Database Workload Capacity Planning using Time Series Analysis and Machine Learning

Antony S. Higginson
Oracle Advanced Customer Services
Manchester, United Kingdom
antony.higginson@oracle.com

Mihaela Dediu
Oracle Advanced Customer Services
Manchester, United Kingdom
mihaela.dediu@oracle.com

Octavian Arsene
Oracle Advanced Customer Services
Bucharest, Romania
octavian.arsene@oracle.com

Norman W. Paton
University of Manchester
Manchester, United Kingdom
norman.paton@manchester.ac.uk

Suzanne M. Embury
University of Manchester
Manchester, United Kingdom
suzanne.m.embury@manchester.ac.uk

ABSTRACT

When procuring or administering any I.T. system or a component of an I.T. system, it is crucial to understand the computational resources required to run the critical business functions that are governed by any Service Level Agreements. Predicting the resources needed for future consumption is like looking into the proverbial crystal ball. In this paper we look at the forecasting techniques in use today and evaluate if those techniques are applicable to the deeper layers of the technological stack such as clustered database instances, applications and groups of transactions that make up the database workload. The approach has been implemented to use supervised machine learning to identify traits such as reoccurring patterns, shocks and trends that the workloads exhibit and account for those traits in the forecast. An experimental evaluation shows that the approach we propose reduces the complexity of performing a forecast, and accurate predictions have been produced for complex workloads.

ACM Reference Format:
Antony S. Higginson, Mihaela Dediu, Octavian Arsene, Norman W. Paton, and Suzanne M. Embury. 2020. Database Workload Capacity Planning using Time Series Analysis and Machine Learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3386140

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3386140

1 INTRODUCTION

As enterprises strive to adopt cloud technologies, one of the key drivers is agility. However, organisations often underestimate the costs that can emerge with *pay-as-you-go* subscription based models. Prior to cloud, in organisations with on-premises architectures, the administration teams would provision environments. However, this was often seen as slow and cumbersome, which elongated development lifecycles to the detriment of the business. Cloud now addresses the need for speed and agility, while potentially opening the doors to unintended consequences in terms of cost. For example, there may be a lack of a regulatory framework within an organisation to oversee the acquisition of cloud resources. This lack of financial and regulatory discipline is not the sole cause of spiralling costs. For every environment provisioned, a proportion of *that* provisioned resource will probably never be used, therefore provisioning the correct shape (in terms of CPU, Memory and Storage) of cloud resource is paramount. When done correctly, Cloud Financial Management can deliver environments that scale in a way that shows promise for increasing business agility.

For Cloud Service Providers (CSPs), a burning issue becomes one of stability for short, medium and long term resource allocation. Having enough resources to satisfy consumer demand without compromising SLAs is paramount. The paradigm for creating and maintaining infrastructure has moved from the consumer (individual I.T. estates) to the CSP, where the I.T. estates become mega-estates with data centres dotted around the world that satisfy network delays. This shift in responsibility, although attractive to the consumer, comes at a revenue cost to the CSP's who, it could be argued, offset this cost back to the consumer. Therefore a

dance is created where capacity planning suddenly becomes a focal point.

Most current work on cloud modelling is at the infrastructure level, building on modelling of virtual machines (VMs). Forecasting at this level bundles a range of things together because a workload can mask what is really going on at the database level. For example, the sum of CPU being used will be shown, but only some of that will be assigned to the database. From a capacity planning perspective this means we are accounting for CPU that the database is not using, as it is being used at the VM level, and from a monitoring persepctive we cannot decipher how much CPU is assigned to the database and if the database CPU growth is specifically increasing. Provisioning for DBaaS presents short and long term challenges:

- *short term* - predicting when you will run out of resources: proactive monitoring.
- *long term* - capacity planning, whether that is on-premises, off-premises or non-cloud environments.

In this paper, we present techniques that are applicable to different prediction durations, but the experiments focus on short term capacity planning, with a view to minimizing over provisioning through timely allocation of the necessary resources.

There has been significant work on modelling and provisioning for VMs and DBs. We are particularly interested in time series analysis because recurring patterns and trends are key features in both short and long term decision making for DBaaS. Key features such as peaks, troughs and trends are all exhibited when workloads are analysed over time. In this paper we investigate the application of time series analysis techniques to represent database workloads. For example, OLTP and OLAP workloads tend to have different footprints and exhibit different behaviours. Specifically, we investigate the applicability of different time series modelling techniques with differing levels of seasonality and growth.

The contributions of this paper are as follows:

- An investigation into the application of time series modelling techniques such as ARIMA and HES to workflow predictions for OLTP and OLAP workloads.
- A proposal for the self-selection and self-configuration of models such as ARIMA and HES for use with a given workload.
- An empirical evaluation with controlled workloads at the database layers of the technological stack such as Database Instances.
- A demonstration of the applicability of the approach in real world workloads hosted by Oracle Advanced Customer Services.

2 RELATED WORK

In this section, we discuss related work of relevance to this paper under the headings of *workload modelling* and *prediction in clouds*.

Currently there is no standard for making predictions that try to answer *"how much resource do I need?/when will I require more resource?"* type questions. Some proposals suggest that predictions should be done at a particular "*layer*" of the technological stack such as a VM. Some propose that this question should be answered as part of the "*provisioning exercise*", when the size of the resource being requested prior to being provisioned is known. Others propose that predictions become important when monitoring or some form of interaction is required as part of a Quality of Service (QoS) agreement. Still others suggest that predictions are important to costings and design of priorities. Jennings *et al.* [16] performed an exhaustive study of 250+ papers on resource demand profiling and highlighted the different types of models and techniques that are used to profile such demand. They noted that the techniques used today model the relationship between the resources used by an application and that application's performance, thus highlighting gaps between the two.

2.1 Workload Modelling

Workload modelling provides a foundation for a variety of activities, including demand prediction. Kraft *et al.* [18] looked at particular storage metrics (such as IO) in the context of data centres, focusing on how workloads influence each other. This work was done in the context of consolidated environments but focused on the VM layers of the stack. Several techniques have been identified and borrowed from other fields. For example, Gmach *et al.* [10] used linear regression techniques to identify trends and use a bin packing algorithm to group VMs together. Lorido-Botran *et al.* [21] looked at how to scale workloads that require elasticising in cloud environments. They focussed on the implications at the IaaS level, but did use Time Series Analysis in the prediction algorithm. In autonomic computing, Jiang *et al* [17] used time series modelling techniques to dynamically adjust power, scheduled labour and auxiliaries at the IaaS level based on consumption, thus predicting power usage and adapting accordingly. They use an ensemble of techniques from Time Series Analysis such as Moving Average, Auto regression, neural networks, Support Vector Machines, etc., and apply this at hourly, daily and weekly granularities, with daily granularity giving favourable results.

Workload modelling can be used to inform short term adaptations or longer resource requirements. In terms of short term actions, several proposals have focused on QoS, using modelling techniques to identify problem workloads.

Carvalho *et al.* [6], Lorido-Botran and Tania [21] and Chaisiri *et al.* [7] all looked at approaches that identified VMs that *could be* a problem then taking action to adapt. Similarly, Sakr and Liu [22] looked at satisfying SLAs and applied rules to resources consumed by the workload. Gmach *et al.* [11] used modelling techniques at the provisioning stage as part of a capacity planning exercise with the aim of creating a just-in-time methodology by analysing the patterns and trends those VMs create and a weighting algorithm which moves the workload to a new set of VMs that are less contentious. Krompass *et al.* [19] and Schaffner *et al.* [23] also studied the impact of workloads causing outages by migrating them to a *less used* host (VM).

In terms of modelling at particular layers of the stack, as discussed previously, most work focusses on the VM. However, modelling at this layer (VMs) masks the true usage of a particular workload, as a natural smoothing of the data points occurs. Duggan *et al.* [8] took a novel approach of modelling the actual workload of a PaaS (Database) using Concurrent Query Performance Prediction (CQPP). They leveraged machine learning and linear regression techniques to predict the performance of a query. It is not clear if the regression techniques used were based on Time Series Analysis techniques.

Calheiros *et al* [5] used the ARIMA (Auto Regressive Integrated Moving Average) model to predict web requests at the VM layer, where a VM is a represented as a standard configuration of resources (CPU, memory and storage) and its expected performance is assumed. If the predicted load is likely to exhaust the expected performance, a new VM is created, thus solving the scaling problem. Tran *et al.* [27] focused on Time Series Analysis at an hourly level when analysing VM workloads. They used Seasonal ARIMA, but only at the VM level, and did not take into consideration seasons within seasons, which computational resources inevitably exhibit. Heuristics in existing work are generally based on greedy algorithms using simple rules. Carvalho *et al.* [6] proposed prediction based techniques for cloud environments by looking at admission control and assuming that capacity planning of resources has been done separately. They use a quota-based approach. The prediction is created based on heuristics and simple rules are applied to determine if a workload is behaving *predicted*, using techniques such as Exponential Smoothing (ETS) to exhaust the available resources.

In this paper we use time series analysis techniques to learn models of mainstream database workloads that include patterns such as trends and seasonality. We apply these techniques to different metrics that can be captured from databases. In the experiments, we evaluate the approach for short term predictions (of the order of days), but we also have some experience using them for longer term provisioning. In so doing, we provide the most comprehensive evaluation of time series based techniques for database workload modelling to date. By understanding the "*data*" and its structures well enough, it doesn't matter what layer of the technological stack or what part of the exercise; we should be able to make predictions based no time series data with complex characteristics.

2.2 Prediction in Clouds

Performing predictions of workloads in clouds is a relatively new idea, but essential from both sides of the relationship whether that is the CSP that provisions the resources or the user who consumes the resources. The argument here is one of elasticity and the *pay-as-you-go* nature of clouds. Jennings *et al.* [16] identify that VM placement is a problem but focus mainly on vector bin packing, such that the sum of the resources requested do not exceed the size of the bin in which they have been allocated. Jiang *et al.* [17] looked at finding a suitable balance between reducing the power cost and maintaining service levels by capacity planning VM workloads based on usage and then assigning a priority based on financial penalties to the provider.

There has been much work at the IaaS layer of clouds, namely VMs, but whether in terms of placement, provisioning, monitoring or where the database resides, work on the PaaS layer is scarce. Arguably the most resource consumptive layer of the stack is the database layer. Given that databases are multifaceted in the tasks they are asked to perform in feature rich DBMSs such as Oracle, IBM, Amazon and Microsoft, Calheiros *et al.* [5] did an in-depth analysis of the ARIMA technique, again focusing on the VM as a workload but for QoS and solving the problem of dynamic provisioning that CSPs currently experience. They predict the workload behaviour and feed this into a queuing model for calculating the VM provisioning. They run ARIMA models against Web VMs at an hourly granularity with data points taken over a month.

The goal of prediction in cloud environments can be to consolidate resources and this benefits both the consumer and the provider. There may be several reasons for this, such as cost, to meet SLA's or to save energy. For example, Dynamic Demand Response (D2R) is a project by the University of Southern California to utilise smart grid architectures specifically to address the supply and demand mismatch [24]. They utilise time series analysis, and specifically ARIMA, as an approach to predict power values based on time series data from a *kWh* metric. They apply this technique across their campus at the IaaS and PaaS layer. However, it is unclear if they apply it specifically to one or both. We make a case that the technique should be architecture independent such that

it should work for time series data regardless of architecture or metric.

There has been some work on prediction for cloud databases. For example, Sousa *et al.* [26] looked at the MySQL database cluster in an Amazon EC2 cloud and used ARIMA to predict the workload. However, the environment is a simple one unlike our environment: an Oracle Exadata database cluster running several TPC workloads. Our aim is to evaluate enterprise workloads that reflect our customers', and as our experiments will show we have evolved the time series technique to encompass ARIMA with Exogenous and Fourier terms to understand the complex structures within the time series data and make predictions.

3 PROBLEM DEFINITION

This section provides more detail on the problem to be solved. The problem can be described as follows.

> Given a time series m that provides monitoring information about a workload w, generate a prediction z for a period following on from that of w. The prediction should account for external influences on w.

The time series m is a trace or log of data in a time series format, as illustrated in Figures 2 and 3. The time series captures specific features of w, such as CPU, memory or logical IOs, but the techniques proposed are generic to any metric that has a time series format: $[x_1, ..., x_n]$. The time series is associated with the frequency of the monitoring, such as hourly, daily, weekly or monthly. The frequency of the prediction matches the frequency of m. The prediction z consists of the predicted values and associated error bars. The external influences on the prediction are events such as batch jobs and backups that routinely and sporadically occur in computational workloads, and which do not always follow a uniform pattern.

4 TIME SERIES ANALYSIS FOR DATABASE WORKLOADS

Time Series Analysis is a family of techniques that is widely used in applications that require forecasting, such as economics, sales, budgeting and market trading. Alysha *et al* [20] found that many time series exhibit complex seasonal patterns and applied a combination of Box-Cox transformation, Auto-Regressive Moving Average (ARMA) errors, Trend and Seasonal components (TBATS). Accounting for these complex data structures, Skorupa [1] suggests using a combination of varieties of the ARMA method to handle seasons within seasons to solve econometric problems—an interesting idea relative to our goal of making predictions from computational workloads that also show these traits. The main mode of operation for time series analysis is to understand the underlying forces that are reflected in the observed data, and to fit a model for forecasting. Both of these goals require that the pattern observed is identified. Once this takes place, we can then interpret and integrate the pattern with other data and then forecast if events will happen again. For example, does computational resource consumption over time have a trend? Is there is a pattern to these observations? Is it recurring?

Time Series are complex because each observation is dependant upon the previous observation that will likely be *influenced* by more than one observation. Applied to a computational problem where resources are assigned and workloads can be volatile, a workload can spike at particular times during a day and these spikes can repeat periodically or randomly. These influences are called *autocorrelation* — dependant relationships between successive observations. Time Series Analysis is broken down into two main areas with a view to uncovering patterns that can lead to the identification of a suitable model for predicting the future resource consumption of a workload:

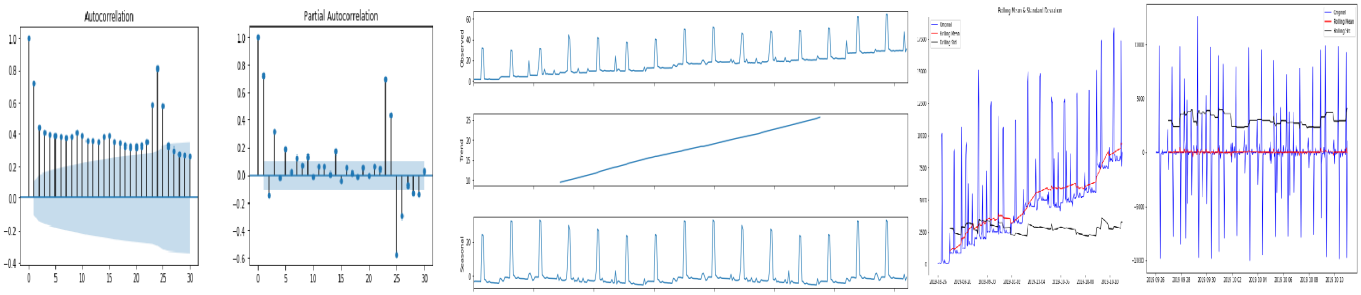
- *Time Domain* - ARIMA uses techniques such as Box-Jenkins and Dicky-Fuller to detect if the data is stationary, trending or requires an element of differencing.
- *Frequency Domain* - Techniques such as Fast Fourier Transform (FFT) to analyse data that is complex in a time domain.

This section reviews the main time series modelling and forecasting techniques, starting with techniques for stationary data and moving on to consider techniques for dealing with increasingly complex data. We also discuss how each approach can be used to capture features that are typical of database workloads.

4.1 Models

ARIMA is a class of models that capture the subtle structures of time series data. It was developed by Norbert Wiener *et al.* in the 1930's and 1940's [28]. Statisticians George Box and Gwilym Jenkins further developed these models for Business and Economic data in the 1970's, hence the name Box-Jenkins [2]. Computational resource utilisation is time series data and also exhibits structures such as trends, seasonality and/or complex multiple seasons, where a season within a season is exhibited. For example, variations may occur in the data at specific times, hourly, daily, weekly or monthly. These variations often repeat cyclically.

Such patterns are relevant to capacity planning for computational resources, where the goal is to answer the question *what resources are we likely to consume in the future?* We therefore investigate how to answer this question via ARIMA type models. Seasonal ARIMA (SARIMA) models also encapsulate seasonality, and are available in mathematical



(a) Correlograms (ACF/PACF) (b) SARIMAX Model Decomposition (c) Differencing Data (Trend)

Figure 1: Visualising Time Series Data

packages for R and Python. Here we cover multiple seasons, and have leveraged supervised Machine Learning to learn the past behaviours of a time series and forecast the future requirements for both long and short term predictions.

Time series models are typically characterized by their parameters. For ARIMA, the parameters are as follows:

- p is the number of autoregressive terms (AR) and can be found by using the Autocorrelation Function (ACF) or Partial Autocorrelation Function (PACF), as shown in Figure 1(a), and discussed in more detail in Section 4.3.
- d is the number of nonseasonal differences needed for stationarity, and
- q is the number of lagged forecast errors in the prediction equation.

The following additional parameters are required to handle the seasonal components:

- P is the order of the seasonal AR component (p),
- D is the seasonal differences needed for stationarity,
- Q is the seasonal order of the moving average terms (lagged forecast errors), and
- F is the frequency, for example 12 months, 24 hours.

Thus the SARIMA parameters are *(p,d,q,P,D,Q,F)*, which enables the model to handle both seasonal and non-seasonal workloads. We discover the seasonality of the data by decomposing it using library functions (in particular *statsmodels.tsa.seasonal* in python), as shown in Figure 1(b). When we decompose the data we visualise the traits as shown in Figure 1(a) and (b). For example, does the data have trend and seasonal patterns, and does it need to be differenced. If the data does have trend and seasonality then we can reduce the effects by differencing the data, as shown in Figure 1(c). This highlights the trend, and by differencing the data once we stablise it. Exogenous variables (Section 4.2) and Fourier Terms (Section 4.4) are leveraged to create more accurate forecasting models that can be executed on time series data at a server, database and transaction level.

From a simple regression model, as shown in formula (1), we see that $y(t) = \{y_t; t = 0, \pm1, \pm2, \ldots\}$ is a sequence of numbers (CPU, memory or IO), that is indexed by time t. We can see an observable signal sequence $x(t) = \{x_t\}$ and an unobservable noise sequence $\varepsilon(t) = \{\varepsilon_t\}$ that has independant variables that are random in nature.

$$y(t) = x(t)\beta + \varepsilon(t) \tag{1}$$

When the ARMA model is employed the (p,q) parameters are represented as:

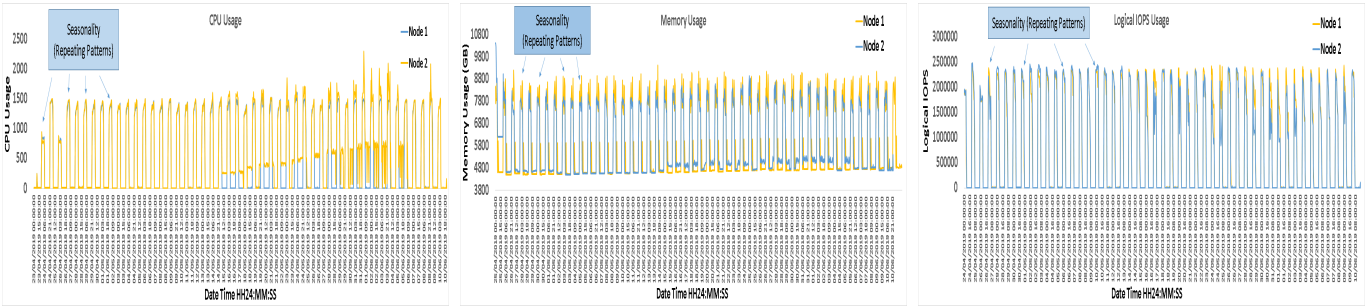
$$Y_t = \sum_{i=1}^{p} \phi_i Y_{t-i} + a_t - \sum_{j=1}^{q} \theta_j a_{t-j} \tag{2}$$

which is often reduced to:

$$\phi_p(B)Y_t = \theta_q(B)a_t \tag{3}$$

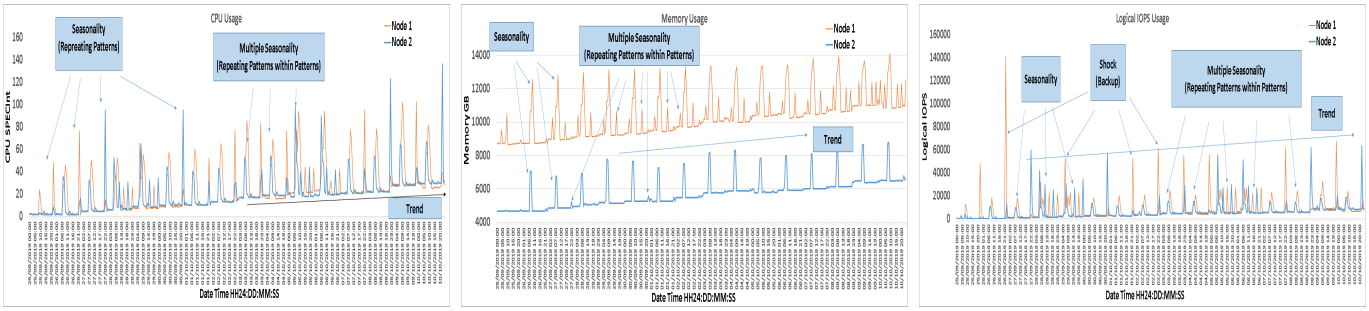
where $\phi_1, \ldots, \phi_p$ are the autoregressive parameters to be estimated, $\theta_1, \ldots, \theta_q$ are the moving average parameter to be estimated and $a_1, \ldots, a_t$ are a series of unknown random errors (or residuals) that are assumed to follow a normal distribution. This is commonly referred to as ARMA(p,q). B is the trend often found in time series data. However most computational time series data will probably introduce an element of stationariness. A stationary process is a stochastic process whose unconditional joint probability distribution does not change when shifted in time (i.e., in the future when performing a forecast). Consequently, parameters such as mean and variance also do not change in time.

When applied to our problem of capacity planning or short term monitoring, there is often an element of trend (incline or decline). Therefore, to make the data stationary we need to difference it, and for this we utilise Box-Jenkins to introduce parameter d:



(a) CPU Workload (b) Memory Workload (b) Logical IOPS Workload

Figure 2: Key Metrics: Workload Descriptions - Experiment One OLAP



(a) CPU Workload (b) Memory Workload (b) Logical IOPS Workload

Figure 3: Key Metrics: Workload Descriptions - Experiment Two OLTP

$$\phi_p(B)(1-B)^d Y_t = \theta_q(B)a_t \quad (4)$$

Thus (3) becomes (4) and creates the d that is the order of differencing. Replacing Y_t in the ARMA model with the differences creates the ARIMA(p,d,q) model. However, seasonality (patterns in the data) must be accounted for, to give:

$$\phi_p(B)\Phi(P)(B^s)(1-B)^d(1-B^s)^D Y_t = \theta_q(B)\Theta_Q(B^s)a_t \quad (5)$$

p, d and q are as defined earlier, and we introduce s as a known number of seasons (hourly, weekly, monthly, yearly). D is the order of differencing (this usually should not be greater than 2). P and Q are the autoregressive and moving averages when accounting for the seasonal shift. With analysis from the autocorrelation and partial autocorrelations (for which functions are available in python [4]) we can pre-populate the (p,d,q,P,D,Q,F) parameters that give a list of SARIMA models that are the most accurate for the data analysed.

4.2 Exogenous Variables

Exogenous variables are external parameters that convert the model ARIMA(p,d,q) to SARIMAX(p,d,q,P,D,Q,F) by including the linear effect that one or more external parameters has on the overall process; for example, a shock[1]. Computationally, examples could be a batch job, backup or fail-over that would seriously influence the computational resource consumption. Therefore Equation (2) extends to

$$Y_t = \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{k=1}^{r} \beta_k X_{tk} + \varepsilon_t + \sum_{j=1}^{q} \theta_j a_{t-j} \quad (6)$$

Here we see the exogenous variable being held in r with time varying predictors t and coefficients denoted by β. It is possible to have multiple exogenous variables applied to a model, thus covering behaviours that many systems exhibit today. For example, a system that has a backup, batch jobs and that periodically fails over with trends and reoccurring patterns of usage could be covered by the SARIMAX model, as long as the exogenous variables (shocks) are understood and accounted for.

4.3 Exponential Smoothing Models

ARIMA type models work by transforming a time series to a stationary series, studying the nature of the stationary series

[1]SARIMAX is a variant of the SARIMA method that includes exogenous variables.

through autocorrelation (ACF) and partial autocorrelation (PACF) and then accounting for the auto-regressive or moving average if it is present. The key point with ARIMA is that the past observations are weighted equally. *Exponential Smoothing* is the other side of the coin, to accommodate missing values in the data or fixed drift. Fixed drift is where the forecast of tomorrow's values is today's values plus a drift term. Some cases can be confusing because computationally the data may be cyclical in behaviour (peaks and troughs) yet exhibit no trend. This is because the cycles are not of a fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be. In exponential smoothing, recent observations are given more weight than older observations, so forecasting is based on weighted averages of past observations. The weights decay exponentially as the observations get older. The exponential smoothing methods were proposed in the late 1950s by Hyndman *et al.* and Brown [3, 15]:

- Simple exponential smoothing (SES), suitable for data with no clear trend or seasonal pattern,
- Holt's linear trend (HLT) [13], extended HES to allow data with trend, and
- Holt-Winters seasonal method [30], extended to include seasonality.

Complex seasonal patterns, such as multiple seasonal periods, high frequency seasonality, non-integer seasonality and dual-calendar effects cannot be included in the above models. A new framework, incorporating Box-Cox transformations, Fourier representations with time varying coefficients and ARMA error correction, was introduced for data exhibiting complex patterns forecasting as suggested by Skorupa [25]. The new model is named TBATS, which stands for **T**rigonometric seasonality **B**ox-Cox **A**RMA **T**rend **S**easonal components. TBATS improved BATS by modeling seasonal effects using a Fourier series based trigonometric representation. This change allows non-integer length seasonal effects to be captured.

model :

$$y_t^{(\lambda)} = l_{t-1} + \Phi \cdot b_{t-1} + \sum_{i=1}^{T} s_{t-m_i}^{(i)} + d_t \tag{7}$$

$$l_t = l_{t-1} + \Phi \cdot b_{t-1} + \alpha \cdot d_t \tag{8}$$

$$b_t = \Phi \cdot b_{t-1} + \beta \cdot d_t \tag{9}$$

$$d_t = \sum_{i=1}^{p} \varphi_i \cdot d_{t-i} + \sum_{i=1}^{q} \theta_i \cdot e_{t-i} + e_t \tag{10}$$

where,
T is the number of seasonalities
m_i is the length of the ith seasonal period
$y_t^{(\lambda)}$ is time series (Box-Cox transformed) at time t
$s_t^{(i)}$ is ith seasonal component
l_t is the level
b_t is the trend with damping effect
d_t is ARMA(p,q) process for residuals
e_t is Gaussian white noise
Φ - trend damping coefficient
α, β - smoothing coefficients
φ, θ are ARMA(p,q) coefficients

seasonal part :

$$s_t^{(i)} = \sum_{j=1}^{k_i} s_{j,t}^{(i)} \tag{11}$$

$$s_{j,t}^{(i)} = s_{j,t-1}^{(i)} \cdot cos(\lambda_i) + s_{j,t-1}^{*(i)} \cdot sin(\lambda_i) + \gamma_1^{(i)} \cdot d_t \tag{12}$$

$$s_{j,t}^{*(i)} = -s_{j,t-1}^{(i)} \cdot sin(\lambda_i) + s_{j,t-1}^{*(i)} \cdot cos(\lambda_i) + \gamma_2^{(i)} \cdot d_t \tag{13}$$

$$\lambda_i = \frac{2 \cdot \pi \cdot j}{m_i} \tag{14}$$

where,
k_i is the number of harmonics for the ith seasonal period
λ - Box-Cox transformation
$\gamma_1^{(i)}, \gamma_2^{(i)}$ - seasonal smoothing (two for each period)

The TBATS model has the following parameters: T, m_i, k_i, λ $\alpha, \beta, \phi, \varphi_i, \theta_i, \gamma_1^{(i)}, \gamma_2^{(i)}$. This leads to the question: how is the final model chosen? TBATS considers various alternatives and fits the models:

- with Box-Cox transformation and without it,
- with and without Trend,
- with and without Trend Damping,
- with and without ARMA(p,q) process used to model residuals,
- with a non-seasonal model, and
- with various amounts of harmonics used to model seasonal effects.

The final model will be chosen using the Akaike information criterion (AIC), which adds a penalty to the complexity of the model; the best model is the one having the lowest cost. The Python implementation was used in the experiments [25].

The reduced forms of TBATS are equivalent ARIMA models [20] generating the same forecast values. A TBATS model allows for dynamic seasonality whereas the ARIMA approach is that the seasonality is forced to be periodic as demonstrated by Hyndman [14].

4.4 Fourier Terms

SARIMAX deals with seasonality. However it has one major draw back, data with multiple seasonality. For example, data that exhibit trends and patterns that occur regularly such as every hour in a day or every day in a week or every week in a month. Such seasonal patterns are modeled through the introduction of Fourier terms, which are used as external regressors. In our analysis we look at computation data with hourly, daily, weekly or monthly seasons *(P1, P2, P3, Pm)*. Thus we have different Fourier series. Consider *Nt* as the ARIMA process:

$$yt = a + \sum_{i=1}^{M} \sum_{k=1}^{K_i} [\alpha sin(\frac{2\pi kt}{P_i}) + \beta cos(\frac{2\pi kt}{P_i})] + N_t \quad (15)$$

In the example shown in equation (5) we look at two seasons, *P1* running over a 24 hours period and *P2* running over a weekly period. Thus for each of the periods, P_i, the number of Fourier terms (k_i) are chosen to find the best SARIMAX parameters, and then ordered appropriately and selected based on which gives the best root mean squared error (RMSE). We determine the granularity of data required from the Makridakis Competition results [29]; for example, for an effective hourly forecast 700 hourly data points (circa 29 days) are required. In our solution we apply Fourier analysis if we detect time series data with multiple seasonality.

5 APPROACH

This section describes how the time series analysis techniques from Section 4 are applied to support capacity planning for DBaaS. This section also details how we propose to use machine learning to automate the forecasts, and algorithmically how we are able to discover the models, removing the need for the user to have an intrinsic understanding of the complexities of time series analysis, which is a field in its own right.

5.1 Overview

Our approach was to execute is to capture key metrics (CPU, IOPS and Memory) that are applicable to monitoring and capacity planning via an agent. The Agent specifically executes commands on the hosts that retrieve the metric values from the database and polls these metrics at regular intervals. The values from the metrics are then stored, centrally, in a repository where they are aggregated into hourly values. This cycle continues for a period of 30 days in the experiments. The Algorithms shown in Figure 4 are then executed. The first stage of the algorithm gathers the data and checks for any missing values. It is possible that the agent may have been at fault and may not have executed or polled the value from the database target; this can happen in live environments due to maintenance cycles or faults. If this is the case, a linear interpolation exercise is carried out to fill in the gaps based on known data points. We then begin the Machine Learning task and split the data into a training set and a testing set. Depending on whether the user chooses Holt-Winters Exponential Smoothing (HES) as discussed in section 4.3 or SARIMAX, a different branch of the algorithm will be followed. If SARIMAX is selected the algorithm then analyses the time series data for the metric being predicted and computes the ACF/PACF to determine which models are probably a good fit for the data selected. As the flow continues, the data characteristics are understood, such as stationarity, seasonality, multiple seasonality and shocks, where each model is then computed to obtain an RMSE. The model with the best RMSE is the most accurate. That model is then stored in a central repository and used for a period of one week or until the model's RMSE drops to a point where it is rendered useless. The same approach is executed for the HES algorithm in that it is kept for one week and the RMSE is continually monitored.

5.2 Machine Learning Algorithm

Time Series Analysis lends itself to a supervised learning approach for several reasons. We take historical data with known relationships between the observations captured over historical time points. The output is a numerical value (prediction) of what we think the future resource consumption of a metric is, therefore it is a regression problem. There is no need to continuously compute the ACF/PACF (Lags), add exogenous or find the Fourier terms. We simply re-train on the data unless the number of observations increases significantly or the time since the last use of the models lengthens beyond a certain period. We have determined that a model becomes stale over time and thus will need to be recomputed after a week, as shown in Figure 4. Depending on the type of time series technique used (HES or SARIMAX), the future prediction length also has a bearing on how the data is treated in the ML algorithm, as shown in Table 1, which is based on Makridakis [29].

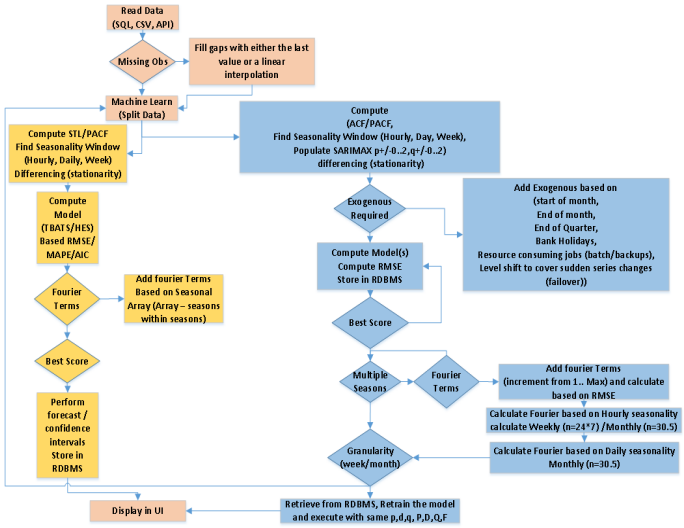


Figure 4: Algorithms: ESM/SARIMAX with FFT and Exogenous Workflow.

Table 1: Machine Learning Breakdown and Observations

Forecast	Obs	Train Set	Test Set	Prediction
SARIMAX Hourly	1008	984	24	24 (Hours)
SARIMAX Daily	90	83	7	7 (days)
SARIMAX Weekly	92	88	4	4 (Weeks)
HES [1] Hourly	1008	984	24	24 (Hours)
HES [1] Daily	90	83	7	7 (days)
HES [1] Weekly	92	88	4	4 (Weeks)

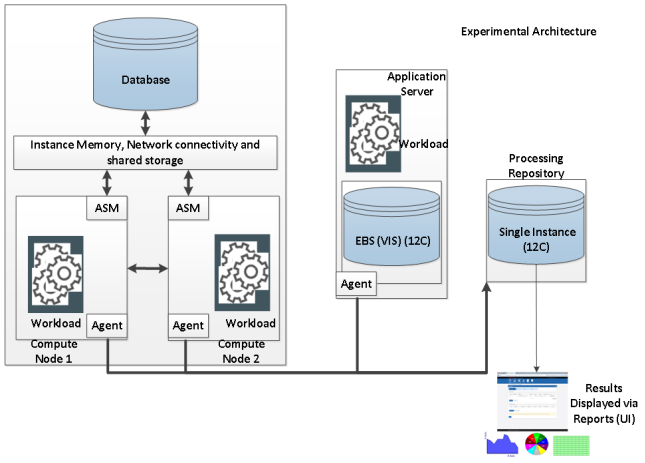


Figure 5: Experimental Architecture: Typical customer N-tier Architecture.

6 EXPERIMENTAL SETUP

6.1 Experimental Environment

The environment we created is based on an N-tier architecture, in that workloads are executed on an Oracle clustered database and through an application server that serves a web-based application; for example, transactions resulting from a sequence of *clicks* on a webpage. The load is shared between the nodes of the clustered database to keep an even balance of activity, as shown in Figure 5.

6.2 Experimental Workloads

Several Database workloads where used to run prediction models against as identified in our previous work [12]. To truly evaluate our full algorithm and the models, the workloads needed to exhibit several key characteristics, such as seasonality (repeating patterns). We produced two experiments running against a clustered database. Workload OLAP is a simple one that reflected seasonality, which can be depicted graphically via a chart as spikes in a workload that are surges in usage, for example users logging on at peak times. This is clearly identified for the key metrics in Figure 2. Trends (non-stationarity) that show the load growing—for example the data set becomes bigger and thus code execution times lengthen—or its resource utilisation increases and decreases again, are also visible in Figure 2. Workload two is a more complicated workload (OLTP), in that it exhibits a trend uniformly across all three metrics, as is shown in

[1]Table Footnote - Holt-Winters Exponential Smoothing

Figure 3. Users log on in surges at peak times to reflect typical usage of OLTP type systems, and this produced multiple seasonality. We also introduce shocks in the form of a backup which produces the large spike in logical IOPS, shown in Figure 3(c). The second experiment is to try and produce a prediction line that also grows in line with the trend, taking into consideration growth, multiple seasonality and shocks.

6.3 Experimental Models

Once the workloads had been executed, the relevant metrics captured and stored in a central repository, we ran the modelling routines. This consisted of exhaustively learning all the permutations of the models across the three techniques. As shown in the correlogram diagram examples in Figure 1(a), we measure the data over 30 lags, so each lag has a maximum of 22 models. For example, for Lag 1, the model combination is (1,0,0)(0,0,1,24) ,..., (1,1,2)(1,1,1,24), making the total number of models evaluated at over 6000 across the two experiments for one clustered database residing on two nodes.

The three techniques and the number of models are:

- ARIMA p,d,q = 180 models per instance (totalling 360 models)
- SARIMAX p,d,q,P,D,Q,F = 660 models per instance (totalling 1320 models)
- SARIMAX p,d,q,P,D,Q,F + Exogenous (4)+ Fourier Terms (2) = 666 models per instance (totalling 1332 models)

We measure the accuracy of every model against the RMSE and then choose the top model from each of the three methods. When we have the top model combination for each of the three methods, we then compare the accuracy of the results, as detailed in Table 2. There are several shocks in the form of backups that run every 6 hours (4 exogenous variables), and the FFT is made up of sine and cosine waves that are then added to the model with the best RMSE to see if it can be further improved in accuracy, again determined by the RMSE. This creates a huge number of models. In practice, we could reduce the number of models by tuning. We do this by looking at the correlogram shown in Figure 1(a), and looking at where the data points intersect with the shaded areas, as this gives an indication of a model that is likely to be suitable, thereby reducing the thousands of potential models considerably. For the purpose of empirical evaluation we compared over 6000 thousand models across two experiments, based on the RMSE.

7 EXPERIMENTS AND ANALYSIS

The experiments have been designed to investigate the extent to which the approach from Section 6 can address the following challenges in workoads:

C1 : Reoccurring patterns (seasonality).
C2 : Trends and Stationariness.
C3 : Multiple patterns (overlapping seasonality).
C4 : Shocks.

Our approach to this question is by way of empirical evaluation. Using increasingly complex deployments with representative workloads, we establish the extent to which we can predict the future load based on past load. This section describes the workloads and the platforms used in the experiments.

7.1 Experiment One - Simple OLAP Workload

Overview. Experiment one investigates Challenges *C1* and *C4*, it uses an OLAP workload with a modest number of 40 OLAP users connecting across the cluster. In this scenario, users connect to a clustered database and perform OLAP activities that are high in IO and execute for long periods of time; the tasks carried out are similar to those in TPC-H [9]. IO is generated via SQL activity and data manipulation language (DML) is executed via updates, inserts and deletes. This workload shows repeating patterns (seasonality), some growth (trend) and it did not exhibit multiple seasonality. A shock is introduced via a Backup that is run to perform housekeeping routines such as keeping the database logs (archivelogs) from filling up the disc drives in the operating system (OS). This is a routine that many database systems employ as some form of disaster recovery architecture, thus is a representative configuration. This produced a workload represented graphically in Figure 2. The dataset grew by several GB per hour.

Results. Examples of the predictions from Experiment one are shown in Figure 6, for CPU, for ARIMA, SARIMAX and SARIMAX with Exogenus Variables and Fourier Terms. The shaded area (blue) shows the data used by the algorithm for learning the behaviour of the workload. The recurring pattern (Challenge C1) is shown in each of the figures. The yellow section of the charts shows the prediction, which displays the continued pattern the algorithm has learnt. The results show that the peaks and troughs have been captured successfully by all three approaches to time series modelling. A similar behaviour is shown for the other metrics, namely Logical IO and Memory for each of the instances, and is not shown here for lack of space. Logical IO is a better metric for displaying complex data structures such as seasonality, and this is shown in Experiment 2, however, for the purpose of variety, forecasting a different metric such as CPU in Experiment 1 was chosen.

A deeper dive into the accuracy of the models is shown in Table 2(a). We tested the accuracy using three methods, which are Root Means Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Absolute Percentage

Accuracy (MAPA). The results show that SARIMAX with Exogenous variables and FFT is consistently more accurate and reduces the error across the three metrics of CPU, Logical IO and Memory, as shown in Table 2(a). The results also show that there is a significant jump in accuracy when the seasonal component of the data is taken into consideration when modelling Logical IOPS's. We only focus on the RMSE which shows a considerably lower number between the three models. With an error of 52879.49 logical IOPS (for instance, cdbm012 in Table 2a), the RMSE is based on values of of 2.3 million logical IOPS per hour throughput at the workloads peak. Thus the error rate is acceptable.

Given the nature of OLAP workloads working in an N-Tier architecture, the results of Experiment One were promising given the added complexity of the structures the data produced. ARIMA models do well in predicting the pattern, but it is expected to do well given that the workload exhibits simple structures and patterns. The SARIMA and SARIMAX models took into consideration any traits, seasonality and trend and improved the scores a little more. This is the reason why the RMSEs are lower in SARIMA than ARIMA. There was a backup task (cbdm011) that was executed from Node 1 at midnight every night, which also contributed to IO, CPU and Memory, thus creating an *exaggerated pattern*. Overall the SARIMAX with FFT and Exogenous variables was the most accurate model. This raises the question as to whether SARIMAX with FFT and Exogenous would give similar results on workloads that are much more complicated, such as OLTP?

7.2 Experiment Two - Complicated OLTP Workload - Seasonality and Growth

Overview. Experiment Two introduces another layer of complexity with the introduction of trends and multiple seasonality via an OLTP workload, thus presenting challenges *C1, C2, C3* and *C4* in a single scenario. In this experiment a workload was executed with users connecting to a clustered database, mirroring an OLTP type system, and we allow the user base to grow per day. The workload is similar to TPC-E [9]. IO is generated via SQL, including updates, inserts and deletes. Memory is consumed as the user connects and executes the SQL, as is CPU via the normal database internal memory and optimiser management systems. This activity runs for a period of 30 days, and metrics are captured every 15 mins via an agent and stored in a central repository. Aggregation then takes place over the hour between the four captured metrics. Multiple seasonality is introduced because the number of users is growing every day and there is a pattern of growth. Trend is introduced by increasing the user base by 50 users per day across the database cluster, thus growing the dataset and the consuming more resources. Surges in users are introduced twice daily at 07:00am of 1000 users for a period of 4 hours and again at 9am for another 1000 users for a period of 1 hour. These introduce several factors of multiple seasonality and a shock. A further shock is introduced by means of a Recovery Manager backup that is employed as a database housekeeping routine that prevents the database redo logs from filling up the disc drives on the OS. This is standard procedure for any RDBMS architecture that employs disaster recovery and thus is a relevant configuration. This produced the workload depicted graphically in Figure 3 a, b and c.

Results. Results of the prediction are shown graphically in Figure 7. The results show that the prediction line grows with the trend line and it captures the seasonality, including multiple seaonsonality. In these charts we show that SARIMAX with Exogenous variables and Fourier terms across three metircs of CPU, memory and Logical IOPS for one database instance provides a prediction line that reflects the past behaviours. In the Logical IOPS, Figure 3(c), the model takes into consideration the introduction of a shock (Backup). This is encouraging as the shock is a backup which significantly increases in IO, therefore the model is able to handle aspects that reflect the true nature of computational workloads and still be accurate. The repeating patterns shown in each of the figures are also reflected in the prediction lines. We have not added all the charts (CPU and Memory) due to there not being enough space in the paper.

A deeper dive into the accuracy of the models is provided in Table 2(b). We tested the accuracy using three methods, which are Root Means Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Absolute Percentage Accuracy (MAPA). The results show that SARIMAX with Exogenous variables and FFT is consistently more accurate, and reduces the error across the three metrics of CPU, Logical IO and Memory. The results also show that there is a significant jump in accuracy when the seasonal component of the data is taken into consideration when modelling Logical IOPS, but it also maintains the accuracy when we add in complex data structures such as multiple seasonality and shocks. Figure 7(c) clearly shows the multiple seasonality being predicted in the orange line as two large spikes (7:00am - 10am), reflecting a surge of 2000 users, and several smaller spikes (shocks) occurring at 00:00 in the form of a backup task. The trend (incline) is achieved by increasing the user base by 50 users per day. You can see the orange prediction line taking this complex workload structure into consideration in its prediction.

The results of Experiment Two were very promising, given the added complexity of the data introduced through multiple seasons, significant trend and shocks. Therefore, the models

(a) Experiment Results - OLAP

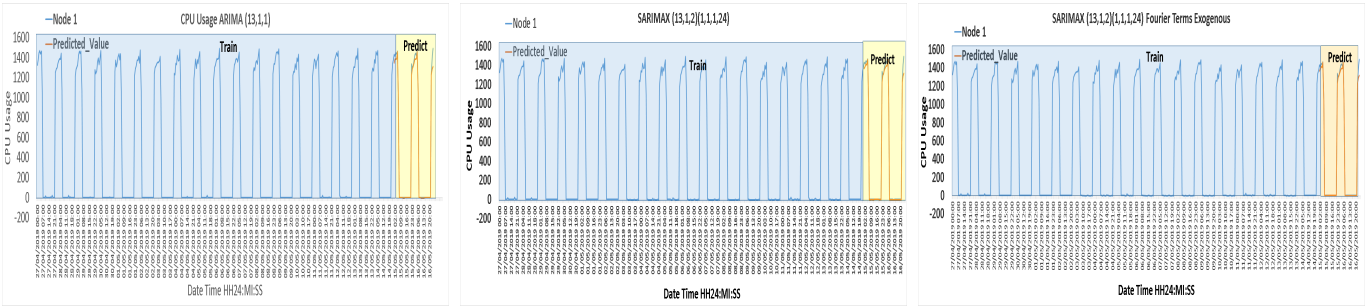
Forecast & Model	Metric	Root Mean Squared Error (RMSE)	MAPE %	Instance
ARIMA (13,1,1)	CPU	8.93	96.1	cdbm011
SARIMAX (13,1,2)(1,1,1,24)	CPU	8.4198	97	cdbm011
SARIMAX FFT Exogenous (13,1,2)(1,1,1,24)	CPU	8.4195	**97.06**	cdbm011
ARIMA (4,1,1)	CPU	44.78	97.28	cdbm012
SARIMAX (4,1,2)(1,1,1,24)	CPU	47.95	97.1	cdbm012
SARIMAX FFT Exogenous (4,1,2)(1,1,1,24)	CPU	38.89	**97.64**	cdbm012
ARIMA (13,1,2)	Memory	135.79	99.02	cdbm011
SARIMAX (1,1,1)	Memory	183.22	98.93	cdbm011
SARIMAX FFT Exogenous (1,1,2)(1,1,1,24)	Memory	130.42	**99.14**	cdbm011
ARIMA (13,1,2)	Memory	90.10	99.86	cdbm012
SARIMAX (1,1,1)(1,1,1,24)	Memory	61.30	98.91	cdbm012
SARIMAX FFT Exogenous (1,1,2)(1,1,1,24)	Memory	53.53	**99.92**	cdbm012
ARIMA (15,1,2)	Logical IOPS	39695	4533.01 -%	cdbm011
SARIMAX (4,1,1)(1,1,1,24)	Logical IOPS	15833.36	950.92 -%	cdbm011
SARIMAX FFT Exogenous (4,1,2)(1,1,1,24)	Logical IOPS	15112.06	**734.62 -%**	cdbm011
ARIMA (15,1,2)	Logical IOPS	151278.45	1375 -%	cdbm012
SARIMAX (4,1,1)(1,1,1,24)	Logical IOPS	52965.44	213.67 -%	cdbm012
SARIMAX FFT Exogenous (4,1,2)(1,1,1,24)	Logical IOPS	52879.49	**210.71 -%**	cdbm012

(b) Experiment Results - OLTP

Forecast & Model	Metric	Root Mean Squared Error (RMSE)	MAPE %	Instance
ARIMA (25,1,1)	CPU	10.89	88.8	cdbm011
SARIMAX (27,1,2)(1,1,1,24)	CPU	8.94	9.16	cdbm011
SARIMAX FFT Exogenous (27,1,2)(1,1,1,24)	CPU	6.02	**90.25**	cdbm011
ARIMA (22,1,1)	CPU	22.54	56.31	cdbm012
SARIMAX (2,1,1)(1,1,1,24)	CPU	12.22	86.08	cdbm012
SARIMAX FFT Exogenous (2,1,1)(1,1,1,24)	CPU	7.38	**86.54**	cdbm012
ARIMA (26,1,2)	Memory	551.07	97.08	cdbm011
SARIMAX (6,1,1)(1,1,1,24)	Memory	341.56	**98.02**	cdbm011
SARIMAX FFT Exogenous (6,1,1)(1,1,1,24)	Memory	359.44	97.66	cdbm011
ARIMA (21,1,1)	Memory	47.62	98.42	cdbm012
SARIMAX (27,1,2)(1,1,1,24)	Memory	0.12	**99.47**	cdbm012
SARIMAX FFT Exogenous (27,1,2)(1,1,1,24)	Memory	94.77	98.81	cdbm012
ARIMA (21,1,1)	Logical IOPS	8192.55	78.91	cdbm011
SARIMAX (14,1,2)(1,1,1,24)	Logical IOPS	6628.97	80.14	cdbm011
SARIMAX FFT Exogenous (14,1,2)(1,1,1,24)	Logical IOPS	4579.14	**82.1**	cdbm011
ARIMA (15,1,2)	Logical IOPS	8218.81	85.83	cdbm012
SARIMAX (1,1,1)(0,1,1,24)	Logical IOPS	1964.54	86.18	cdbm012
SARIMAX FFT Exogenous (4,1,2)(1,1,1,24)	Logical IOPS	2373.86	**87.33**	cdbm012

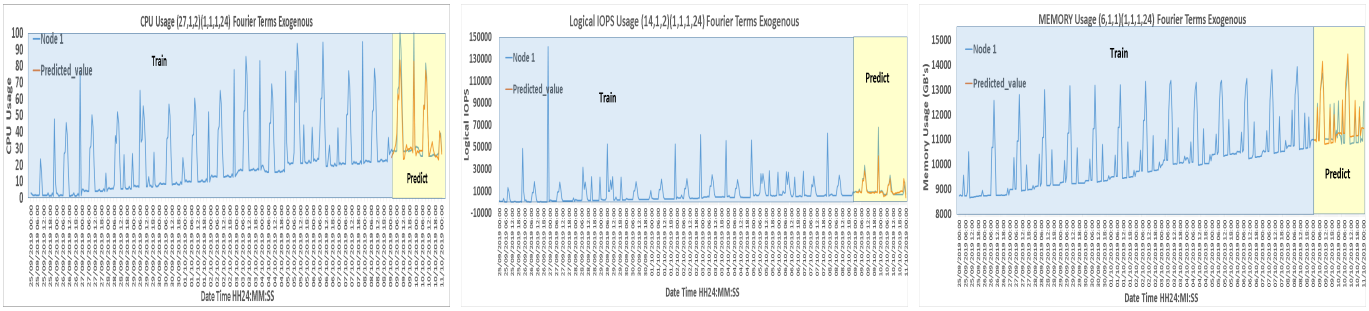
Table 2: Table of Results

were able to predict a trend and reflect all the traits of a complex computational model accurately.



(a) Exp 1 - ARIMA (b) Exp 1 - SARIMAX (c) Exp 1 - SARIMAX With Exog and Fourier Terms

Figure 6: Experiment 1: Prediction charts Comparing Three ARIMA Techniques



(a) Exp 2 - CPU (b) Exp 2 - Logical IOPS (c) Exp 2 - Memory

Figure 7: Experiment 2: Prediction Charts Using SARIMAX with Exogenous and Fourier Terms

8 DATABASE CAPACITY PLANNING IN PRACTICE

Oracle Advanced Customer Services has incorporated the work from this paper into its monitoring and assessment services that chart time series data. Figure 8 shows an early design of the UI and how the time series data is displayed across a clustered database instances. The user can select between SARIMAX or HES, as we have shown that these two models cover most nuances shown in computational workloads we studied.

The model has been found to be applicable in several use cases:

- *Short term monitoring:* within the next few days, what will resource usage look like across my technology estate?
- *Medium to Long Term Capacity Planning:* do I need to find extra capacity for my estate?
- *Migration:* If I need to migrate to a new platform, such as a Cloud architecture, what resource capacity do I need in the next 6 months to a year?

The approach is being applied across several thousand customers, covering 1000's of workloads involving different components in the technological stack. It has been applied to the following scenarios, as we can capture the metric usage via agents and store results in a central repository for time series analysis:

- Groups of *clicks* that make up a transaction in a web page.
- In conjunction with OATS, the Oracle Applications Testing Suite, we can predict if a transaction is beginning to slow down to aid pro-active monitoring of the application layer.
- Application containers such as weblogic can also be monitored as they are also a source of time series data.
- Network layers of storage, such as Network Attached Storage and SAN Volume Controllers, that are critical to the database instance are also monitored to display if the database is likely to suffer performance bottlenecks.
- Capacity Planning and mapping of different architectures to aid migrations to cloud technologies from on-premise.

We also intend to investigate the idea of applying this time series analysis approach with other learning techniques to act or make changes dynamically.

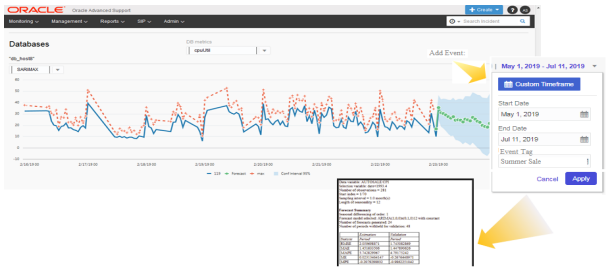
9 CONCLUSIONS

In conclusion, the use of machine learning has greatly increased the ability of these models to be utilised on computational workloads that exhibit diverse patterns over time. Manually creating a forecast is cumbersome, requires a degree of expertise in how the models work and understanding of the data to which the model is being applied. Forecasting is susceptible to mistakes, rendering the forecast inaccurate or wildly off when compared to actual findings because of these risks. Computational workloads are sometimes unsuitable for forecasting as the time series data is often unstable in that it has many variables (external and internal) that influence its behaviours; for example, system backups, batch jobs that aggregate data, or reports that consume vast amounts of resource infrequently or periodically. The data is also captured at different levels of granularity (polled every minute, hourly or daily) which also displays nuances or complex structures that often confuse forecasting models. When all of these challenges are aggregated, paired with the need for expertise and understanding from the person performing the forecast, manual forecasting of system workloads becomes impractical.

By automating the process and applying Machine Learning to continually assess the models performance, we reduce many of the costs and barriers. We reduce the risks of applying the wrong model as we continually assess the performance through Machine Learning to account for new behaviours the data (system) may adopt because the learning engine moves forward as the system moves forward, organically working together. We also don't *relearn* unless the model becomes unsuitable or the system (data) has changed significantly (shocks or new behaviours). Only at that point do we adjust to these new behaviours that all systems experience as new functionality is introduced throughout the systems life-cycle.

We also have performance tuned the algorithm by reducing the number of models we evaluate by automating the ACF and PACF functions. When we ran the experiments on a workload we evaluated several thousand models running on a two node database. If the clustered database resided on four nodes then the number of models needed to be evaluated across two experiments would be nearly 24000 and this is unmanageable, especially when faced with the challenge of providing analysis against thousands of workloads from thousands of customers routinely. Gains are also achieved by parallel processing the models. We only use HES or SARIMAX with Exogenous and Fourier terms.

Using established techniques such as SARIMAX with Exogenous variables to account for the nuances exhibited in computational workloads, we have improved the user experience of resource charts. Historically a chart would advise



(a) Single Instance With Exogenous Selection

Figure 8: Proposed User Interfaces: Model Selections and Predictions. ©Oracle Corporation

on past or present usage of a workload, in terms of metric based consumption such as CPU, IO, Memory or network bandwidth, and the user then had to interpret what that chart indicated about the future behaviour. By using this approach, we can now advise users on what we *think* may happen to their workloads. This gives a way to mitigate the responsibilities system administrators face when monitoring critical and important systems, rather than the "old" threshold-based monitoring approach, that often led to a reactive way of working when system outages took place. For example, consider a performance problem that begins weeks earlier but suddenly hits a threshold, becoming non-compliant relative to the SLA. The approach proposed in this paper could advise through a prediction that there is *likely* to be an issue soon, and therefore recommend refocussing some resources to avoid an outage. Providing this early warning capability to system administrators can only be a good thing.

Where this approach has its challenges is when a system is unstable or in a period of fault, for example frequent crashes as the Learning Engine then relearns to adopt new behaviours. In our algorithm we account for this by suggesting that the event needs to happen more then *3* times for it to be a behaviour, which can be changed manually. It is perfectly plausible that the system fails over to a new site to test disaster recovery. Therefore if a system crashes we discard it, however if the system continually crashes the learning engine will see it as a behaviour and account for it in its forecast. Live systems rarely continually crash but they do crash, therefore manual override is needed to accommodate systems that are *in-fault* as we suggest that forecasting will not be a true reflection of the system when stable.

There is still a need for threshold based monitoring. Utilising these techniques to predict when a threshold is likely to be breached is an advisable way to implement this approach for proactive monitoring and capacity based questions; we don't see our approach as a complete replacement for thresholds just yet.

REFERENCES

[1] 2019. *Forecasting Time Series with Multiple Seasonalities using TBATS in Python.* https://medium.com/intive-developers/forecasting-time-series-with-multiple-seasonalities-using-tbats-in-python-398a00ac0e8a

[2] George E. P. Box. 2008. *Time series analysis forecasting and control* (fourth edition. ed.). Hoboken, N.J.

[3] RG Brown. 1959. *Statistical forecasting for inventory control.* http://documents.irevues.inist.fr/handle/2042/28540.

[4] Jason Brownlee. 2014. Machine learning mastery. *URL: http://machinelearningmastery. com/discover-feature-engineering-howtoengineer-features-and-how-to-getgood-at-it* (2014).

[5] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. 2015. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications in QoS. *IEEE Transactions on Cloud Computing* 3, 4 (Oct 2015), 449–458. https://doi.org/10.1109/TCC.2014.2350475

[6] M. Carvalho, D. MenascÃĺ, and F. Brasileiro. 2015. Prediction-Based Admission Control for IaaS Clouds with Multiple Service Classes. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. 82–90. https://doi.org/10.1109/CloudCom.2015.16

[7] S. Chaisiri, B. Lee, and D. Niyato. 2010. Robust cloud resource provisioning for cloud computing environments. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 1–8. https://doi.org/10.1109/SOCA.2010.5707147

[8] Jennie Duggan, Olga Papaemmanouil, Ugur Cetintemel, and Eli Upfal. 2014. Contender: A Resource Modeling Approach for Concurrent Query Performance Prediction.. In *EDBT*. 109–120.

[9] Dominic Giles. 2019. *SwingBench 2.2 Reference and User Guide.*

[10] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. 2007. Capacity Management and Demand Prediction for Next Generation Data Centers. In *IEEE International Conference on Web Services (ICWS 2007)*. 43–50. https://doi.org/10.1109/ICWS.2007.62

[11] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. 2007. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *2007 IEEE 10th International Symposium on Workload Characterization*. 171–180. https://doi.org/10.1109/IISWC.2007.4362193

[12] Antony Higginson, Norman W. Paton, Suzanne M. Embury, and Clive Bostock. 2017. DBaaS Cloud Capacity Planning - Accounting for Dynamic RDBMS System that Employ Clustering and Standby Architectures. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*. 687–698. https://doi.org/10.5441/002/edbt.2017.89

[13] CE Holt. 1957. *Forecasting seasonals and trends by exponentially weighted averages.*

[14] Rob. J. Hyndman. 2014. *TBATS with regressors.* https://robjhyndman.com/hyndsight/tbats-with-regressors.

[15] Rob J. Hyndman, George Athanasopoulos, and OTexts.com. 2014 2014. *Forecasting : principles and practice / Rob J Hyndman and George Athanasopoulos* (print edition. ed.). 291 pages ; pages.

[16] Brendan Jennings and Rolf Stadler. 2015. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management* 23, 3 (01 Jul 2015), 567–619. https://doi.org/10.1007/s10922-014-9307-7

[17] Y. Jiang, C. Perng, T. Li, and R. Chang. 2012. Self-Adaptive Cloud Capacity Planning. In *2012 IEEE Ninth International Conference on Services Computing*. 73–80. https://doi.org/10.1109/SCC.2012.8

[18] Stephan Kraft, Giuliano Casale, Diwakar Krishnamurthy, Des Greer, and Peter Kilpatrick. 2013. Performance models of storage contention in cloud environments. *Software & Systems Modeling* 12, 4 (01 Oct 2013), 681–704. https://doi.org/10.1007/s10270-012-0227-2

[19] Stefan Krompass, Harumi Kuno, Umeshwar Dayal, and Alfons Kemper. 2007. Dynamic Workload Management for Very Large Data Warehouses: Juggling Feathers and Bowling Balls. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*. VLDB Endowment, 1105–1115. http://dl.acm.org/citation.cfm?id=1325851.1325976

[20] Alysha M. De Livera, Rob J. Hyndman, and Ralph D. Snyder. 2011. Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing. *J. Amer. Statist. Assoc.* 106, 496 (2011), 1513–1527.

[21] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* 12, 4 (01 Dec 2014), 559–592. https://doi.org/10.1007/s10723-014-9314-7

[22] S. Sakr and A. Liu. 2012. SLA-Based and Consumer-centric Dynamic Provisioning for Cloud Databases. In *2012 IEEE Fifth International Conference on Cloud Computing*. 360–367. https://doi.org/10.1109/CLOUD.2012.11

[23] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. 2011. Predicting in-memory database performance for automating cluster management tasks. In *2011 IEEE 27th International Conference on Data Engineering*. 1264–1275. https://doi.org/10.1109/ICDE.2011.5767936

[24] Yogesh Simmhan, Saima Aman, Alok Kumbhare, Rongyang Liu, Sam Stevens, Qunzhi Zhou, and Viktor Prasanna. 2013-07. Cloud-Based Software Platform for Big Data Analytics in Smart Grids. *Computing in Science Engineering* 15, 4 (2013-07), 38,47.

[25] Grzegorz Skorupa. 2019. *TBATS implementation in Python.* https://github.com/intive-DataScience/tbats.

[26] Flavio R. C. Sousa, Leonardo O. Moreira, JosÃĺ S. Costa Filho, and Javam C. Machado. 2018. Predictive elastic replication for multi-tenant databases in the cloud. *Concurrency and Computation: Practice and Experience* 30, 16 (2018), e4437. e4437 cpe.4437.

[27] V. G. Tran, V. Debusschere, and S. Bacha. 2012. Hourly server workload forecasting up to 168 hours ahead using Seasonal ARIMA model. In *2012 IEEE International Conference on Industrial Technology*. 1127–1131. https://doi.org/10.1109/ICIT.2012.6210091

[28] Norbert Wiener. 1950. *Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications.* http://hdl.handle.net/2027/uc1.b4062686

[29] Wikipedia contributors. 2019. Makridakis Competitions — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Makridakis_Competitions&oldid=903376442 [Online; accessed 1-August-2019].

[30] PR Winters. 1960. Forecasting sales by exponentially weighted moving averages. In *Management Science*, Management Science (Ed.).