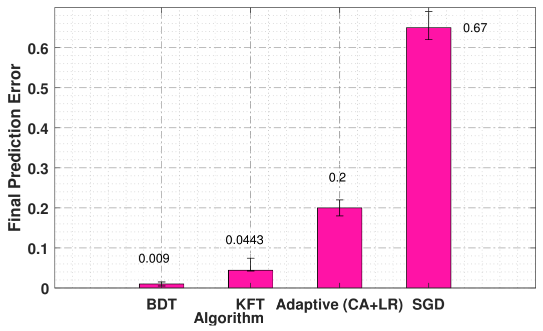


Fig. 10. Comparison of the four different algorithms in terms of performance degradation (smaller values mean better performance).

7 Application and Case Study:

We present a direct application of our predictive and analytic framework in a case study below. As shown in the Fig. 7, the response latency seems quite predictable and stable around the mean value for the first 200 minutes of the experiment. A sudden increase of more than 500 ms can be observed from the 200th minute as more workload is added to the application server. The empirical observation here is that the predicted spikes of the response latency are quite concerning for capacity planning and suggest a limiting factor of the application as we simulate virtual users of more than 1000. A plausible step may be to examine server resources, for instance CPU resources and add more capacity to the server. The increase in response latency means that we need to examine the resources provisioned for the application in order not to violate SLA metrics. Another immediate step to take with this observation may be to redirect or control admission of more users to the application server.

Fig. 9 shows the prediction of the server hit per second on the application server. With the predictions from the training and validation data set, we observe a maximum of 70 server hits per second as robot user open sessions and perform activities such as browsing or filling out forms on the web service platform. We want to do a capacity planning with the information presented by the plots of these two figures.

Let us consider the cases that our application was running on a 10 Mbps Ethernet, requests are transmitted via TCP/IP protocol (size is 180 bytes) and a normal GET request is about 256 bytes. We also assume that any page has about 200 Kilobytes of dynamic content. For a standard TCP/IP a packet has about 32 bytes header to route this packet. This leads to a total of

214292 bytes (209.3KB). If this amount of data is moved over a 10 Mbps Ethernet then we expect a total of $10,000,000$ $(bits\ per\ second)/214292$ $(bits\ per\ page)$ to obtain the number of pages per second. Our computation leads to about 47 pages per second for the 10Mbps Ethernet. From Fig. 9a and Fig. 9b we observed a maximum of 70 hits per second for both training and test data set. We can therefore conclude that an application designed to expect more hits per second like the Amazon site on a black Friday or an application for a world soccer game, where millions of hits per second are expected, then 70 hits per second as we have observed from our predictive framework may not be adequate to serve our users. As indicated in table I the root cause of the AWS S3 outage in 2008 was due to under-provisioning of resources [5]. We argue that a proactive monitoring and a proper capacity planning through a system like this can prevent this type of situation.

7.1 Threats to validity

There are two major threats to the validity of our research. Web servers run on heterogeneous network devices with different configurations. If the communication established between JMeter and the web server is done on a server that is running on a Fast Ethernet device of 100 Mbps, then one does not expect the same speed from a server that uses a Gigabyte (1000 Mbps) Ethernet device and this can constitute a threat to external validity (the question whether the results obtained can be generalized beyond the experimental context). To mitigate this challenge, the experiments were run for a reasonable period of time (each experiment was repeated 10 times) on JMeter until the data were showing consistent results before they could be used for training the models.

The second threat to validity is mainly due to whether there could be errors arising from the implementation of our approach. This type of validity threat is called internal validity. Both automated and manual JMeter test scripts were written and executed for the experiments and the results were also manually verified to ensure that the system was doing 'what it was supposed to do'.

The H_2 Kalman filter offers accurate computation as the estimates and measured outputs are recursively compared and corrected. This process of predicting and correcting improves the performance of Kalman filtering techniques and this makes the approach more reliable and generalizable than linear regression approaches.

8 Summary and recommendations for future work

We have described and constructed a real-world cloud test bed for proactive cloud resource monitoring, adaptation and information gathering on cloud virtual infrastructure network and application server KPIs. To validate our approach, we implemented a web service platform merchandising a selection of products for clients to browse and make purchases. We then employed JMeter to simulate client-server behavior using robot users. For monitoring purposes, we interfaced our application with Google Analytics and Azure Application Insights for live server KPI monitoring and sampling.

We successfully applied the H_2 optimal Kalman estimator in training and building models on the average response time and the percentage CPU consumption of the application server and the virtual infrastructure network. We obtained good performance results for both training and validation datasets with an average prediction accuracy of 95.57 %. A decay constant of -0.0085 per unit time shows that the model is robust and resilient in making k-step-ahead predictions.

For future work we aim to evaluate the effectiveness of the $H\infty$ filtering technique for characterizing the latent variables of the virtual infrastructure network bandwidth consumption, percentage CPU utilization and the average throughput of the application server.

References

[1] 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, PP. 359-415, 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] K. Costello and M. Rimol, "Gartner forecasts worldwide public cloud end-user spending to grow 23% in 2021: Cloud spending driven by emerging technologies becoming mainstream," *Gartner Press Release Point*, 21 April 2021.

[4] Peter Cohen, "How much of Amazon's $7.3 billion AWS profit will rivals win?" Available: https://www.forbes.com/sites/petercohan/2020/01/06/how-much-of-amazons-73-billion-aws-profit-will-rivals-win/#6ec61a9f5bcd, Accessed: 2020-07-30.

[5] L. Zheng, L. Mingfei, L. O'Brien, and H. Zhang, "The cloud's cloudy moment: A systematic survey of public cloud service outage," *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 2, 2013.

[6] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[7] P. Diniz and R. Sergio, "Adaptive filtering : algorithms and practical implementation," *The Kluwer international series in engineering and computer science; SECS 399l.* Kluwer Academic Publishers: Boston, Mass.; London; Upper Saddle River, N.J. 1997. isbn. 0792399129.

[8] Mohinder S. Grewal, "Kalman filtering: theory and practice using MATLAB," *3rd ed, International ed. Wiley:* Hoboken, N.J., 2008. isbn. 1-282-68702-6.

[9] S. J. Russell, "Artificial Intelligence: A Modern Approach," *Prentice Hall series in Artificial Intelligence*, 2nd ed, International ed., 1998. Prentice Hall: Upper Saddle River, N.J., isbn. 0130803022

[10] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting," in *IEEE 4th International Conference on Cloud Computing*, vol. 79-80, pp. 500–507, 2011.

[11] S.J. Julier and J.K. Uhlmann, "Unscented filtering and nonlinear estimation," in *Proceedings of the IEEE. JProc. 92 (3),* pp. 401–422, 2004.

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

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

[14] E. Kalyvianaki, T. Charalambous, and S. Hand, "Adaptive resource provisioning for virtualized servers using Kalman filters," *LACM Transactions on Autonomous and Adaptive Systems, Assoc. Computing Machinery*, vol. 9, no. 2, Article No. 10, pp. 1–35, 2014.

[15] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters," *Proceedings of the 6th International Conference on Autonomic Computing, ICAC'09*, ACM Transactions on Autonomous and Adaptive systems., 2009, pp. 117–126.

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

[17] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, “Model-driven optimal resource scaling in cloud,” *Software and Systems Modeling, Springer Berlin Heidelberg*, vol. 17, no.2, pp. 509–526, 2018.

[18] R. Hu, J. Jiang, G. Liu, and L. Wang, “Efficient resources provisioning based on load forecasting in cloud,” *The Scientific World Journal*, vol. 2014, no. 2, pp. 3212–3231. ScientificWorld Ltd., 2014.

[19] Z. Chen, Y. Zhu, Y. Di, S. Feng, and J. Geng, “A high-accuracy self-adaptive resource demands predicting method in IAAS cloud environment,” *Neural Network World, Czech Technical University in Prague, Faculty of Transportation Sciences*, vol. 25, no.5 pp. 519–539, 2015.

[20] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, “Prediction of cloud data center networks loads using stochastic and neural models,” *IEEE 2011 6th International Conference on System of Systems Engineering*, pp. 276–281, 2011.

[21] A. Eddahech, S. Chtourou, and M. Chtourou, “Hierarchical neural networks based prediction and control of dynamic reconfiguration for multilevel embedded systems,” *Journal of Systems Architecture*, vol. 59, no. 1, pp. 48–59. Elsevier B.V, 2013.

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

[23] T.W. Gyeera, A.J.H. Simons, and M. Stannett, “Regression Analysis of predictions and forecasts of cloud data centre KPIs using the boosted decision tree algorithm,” *IEEE TechRxiv. Preprint*, March 2021.

[24] 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.

[25] U. Forssell, “On H_2 and H_∞ optimal estimation,” *Linkoping University Electronic Press*, 1996.

[26] L. J. Crassidis and L.J. Junkins, “Optimal estimation of dynamic systems,” *Chapman and Hall (CRC) Applied Mathematics and Nonlinear Science Series*. Chapman and Hall/CRC 2011. isbn. 978-1-4398-3985-0.

[27] B. Plaza, “Google analytics for measuring website performance,” *Journal of Tourism Management*, Vol. 32, No. 3, pp. 477–481, 2011.

[28] B. Plaza, “Monitoring web traffic source effectiveness with Google analytics: An experiment with time series,” *In Aslib Proceedings, Emerald Group Publishing Limited* , Vol. 61, No. 5, pp. 474–482, September 2009.

[29] D. Stephens and B. Wren, ”Azure monitor application insights documentation,” *Azure, Microsoft Research Academic*, 2021.

[30] Apache JMeter - User’s Manual, *The Apache Software Foundation*, 2020.

[31] J. Davies, B. Fox, N. Hodgkinson, and P. Stork, “Azure reference architectures,” *Azure, Microsoft Research Academic*, 2021.

[32] M. Quelhas, A. Petraglia, and M. Petraglia, “Design of IIR filters using a pole-zero mapping approach,” *Digital Signal Processing*, vol.23, no. 4, pp. 1314–1321, 2013. issn. 1051-2004.

[33] L. Tan and J. Jean, “Digital signal processing (Second Edition),” *Academic Press, Science Direct*, Boston, 301–403, 2014. issn. 978-0-12-415893-1.

[34] D. Doran, “Filter design using Matlab.” Available: https://dadorran.wordpress.com/2013/10/18/filter-design-using-matlab/, Accessed: 2020-06-20.

[35] “Identification toolbox documentation.” *Mathworks Inc: MATLAB*, March 2021.

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. He is a member of the IEEE.

Dr Anthony J.H. Simons is a Senior Lecturer in the Department of Computer Science, University of Sheffield, where he has served as the Director of Teaching, the Director of Undergraduate Admissions and former Head of the Testing Research Group. He has wide ranging research interests in object-oriented systems, including type theory, design methods, model-based verification and testing and model-driven engineering. He is the creator of the JWalk and CatWalk unit testing tools for Java. Recent work has included verification of English engineering requirements for Rolls Royce and generation of cross-platform tests for cloud brokers. He is a reviewer for the journal Software Systems Modelling (SoSyM).

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 modeling techniques to generate new understandings of physical systems; and computational modeling 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.