Chapter 18
Online Workload Forecasting

Nikolas Herbst, Ayman Amin, Artur Andrzejak, Lars Grunske, Samuel Kounev, Ole J. Mengshoel and Priya Sundararajan

Abstract This chapter gives a summary of state-of-the-art approaches from different research fields that can be applied to continuously forecast future developments of time series data streams. More specifically, the input time series data contains continuously monitored metrics that quantify the amount of incoming workload-units to a self-aware system. It is the goal of this chapter to identify and present approaches for online workload forecasting that are required for a self-aware system to act proactively - in terms of problem prevention and optimization - inferred from likely changes in their usage. The research fields covered are machine learning and time series analysis. We describe explicit limitations and advantages for each forecasting method.

Nikolas Herbst
University of Würzburg, Germany, e-mail: nikolas.herbst@uni-wuerzburg.de

Ayman Amin
Swinburne University of Technology, Melbourne, Australia,
e-mail: aabdellah@swin.edu.au

Artur Andrzejak
Heidelberg University, Germany,
e-mail: artur.andrzejak@informatik.uni-heidelberg.de

Lars Grunske
Humboldt-Universität zu Berlin, Germany,
e-mail: lars.grunske@informatik.hu-berlin.de

Samuel Kounev
University of Würzburg, Germany, e-mail: samuel.kounev@uni-wuerzburg.de

Ole J. Mengshoel
Carnegie Mellon University, CA, USA, e-mail: ole.mengshoel@sv.cmu.edu

Priya Sundararajan
Carnegie Mellon University, CA, USA, e-mail: priya.sundararajan@west.cmu.edu

18.1 Introduction

The purpose of online workload forecasting, as discussed here, is to predict the future workload of a computing system. Forecasting the future workload is central in the context of self-aware systems as defined in Chapter 1. The quality of such forecasts will, along with the future state of the computing system itself, have a profound effect on the quality of the services provided to users by the computing system. Online workload forecasting is similar to other forecasting problems in related areas of technology, science, and business. These related problems can inspire and inform our forecasting objectives, which center on workloads in data centers, including private and public clouds.

Online workload forecasting plays a crucial role in monitoring and managing the resources in data centers, an environment where self-aware computing can provide significant value. Specifically, as the workload offered to a data center increases or decreases, data center resources can be scaled accordingly. An important consideration is to carefully balance resource consumption, on the one hand, and response time, on the other hand. With the increasing ubiquity of cloud computing, smart phones, wearable computing enabled by clouds, PaaS, SaaS, virtual machines, and related technologies, online workload forecasting clearly has a key role to play.

Different stakeholders clearly have different goals and concerns when it comes to data center resource management. Three main types of stakeholders, along with their goals and concerns, should be considered. *End users* focus on the availability, performance, and utility of their applications and their supporting cloud services. A *data center owner* is typically concerned about cost, associated with both buying equipment and running the data center (energy cost, for example, can be very high). One component of cost optimization is to carefully perform re-balancing of services in cloud data centers [60]. The *app owner* is in the middle and purchases resources from the data center owner for hosting one or more applications that provide services to the end users. An app owner tends to focus on supporting and pleasing a potentially broad range of users while not incurring too much cost with the data center owner. Workload forecasting can play a key role in helping to meet the goals of each of these three stakeholder groups.

In the presence of seasonal (or cyclic) patterns or long-lasting trends, forecasting results still need to be accurate and reliable. Intuitively, there are several complications making forecasting difficult, including: abrupt events, randomness, seasonal variation, and trend changes. Seasonal or cyclic behavior includes the diurnal cycle, which is a pattern that repeats every 24 hours around the world, impacting how computers, mobile devices, the Internet, and the Web are used. For example, the traffic or load of social networking Web sites often shows a clear diurnal pattern [13, 47]. In addition, there may be weekly cycles, due to the fact that people's behavior on weekdays is typically quite different from their behavior on weekends. Both daily and weekly seasonality patterns, as well as other types of cyclic load variations, can have a dramatic impact on data center load. Obvious examples of this include holidays and other events that occur on an annual or multi-annual cycle. In addition to

these temporal patterns, there are strong empirical indications that humans follow clear spatial and social patterns [13].

That being said, there are issues beyond cyclic behavior, namely trends, abrupt events, and randomness. As examples of trends, we can consider the steady increases in various prices, for example stock market prices or housing prices. Examples of abrupt events include, for example, crashes in stock markets, strong earthquakes, rain in arid climates, and reaction to headline news on the Web [44, 55]. Finally, there is probabilistic, random, statistical or stochastic behavior.

In the following Section 18.2, we discuss preprocessing actions that are relevant for forecast methods in general. In Section 18.3 and Section 18.4, we describe forecast methods from the machine learning and the time series analysis domain that are applicable in the context of workloads mentioning explicit advantages and limitations of each. Section 18.5 presents the state-of-the-art of applying forecast methods in the IT workload domain.

18.2 Process of Modeling and Data Preprocessing

Online workload data is typically represented in the form of a time series $(\mathbf{w}_0, \mathbf{w}_1, \ldots)$, i.e., a sequence of scalar or vector values $\mathbf{w}_i$ ordered by timestamps indicating the sampling times. The forecasting of future values in this time series is performed as an evaluation of a so-called *forecasting function* f (or *model*). Such function utilizes as its input raw and/or preprocessed data known at the instant of the forecast computation. Such data can include historical workload data, system-related measurements, external data or signals, as well as past forecast values.

For example, the load of a public web server can be expressed as its CPU utilization over the last minute, i.e., the portion of time the server was busy. A corresponding forecast function f might use as input: the load values of the last hour of this and other servers in a cluster (historical workload data), currently available RAM and number of users on this server (application-related measurements), time-of-day and day-of-week (external data), and error of the forecast values in the last hour.

More formally, a time series forecasting function f can be expressed as follows:

$$y_t = f(x_t, x_{t-1}, \ldots, x_{t-L}, y_{t-1}, y_{t-2}, \ldots, y_{t-L}), \tag{18.1}$$

where y_t is the current prediction, $y_{t-1}, \ldots, y_{t-L}$ are past predictions, and x_t and $x_{t-1}, \ldots, x_{t-L}$ denote the vectors of current and past observations, respectively [58]. The observations include raw and/or preprocessed workload data and other types of data listed above. The parameter L specifies how much historical information is used as model input. In general, using all available data (i.e., setting L close to t) is hardly feasible as this would greatly increase model dimensionality, leading to spurious and probably inaccurate results due to over-fitting.

The forecasting functions (or models) are typically either models based on machine learning (Section 18.3) or autoregressive models including hybrid models

(Section 18.4). As a prerequisite for forecasting we may need to prepare data as well as to train (fit) the model for a particular application scenario. We explain in detail the data preparation process in the remainder of this section. Model fitting depends strongly on the type of the predictive function and is discussed in Section 18.3 and Section 18.4.

18.2.1 Overview of Modeling Steps

Projects in time series forecasting typically consist of two phases: the *modeling phase* and the *deployment phase*. The purpose of the modeling phase is to understand the data at hand, to engineer features, and to train a predictive model, or a set of those. In the deployment phase, a production environment, these models are used for computing forecasts.

18.2.1.1 Modeling Phase

This is typically an exploratory process with multiple iterations. The key steps of modeling are the following:

- Extraction and transformation of raw data into relational format. Examples include parsing of log data with format conversion and import into spreadsheets, into an SQL database, or into a distributed file system like HDFS. In several cases, corrective measures are needed prior to import to handle mal-formatted input lines or values, or to eliminate illegal characters.
- Cleaning of relational data by removing outliers and handling missing values is the next step. In many cases more extensive preprocessing steps such as median filtering and normalization take place.
- Feature engineering is a more advanced form of data preprocessing. It can be understood as creation of derived time series, which expose and represent forecasting-relevant information in a form accessible to the modeling algorithms. An example is to extract *time* until host load exceeds, e.g., 50% for more than one minute, and use the corresponding time series as additional input for the modeling algorithm. Even sophisticated modeling approaches, such as support vector machines (see Section 18.3), could hardly derive this information from the raw data. Another popular and automatic approach to feature engineering is dimensionality reduction, which can create new attributes (e.g., in case of principal component analysis (PCA) [25]) and remove irrelevant ones in a single step. Feature engineering is typically an essential but complex component of the modeling phase, known to be decisive for the overall success. We discuss this in detail in Section 18.2.2.
- Computation of a prediction target is needed when forecasting some derived signal instead of "raw" data. For example, for regression, we might want to forecast a difference from the current value one hour in advance instead of ab-

solute host load. In case of classification, target values must be quantified to represent them by symbolic elements from a fixed set of values, e.g., value intervals [7].

- Preprocessed data is then split into training, validation, and testing sets. The first two data sets are used in the subsequent model fitting step, while test data is necessary for evaluating model accuracy.
- Model fitting on the training data is the actual core of the modeling phase. In machine learning, typically a collection of algorithms such as decision trees, SVMs, and others (Section 18.3) are investigated. For autoregressive methods (Section 18.4), model coefficients are fitted by, e.g., least-squares-methods [25]. This step can be computationally demanding but can be easily automated by leveraging available libraries such as WEKA [68], Matlab toolboxes, and Python libraries (e.g., Scipy learn).
- Trained models are validated using test data sets (i.e., "out-of-sample" data not used in the fitting step). For time series data, the cross-validation approach [68] frequently used for evaluation is not suitable and can lead to overoptimistic results. Instead, so-called walk-forward testing known from the financial industry can be used [38]. Multiple types of evaluation metrics are commonly used to estimate the forecasting error, for example, mean squared error or mean absolute error. In case of classification, the confusion matrix or F_1-score [25] is commonly used.

As noted above, obtaining an accurate model typically requires several iterations, where the steps from feature engineering until validation are repeated. A lot of manual effort is spent for programming extraction and optimization of features, while other steps can more easily be automated.

18.2.1.2 Deployment phase

In this phase, the trained prediction functions are used for performing predictions in a production setting. It is typically dominated by a software engineering effort during which an automated prediction pipeline is created. The pipeline is used to process data streams or data batches for scenarios where the availability of forecast result is not time constrained. Several aspects need to be considered here:

- Robust functions for efficient computation of derived features need to be implemented. Such functions must be able to handle corrupt data and missing values (if the model cannot handle them) to ensure that the data forecasting subsystem does not break down for non–standard inputs.
- Another essential issue - especially for long-term forecasting - is correction of a so-called concept drift. Concept drift refers to the modeled process changing after some time, and the prediction function becoming less accurate [51]. The corrective measure includes monitoring the accuracy of the forecasting function online, and retraining it if an error exceeds a critical threshold [6]. In more severe cases it may be necessary to repeat the feature engineering step, especially

if the currently used features do not capture the essential information anymore, or new raw data types became available.

18.2.2 Feature Engineering

Feature engineering is the (partially manual) process of considering the raw data and identifying suitable derived values that become components of the vectors $\mathbf{x}_i$ serving as input for the forecasting function in Equation (18.1).

In the case of time series, designing and implementing features is particularly challenging since data can contain useful sequence information, e.g., event patterns or periodical fluctuations. In most scenarios, it is recommended to include data (or derived values) from the "past" relative to the time of sampling, in addition to information sampled at a given time instant. Typically, manual feature engineering (possibly supported by automated methods such as feature selection) is needed to both specify the amount of historical data (value of L in Equation (18.1)) and to identify significant historical data values.

Below, we give examples of typical features in the context of time series. However, a multitude of other approaches to preprocessing time series data exist. As an interesting example, Symbolic Aggregate approXimation (SAX) [39] converts time series values into a sequence of symbols from a finite set (essentially, the values are quantized according to their distance to the series mean value). This allows using more advanced methods for feature generation like identification of frequent patterns in the temporal behavior.

18.2.2.1 Aggregated values

A simple yet effective approach is to compute aggregates for each of the components of the observation vector $\mathbf{x}_i$. Such aggregates can include simple or exponential moving averages, maximum or minimum values in a (sliding) time window, medians over such a time window, and combinations thereof [51]. To capture different time scales, multiple features with different time window sizes (or other parameters) can be used. Since the number of candidate features can be large, typically feature selection [68] is also used to identify most informative raw data and derived features.

18.2.2.2 Exposing periodic patterns

Features exposing periodic patterns in temporal data are particularly relevant when hosts are subjected to recurrent demand fluctuations, e.g., due to periodic patterns in user behavior. It may be essential to consider multiple time scales. For example, a load of a host in a Web server cluster is likely to depend on the load for the last

few minutes, the time of the day (business hours versus night hours), and day of the week.

Moreover, for an online retailer like Amazon, considering week of the calender year might also be of value (e.g., Cyber Monday or Black Friday sales peak). Features exposing periodical patterns over multiple time scales can be implemented in several ways. A straightforward approach is to subsample past data by using only vectors $\mathbf{x}_{t-k}, \mathbf{x}_{t-2k}, \mathbf{x}_{t-3k}, \ldots$, where k is the targeted cycle period. This can be done for different values of k, e.g., 24 hours, one week, one month, and so on.

A more sophisticated option is to use calendar methods [57]. Here one computes load averages over the same hour of a day or week, over the same day of a month, etc. In the former case one could obtain 24 feature "groups" (each group with all components of the observation vector $\mathbf{x}_i$), one per each hourly slot in a day, and in the latter case 7 feature groups, one per day of the week.

If periodic patterns in data exist, features can be used to represent them directly and thus greatly facilitate the learning of an accurate model. In contrast, using consecutive raw historical values over a prolonged historical period is unlikely to exploit periodical patterns in data, and only increases the risk of overfitting.

18.2.2.3 Downsampling with feature selection

A more automated method of feature generation for time series is to select vectors of past observations (and even components of these vectors). This can be done using standard feature selection approaches like forward search with a wrapper method [68]. For example, forward search has been used in conjunction with model-based evaluation in [58]. Given a set of already selected k feature groups derived from $\mathbf{x}_{t-i(1)}, \mathbf{x}_{t-i(2)}, \ldots, \mathbf{x}_{t-i(k)}$, a next feature group derived from $\mathbf{x}_{t-i(k+1)}$ is added only if a candidate feature set $\mathbf{x}_{t-i(1)}, \mathbf{x}_{t-i(2)}, \ldots, \mathbf{x}_{t-i(k)}, \mathbf{x}_{t-i(k+1)}$ gives rise to a more accurate model (fitted on the training data and evaluated on the validation data set). To identify an optimal value of index offset $i(k+1)$, all values between $i(k+1)$ and L can be evaluated, which might be computationally costly.

18.3 Predictive Models Based On Machine Learning

18.3.1 Bayesian Networks

A Bayesian network (BN) is a directed acyclic graph whose vertices are random variables and the directed edges denote the dependency relationship among the random variables. Bayesian networks are well suited for systems where we need to make predictions under uncertainty [49]. Formally, a BN is defined as $B = (G, P)$, where G is a directed acyclic graph and P is the set of probability distributions associated with $G's$ nodes. The graph is denoted by $G = (\mathbf{X}, \mathbf{E})$ where

$\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ is the node set and $\mathbf{E}$ is the edge set. If there is an edge from X_i to X_j, then $(X_i, X_j) \in \mathbf{E}$ and we denote X_i as the parent of X_j. $pa(X_j)$ denotes the set of all parents of X_j. The key property of BNs is the *conditional independence* of the variables from any of their non-descendants, given the value of their parent variables, i.e., given the value of $pa(X_j)$, X_j is conditionally independent of all its non-descendants. A BN factorizes a joint distribution $\Pr(\mathbf{X})$ as shown below:

$$P(\mathbf{X}) = \prod_{j=1}^{n} (P(X_j \mid pa(X_j)) \tag{18.2}$$

For example, a set of random variables can be used to predict network traffic. Let us consider a Bayesian network model with the following random variables: resources assigned (R), workload (W), cloud service level (C), user experience (U) and energy consumed (E). Using probability theory, we can compactly represent the relationship among these random variables: $P(W)$, $P(R)$, $P(C|W,R)$, $P(U|C)$ and $P(E|R)$.

Modeling phase: Let $\mathbf{E} \subset \mathbf{X}$ be the variables under observation (we call them evidence variables) and let $\mathbf{e}$ denote their observed values (evidence). For the purpose of cloud performance prediction, a BN can be learned from data. Two of the many dimensions to consider are: (i), whether the BN's graph structure is known or not and (ii) whether the data is complete or not. For learning, if the data is complete, maximum likelihood estimation can be used. For incomplete data, the expectation maximization algorithm (EM) [16] can be used. EM is found to be computationally expensive for difficult BNs. An age-layered approach (ALEM) [48], has been developed to speed up EM. ALEM has been extended to the MapReduce framework [56]. The ALEM strategy was found to achieve gains in parameter quality and runtime. After BN learning, the different types of queries as mentioned above can be answered using the inference algorithm.

Deployment Phase: For online load forecasting, a Bayesian network can help to solve different probabilistic queries. The process of solving the probabilistic queries is called inference. The inference algorithms assume that the nodes in $\mathbf{E}$ are clamped to values $\mathbf{e}$. Computation of *most probable explanation (MPE)* amounts to finding a most likely assignment (y) to all of the non-evidence variables $\mathbf{R} = \mathbf{X} - \mathbf{E}$, or MPE($\mathbf{R}$). Computation of marginals amounts to inferring the *most likely value (MLV)* over one query variable $\mathbf{Q} \in \mathbf{R}$. Computation of the *maximum a posteriori probability (MAP)* generalizes MPE computation and finds a most probable instantiation over some variables $\mathbf{Q} \subseteq \mathbf{R}$, MAP($\mathbf{Q}, \mathbf{e}$). Different BN inference algorithms [41], can be used to perform the above computations.

Example: Di et al., have used a Bayesian model to predict the host load in a cloud system [18]. Their model predicts the mean load over a long-term time interval, as well as the mean load in consecutive future time intervals. The experiments are based on a Google trace with over 10K hosts and millions of jobs. The results suggest that this Bayesian method improves the load prediction accuracy by 5.6-

50% compared to other state-of-the-art methods based on moving averages, auto-regression, and noise filters.

Limitations:

- There is no universally accepted method for constructing a BN from data.
- It takes some effort to design a BN by hand.
- For large and complex BNs, posterior probabilities can be expensive to compute.

Advantages:

- The graphical structure of BNs is very intuitive and easy to understand for people who are not familiar with the domain.
- BNs provide a theoretical framework for handling missing data. Missing data is marginalized out by summing or integrating over all the possibilities of the missing values.
- BNs can handle both discrete and continuous variables in the same model.
- BNs provide a theoretical framework for handling expert knowledge using prior probabilities.

18.3.2 Neural Networks

Neural networks (NNs), often referred to as artificial neural networks (ANNs), are inspired by the way the human brain learns and processes information. ANNs consist of a large number of neurons interconnected in layers. The first layer consists of input neurons where input data is fed into the neural network. The input neurons send the data to the next layer, which consists of hidden neurons. The activity of the hidden neurons is determined by the input data, the weights in the interconnections between the input neurons and the hidden neurons and the activation functions. The output layer consists of output neurons whose behavior is determined by the hidden neurons and weights in the interconnections between the hidden neuron and the output neuron. The complexity of learning in a neural network depends on the number of layers, the interconnection patterns between different layers, the learning process for updating the weights, and the activation function that converts input to output. A single-layer neural network or perceptron consists of a single layer of output neurons. The inputs are fed directly to the output neurons through a series of weights. A multi-layer perceptron (or a deep neural network) consists of many hidden layers of neurons, with each layer fully connected to the next one.

Modeling phase: In order to train an ANN in a supervised manner to perform a task, we must adjust the weights of each interconnection in such a way that the error between the desired output and the actual ANN output is minimized. Backpropagation is the most widely used algorithm for determining the error derivative of the weights ($\frac{dW}{dt} = 0$). ANN learning algorithms fall into two broad categories: heuristic

techniques (variable learning rate back propagation, resilient back propagation) and numerical optimization techniques (conjugate gradient [33], quasi-Newton [17]).

In a supervised learning setting, input data along with its desired output data (x,y) is fed to the network. The task of the network is to find a mapping function $f : X \rightarrow Y$. For workload forecasting, the network can be trained at fixed intervals depending on the nature of the load variation (i.e., batch learning, e.g. hourly, seasonally or annually). In an unsupervised setting, the task of the learning algorithm is to update a sequence of functions $f_1, f_2, \ldots, f_t$ in such a way that the prediction (x_{t+1}, y_{t+1}) depends on the previous function f_t and the current data (x_t, y_t). Large memory is needed if the current prediction depends on all previous functions $f_1, f_2, \ldots, f_t$ and data $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$. For online workload prediction, adaptive neural networks can be used, i.e., learning and operating at the same time.

Deployment Phase: After training, the learnt ANN model is used for prediction. The test data is given as inputs to the input neurons. The learned weights are used to predict values at the output neurons. For online workload forecasting, we first train the ANN for a limited time of historical workload data. The trained ANN is then used to forecast the future workload.

Example: An ANN based framework (PRACTISE), has been proposed for online prediction of compute loads such as CPU utilization, memory utilization, disk usage and network bandwidth [71]. PRACTISE uses an online updating module that monitors the prediction errors periodically. If the errors suddenly surge, then the neural network model is retrained. Experiments were done using traces from IBM data centers. PRACTISE is able to efficiently capture the peak loads in terms of their intensities and timing, in contrast to classic time series models.

Limitations:

- Neural networks are difficult to design. One must determine suitable number of nodes, hidden layers and activation functions.
- The output values do not come with a confidence measure.
- Neural networks are a “black box”. During training, there is no easy way to ensure how the domain specific information is being used.
- Training typically requires significant computational resources.
- Neural network models can be prone to over-fitting.

Advantages:

- Neural networks can implicitly detect non-linear patterns in the training data.
- The static, non-linear function used by neural networks provide a method to fit the parameters of a particular function to a given set of data.
- A wide variety of ANN architectures and activation functions can be used to fit a given set of data.

18.3.3 Decision Trees

A decision tree [53] is a tree structured directed graph, where the goal is to predict the leaf nodes based on several attributes. If the target leaf node is continuous, then the decision tree is called a regression tree. If the target leaf node takes categorical values, then the decision tree is called a classification tree.

Modeling phase: For example, consider the data vector $(x,y) = (x_1,x_2,x_3,\ldots,x_k,y)$. Here, the target is y and it depends on the values of the attributes $(x_1,x_2,x_3,\ldots,x_k)$. Each interior node in the decision tree corresponds to an attribute X_i with value x_i. The edges emanating from node X_i are labeled with possible values of the attribute. Each leaf represents value of the target given the values of the attribute as we traverse the path from the root to the leaf node. A decision tree can be constructed by recursively partitioning the training data into subsets based on the attributes. The first step is to choose the attribute that best splits the given training data. The measure of goodness of split (score) is based on the impurity of child nodes. The best attribute to split on is the one that produces smaller impurity and thus more skewed label distribution at the child nodes. Examples of goodness of split measures include information gain and Gini index.

Deployment Phase: After constructing the decision tree, it can be used for prediction. Consider a test data vector $(x,y) = (x_1,x_2,x_3,\ldots,x_k,y)$. Our goal is to predict the target y. The test data X is classified by passing it through the tree starting at the root node. The test at each internal node along the path is applied to the attributes of X_i to determine the next arc along which X should go down. The label at the leaf node at which X ends up is output as its classification.

During online learning, it is preferable to update the existing decision tree as new training instances arrive without needing to build a new decision tree. Utgoff proposed an incremental decision tree (ID5R) [65]. The ID5R algorithm updates the tree at each node based on the "best" split according to the new training instance. This leads to efficient re-structuring of the subtrees without the need to re-iterate through the past training examples.

Example: Decision trees are used for both long term (yearly) [23] and short term (every 15 minutes) [52] forecasting. Ding [23] uses decision trees for long term forecasting of energy consumption. The attributes consist of 14 key factors that contribute to energy consumption including primary, secondary, and tertiary GDP, residential energy consumption, industry output, financial revenue and economic index. Ding's improved decision tree method outperforms traditional forecasting methods such as linear regression and exponential curves.

Limitations:

- Over-fitting the training data can happen if the training data is small or noisy. Post-pruning is often performed to avoid over-fitting.
- The accuracy of the model depends on the high order interactions of the input variables.
- Finding an optimal decision tree is an NP-complete problem [35]. Many decision tree algorithms employ a heuristic based approach (such as information gain) to guide the search in the hypothesis space. Each split depends on the previous splits, so an error in a higher split is propagated down towards the leaves.

Advantages:

- The decision tree can clarify the relationship between the factors influencing the forecast and possible forecasting.
- The decision tree extracts the decision rules from the dataset, which are stored in a knowledge base and used for forecasting.
- A large variety of extensions to the basic algorithm have been developed. These include algorithms to handle missing data, real valued attributes, post pruning methods, and incremental learning.

18.3.4 Support Vector Machines

The Support Vector Machine (SVM) is a supervised algorithm where the goal is to predict the value of a target given the values of the attributes. It can be applied to both real-valued and categorical data that are linearly or non-linearly separable. SVM is a robust classification and regression technique. SVM works by finding the optimal separating hyperplane that maximizes the margin of the attributes. A hyperplane can be defined by an intercept term b and a normal vector $\mathbf{w}$ perpendicular to the hyperplane. A hyperplane can be written as the set of points $\mathbf{x}$ (also called support vectors) satisfying the equation $\mathbf{w}^T\mathbf{x} = -b$.

Modeling phase: Consider a dataset $D = \{(\mathbf{x}_i, y_i)\}$, where each $\mathbf{x}_i$ is a point and y_i is the class label. For example, in a 2-class classification problem, the class labels are $y_i = +1$ or $y_i = -1$. Our goal is to learn a mapping function: $y_i = f(\mathbf{x_i}, \{\mathbf{w}, b\})$. The linear SVM classifier can be written as $y_i = sign(\mathbf{w^T x_i} + b)$. A value of $+1$ indicates one class and -1 indicates the other class. The maximum margin is given by $\frac{2}{\sqrt{\mathbf{w^T w}}}$. We want to maximize this margin, which is equivalent to minimizing $\frac{\mathbf{w^T w}}{2}$, subject to the constraints $(\mathbf{w^T x_i} + b) >= 1$ if $y_i = 1$ and $(\mathbf{w^T x_i} + b) <= 1$ if $y_i = -1$. Now we have a quadratic optimization problem and we need to solve for w and b.

In real world, training data can be non-linear. For non-linear SVMs, we map data to a rich feature space and construct hyperplanes in that space. Formally, we pre-process the data: $\mathbf{x} \leftarrow \Phi(\mathbf{x})$. Then, we learn the classifier: $f(\mathbf{x}) = \mathbf{w}\Phi(\mathbf{x}) + b$. But the dimensionality of $\Phi(\mathbf{x})$ can be large, making it computationally difficult to

solve for $\mathbf{w}$ and b. So we use kernel functions $K < x_i, x_j >$. The kernel functions and parameters have to be carefully chosen as this would influence the performance of the SVM model. The resulting high dimensional space from the kernel functions can make the data linearly separable, even though it was not linearly separable in the original attribute space.

Deployment Phase: After training the model, it is used to predict the labels of the test data. During online prediction, as one receives new data $(\mathbf{x})$, the model uses the learnt weights to predict the label. Online training can happen either in batch for a set of test cases or per test case based on the results of the prediction. Several online SVM algorithms have been developed [9, 43].

Example: Support Vector Machines (SVM) have been applied for measurements of CPU utilization, memory utilization, disk space usage, communication latency and bandwidth [50]. SVN-based forecasts are found to be more accurate and outperform the existing methods based on metrics like mean absolute error and mean square error among others. Hu et al., [34] have explored different machine learning models for multi-step-ahead prediction of grid resources. Their experiments indicate that Epsilon-Support Vector Regression and Nu-Support Vector Regression achieve better performance than Back Propagation Neural Network, Radial Basis Function Neural Network, and Generalized Regression Neural Network.

Limitations:

- Difficult determination of the regularization parameters, kernel parameters and choice of kernel.
- Sensitive to skewed distributions.
- Computationally expensive training process.

Advantages:

- Both simple (few dimensions) and complex (many dimensions) classification models can be learned.
- Model is robust to small training datasets.
- SVM employs sophisticated mathematical methods to avoid over-fitting.

18.4 Predictive Models based on Time Series Analysis

18.4.1 ARIMA Models

Auto-regressive integrated moving average (ARIMA) models were originally proposed by Box and Jenkins [10], and they are commonly used to fit linearly dependent time series data and forecast their future values. ARIMA models have been successfully applied in many fields such as financial and economic forecasting [54].

In addition, some researchers have applied these models to forecast future values of QoS attributes [2, 3, 12, 28] or for estimating system reliability [4].

The time series $\{z_t\}$ is said to be generated by an auto-regressive integrated moving average (ARIMA) model of orders p, d, and q, denoted by ARIMA(p,d,q), if it satisfies:

$$\Phi_p(B)y_t = \Theta_q(B)\varepsilon_t \tag{18.3}$$

where y_t is a stationary time series (of the original non-stationary time series $\{z_t\}$) computed by using d differences as $y_t = (1-B)^d z_t$ and $B^d z_t = z_{t-d}$. In addition, $\{\varepsilon_t\}$ is a sequence of independent normal errors with zero mean and variance σ^2. The auto-regressive polynomial is $\Phi_p(B) = \left(1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p\right)$ with order p and $\Theta_q(B) = \left(1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q\right)$ is the moving average polynomial with order q. The auto-regressive and moving average coefficients are $\Phi = (\phi_1, \phi_2, \cdots, \phi_p)^T$ and $\Theta = (\theta_1, \theta_2, \cdots, \theta_q)^T$, respectively. The model (18.3) can be rewritten as

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t \tag{18.4}$$

where y_{t-i} for $i = 1, ..., p$ are the past stationary observations, ε_t is the current error, and ε_{t-i} for $i = 1, ..., q$ are the past errors.

To forecast one-step-ahead, we shift from t to $t+1$:

$$y_{t+1} = \sum_{i=1}^{p} \phi_i y_{t+1-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t+1-i} + \varepsilon_{t+1} \tag{18.5}$$

and similarly we can forecast multi-step-ahead values.

If the original time series $\{z_t\}$ is stationary, then there are no differences used and $\{z_t\}$ is said to be generated by an auto-regressive moving average (ARMA) model of orders p and q and denoted by ARMA(p,q). In addition, most of the existing nonlinear time series models are normal extensions for ARIMA models including SETARMA models [61], introduced in Subsection 18.4.3.

Limitations:

- ARIMA models are generally used as black-box models. Insights are limited to the values of p, q, and d.
- Model selection and identification of ARIMA parameters is time-consuming.

Advantages:

- ARIMA defines a family of models that can be (automatically) selected and parametrized to provide accurate forecasts.
- ARIMA is usable as a black-box model.
- ARIMA is fast when used for prediction once model and parameters are selected.

18.4.2 GARCH Models

The auto-regressive conditional heteroscedastic (ARCH) models were introduced by Engle [24] to model high volatility by describing the dynamic changes in time-varying variance as a deterministic function of past errors. These models have become widely accepted for financial time series with volatility clustering and turn out to be an important tool in the field of financial forecasting [45] but also have been applied to software systems [2].

Engle formally defined the ARCH model for a conditional variance σ_t^2 of the dependent variable y_t as:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \cdots + \alpha_m \varepsilon_{t-m}^2, \tag{18.6}$$

where $\varepsilon_t = y_t - \sum_{i=1}^{p} \phi_i y_{t-i} - \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$, and m and α_i for $i = 0, \ldots, m$ are the ARCH model order and coefficients, respectively.

A generalization of the ARCH model (GARCH) where additional dependencies are permitted on lags of the conditional variance was introduced by Bollerslev [8]. In the GARCH model the conditional variance is more general than in the ARCH model and can be written as follows:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{r} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=0}^{m} \beta_j \sigma_{t-j}^2 \tag{18.7}$$

with these constraints

$$\alpha_0 > 0,\ \alpha_i \geq 0,\ \beta_j \geq 0,\ \textit{and}\ \sum_{i=1}^{r} \alpha_i + \sum_{j=0}^{m} \beta_j < 1. \tag{18.8}$$

Limitations:

- Inherits all ARIMA limitations (Section 18.4.1).
- Model selection and construction are more expensive compared to ARIMA models.

Advantages:

- Inherits all ARIMA advantages (Section 18.4.1).
- Can be used on time-series with high volatility.

18.4.3 SETARMA Models

Self exciting threshold auto-regressive moving average (SETARMA) models were first introduced by Tong and Lim [61] and further studied by Tong [62], Lim [46], and Cook and Broemeling [14]. Tong [62] reports that threshold effects can arise in

many scientific fields and the SETARMA models are able to characterize this because of their attractive features such as limit cycles. Consequently, the SETARMA models have been found useful in many real life applications including economics, population biology, hydrology, and software [3, 63].

The SETARMA model is a generalization of the ARMA model in the nonlinear domain, and its main idea is to start with a linear ARMA model and then allow the parameters to vary according to the past values of the time series data. In order to explain the idea, let l disjoint intervals be defined as $R_j = [r_{j-1}, r_j)$ for $j = 1, 2, \ldots, l$, and let an integer d_p be known as the delay parameter. For the time series $\{y_t\}$, each interval R_j defines a "regime", in the sense that, the time series value y_t is said to follow the regime j if $y_{t-d_p} \in R_j$. Accordingly, a SETARMA model is defined as a piecewise linear structure that follows a linear ARMA model in each j alternative regime, for $j = 1, 2, \ldots, l$, and switches among the different regimes based on the threshold values determined by the past values of the time series data.

A SETARMA model of order $(l; p_1, p_2, \cdots, p_l; q_1, q_2, \cdots, q_l)$ or SETARMA $(l; p_1, p_2, \cdots, p_l; q_1, q_2, \cdots, q_l)$ can be written as follows:

$$y_t = \mu^{(j)} + \sum_{i=1}^{p_j} \phi_i^{(j)} y_{t-i} + \sum_{s=0}^{q_j} \theta_s^{(j)} \varepsilon_{t-s} + \varepsilon_t^{(j)} \quad if \quad r_{j-1} \le y_{t-d_p} < r_j \tag{18.9}$$

where $\phi_i^{(j)}$ and $\theta_s^{(j)}$ $(i = 1, 2, \cdots, p_j; s = 1, 2, \cdots, q_j; j = 1, 2, \cdots, l)$ are model parameters, and $\left\{\varepsilon_t^{(j)}\right\}$ $(j = 1, 2, \cdots, l)$ are a sequence of independent normal errors with mean zero and variance σ_j^2. The ordered constants $-\infty = r_0 < r_1 < \cdots < r_l = \infty$ are known as the thresholds.

When q_j equals zero (for $j = 1, 2, \cdots, l$), the SETARMA model is reduced to the self exciting threshold autoregressive model which is denoted by SETAR$(l; p_1, p_2, \cdots, p_l)$. Similarly, the self exciting threshold moving average model, SETMA$(l; q_1, q_2, \cdots, q_l)$, is a special case of SETARMA when p_j equals zero (for $j = 1, 2, \cdots, l$). If the autoregressive orders and moving average orders are the same for all regimes and equal to p and q, respectively, the SETARMA model (18.9) takes the form:

$$y_t = \mu^{(j)} + \sum_{i=1}^{p} \phi_i^{(j)} y_{t-i} + \sum_{s=0}^{q} \theta_s^{(j)} \varepsilon_{t-s} + \varepsilon_t^{(j)} \quad if \quad r_{j-1} \le y_{t-d_p} < r_j, \tag{18.10}$$

which is SETARMA(l, p, q) where p and q are repeated l times. Similarly to bilinear models and exponential auto-regressive models, it is easy to show that the ARMA(p, q) model is a special case of the SETARMA model (18.10) when l equals one, which implies that the SETARMA models are natural extensions of the ARMA models.

Limitations:

- Inherits all ARIMA limitations (Section 18.4.1).
- Model selection and construction are more expensive compared to ARIMA models.

Advantages:

- Inherits all ARIMA advantages (Section 18.4.1).
- Can be used on non-linear time-series.

18.4.4 Cubic Smoothing Splines

Cubic smoothing splines (CS) can be fitted to univariate time series data to obtain a linear forecast function that estimates a trend [37]. The smoothing parameters are estimated using a likelihood approach enabling the construction of confidence intervals. This method is an instance of the ARIMA($(p,d,q) = (0,2,2)$) model as described above with a restricted parameter set that does not impair the forecast accuracy. This method is suitable for trends extrapolation, but seasonal patterns are not captured. In steep parts of a time series, the method overestimates the trend - accordingly the method should only carefully be applied to time series data with bursts, high noise ratio, or strong seasonal patterns. It has been observed that the computation time rises for more data points without an observable improvement in forecast accuracy.

Limitations:

- In presence of bursts or dominating seasonal patterns, the method is likely to overestimate the impact of an extrapolated trend.
- Forecast accuracy may not increase with more historical data.

Advantages:

- Simple model that is easy to understand.
- Gives good trend estimates in many cases.
- Only needs a few historical values and has a low computational overhead.

18.4.5 Extended Exponential Smoothing

Extended exponential smoothing (ETS) is based on the state space approach and can explicitly model a trend, a season and a trend component in individual exponential smoothing equations that are combined in the final forecast [36]. The component combination can either be modeled as additive or multiplicative. In addition, damping the influence of one of these components is possible. The forecasting process

starts in the first step by selecting an optimized model instance, before the parameters of the individual exponential smoothing equations are estimated. Having the model and the parameters adapted to the time series data, point forecasts and confidence intervals are computed. This method is able to detect and capture sinus-like seasonal patterns that are contained at least three times in the time series data. In the presence of more complex seasonal patterns, the ETS often fails to model those and returns quicker with a worse forecast accuracy compared to other methods.

Limitations:

- Positive time series values are required.
- May fail to model complex and overlapping seasonal patterns.
- User needs to decide if a seasonal pattern and trend component should be modeled as additive or multiplicative.

Advantages:

- Established approach with various implementations available.
- Reasonable model complexity and computational overhead.

18.4.6 tBATS Innovation State Space Modeling Framework

The *tBATS* innovation state space modeling framework [15] extends the ETS state space model. The goal of *tBATS* is the better handling of more complex seasonal effects by making use of a trigonometric representation of seasonal components based on a Fourier transformations, by incorporation of Box-Cox transformations and use of ARMA error correction. *tBATS* relies on a method that reduces the computational burden of the maximum likelihood estimation. We observed examples of higher forecast accuracy compared to the ETS method while computation time stayed in the same ranges.

Limitations:

- Positive time series values required.

Advantages:

- Improved capability to handle complex seasonality compared to the classical ETS approach (Section 18.4.5).

18.4.7 Model Selection and Conclusion

Before constructing a forecasting model, the given time series data should be checked for fulfillment of the various model assumptions. For example, a time series

for an ARIMA model should satisfy the following assumptions: *serial dependency*, *normality*, and *stationarity*. Consequently, before using any model, statistical tests are needed to check for these assumptions. If they are not satisfied, a suitable transformation is required to align the data with the assumptions. Traditional statistical tests include: the runs test for serial dependency, the Kolmogorov-Smirnov (K-S) test for normality, and the KPSS test for stationarity [26]. For achieving normality and stationarity in the variance, Box-Cox transformations [10] can be used, and differences of non-stationary time series can be used to produce stationary series in the mean. To determine if GARCH model should be used, the time series has to be checked for volatility and for dynamic changes in time-varying variance as a deterministic function of past errors. The common statistical test for volatility is the Engle test [24]. The selection of a SETARMA model is recommended for a nonlinear time series, which can be tested with the Hansen test [31]. The Cubic Smoothing Splines are basically an ARIMA($(p,d,q) = (0,2,2)$) model, and thus, this model can be used whenever the ARIMA assumptions are satisfied. For the Extended Exponential Smoothing approach the same model assumptions as for ARIMA need to be satisfied. The tBATS approach includes a Box-Cox transformation and accordingly can cope with non-normality. Besides the required assumption checking, rules based on common statistical measures like seasonality, normality, linearity, kurtosis and skewness can also be derived for model selection or ranking. A detailed approach for rule derivation and a set of general rules for a subset of methods is given in [67]. In practice, different forecast approaches are applied concurrently and either the result that is more likely to be accurate or a weighted combination of the different results is used. This technique is also known as ensemble forecasting or boosting. An example can be found in [32].

18.5 Applications of Workload Forecasting

In this section, we discuss applications of workload forecasting. Further applications and a broader view on workload characterization can be found in a recent survey [11]. It is worth noting that methodology as well as applications strongly resembling workload forecasting can be found in the domain of managing electricity supply [42]. Short term forecast (hours to one day ahead) of demand on electric power are of particular interest to power suppliers [42].

18.5.1 Load Forecasting for Data Centers, Grids and Cloud Environments

Load forecasting is used in this domain in a variety of ways including system and performance scaling, optimization of resource utilization and its sharing, estimation of the runtime of tasks, and others. For a more recent overview see [59], a survey

focusing on job runtime prediction, but also discussing applications and methods related to workload prediction in computational grids.

The Network Weather Service (NWS) [69] is one of the first and most widely known systems for load forecasting in Grid environments. It uses an ensemble of simple prediction methods and selects the one that performed best in the recent time frame. It can forecast usage of CPU, network, and memory of a single server on a short time frame.

Peter A. Dinda has contributed a series of early works in this domain [20], [21]. He typically uses simple autoregressive models (e.g., AR) since in his evaluation they compare favorably, e.g., to BM and LAST models. He evaluates two applications of host load forecasting: estimation of running times of jobs [20] and real-time scheduling for interactive applications where new jobs are assigned to resources with minimum expected load [22]. The results of the evaluation show the clear benefits of using a prediction-based strategy. In [21], also a forecasting method based on wavelet analysis is introduced.

In [1], the authors use a combination of seasonal load variation modeling and Markov model-based meta-predictors for CPU and network load forecasting within a wide range of time horizons: from several minutes to more than a week. The motivating applications are grid scheduling and support for infrastructure maintenance.

The approach in [73] also targets the prediction of running times of grid tasks. The application scenario is CPU load prediction. The forecasting technique is a combination of polynomial fitting (essentially, a variant of autoregressive methods) and detection of similar patterns; the latter method helps to exploit periodicity in the signal.

In [70], forecasting the future load of a grid system is achieved by combining an autoregressive model (AR) enhanced by confidence interval estimations with two filtering techniques: Savitzky-Golay smoothing filter and Kalman filter. The potential application of this approach is forecasting task run-time and guiding scheduling strategies.

Workload analysis, performance modeling, and capacity planning in the context of data centers are considered in [27]. In [40], a model is proposed to capture groups of VMs that behave in frequent and repeatable patterns. The targeted application domains are capacity management and VM placement in a data center. Other works targeting such scenarios and environments include [30] and [29].

The authors of [19] propose an approach to predict host load in a Google cluster with 10,000+ hosts. The method includes an exponentially segmented pattern model and a Bayes method exploiting a feature engineering and selection process.

Another approach of workload management in the context of video streaming is presented in [72], where the characteristics of workload burstiness are captured. In this way, they can model workload spikes caused by demand for a few very popular video clips.

18.5.2 Forecasting for Web-based Systems

Optimizing the performance, resource usage, quality-of-service (QoS) and costs of complex multi-tier applications is another domain where host load prediction plays an essential role.

In [64], an analytical model of a multi-tier Internet application based on a network of queues was introduced, where the queues represent different tiers of the application. This model can accurately predict response times even in complex workload scenarios, including caching.

The issue of predicting future resource loads under constraints in Internet-based systems is addressed in [5]. The authors propose a two-step approach that first obtains a representative view of the load trend from measured raw data, and then applies a load prediction algorithm upon this data. The envisioned application scenarios are load balancing and load sharing, overload and admission control, and job dispatching and redirection.

In [66], the topic of allocating servers to each of the websites in a data center with shared resources is considered. The constraints here are maintaining the QoS levels in different classes, and optimizing server usage while maintaining cost-efficiency. The proposed solution is based on a web server load prediction schema based on a hierarchical framework with multiple time scales.

18.6 Conclusion and Open Challenges

The ability to forecast external behavior that impacts a self-aware system (e.g., amount of arriving work) becomes a crucial feature for a self-aware system if it should not only react to changes in its environment, but also prepare for likely changes to maintain or increase goal compliance. In this chapter, we presented an overview of workload forecasting methods based on machine learning and time series analysis. We discussed the common methodological process and reviewed related work in the area of workload forecasting. We see this as a first step towards enabling self-aware systems to become proactive.

Besides the general challenge to apply meaningful forecasting techniques in an autonomic manner within self-aware systems, several open challenges exist in this field. One open challenge is forecasting for highly dynamic applications. As an example of a highly dynamic application, consider a cloud-based startup company with a cloud-based smart phone app that all of a sudden becomes extremely popular. It is extremely hard to forecast the medium- to long-term growth and the corresponding computational load for such an app. In contrast, it is relatively easy to forecast the short-term diurnal variation in demand, although this pattern will vary depending on the nature of the app. For example, a location-based social networking web site such as Brightkite or Gowalla often has higher traffic around dinner and lunch [47], while the data center load of a travel-oriented app like Waze may peak slightly before and after typical meal times.

References

1. S. Akioka and Y. Muraoka. Extended forecast of CPU and network load on computational Grid. In *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004*, pages 765–772, April 2004.
2. Ayman Amin, Alan Colman, and Lars Grunske. An Approach to Forecasting QoS Attributes of Web Services Based on ARIMA and GARCH Models. In *proceedings of the 19th International Conference on Web Services*, pages 74–81. IEEE, 2012.
3. Ayman Amin, Lars Grunske, and Alan Colman. An automated approach to forecasting qos attributes based on linear and non-linear time series modeling. In Michael Goedicke, Tim Menzies, and Motoshi Saeki, editors, *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 130–139. ACM, 2012.
4. Ayman Amin, Lars Grunske, and Alan Colman. An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software*, 86(7):1923–1932, 2013.
5. Mauro Andreolini and Sara Casolari. Load Prediction Models in Web-based Systems. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, New York, NY, USA, 2006. ACM.
6. A. Andrzejak and J.B. Gomes. Parallel Concept Drift Detection with Online Map-Reduce. In *2012 IEEE 12th International Conference on Data Mining Workshops (ICDMW)*, pages 402–407, December 2012.
7. Artur Andrzejak and Luis Silva. Using Machine Learning for Non-Intrusive Modeling and Prediction of Software Aging. In *IEEE/IFIP Network Operations & Management Symposium (NOMS 2008)*, Salvador de Bahia, Brazil, April 2008.
8. T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.
9. Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *J. Mach. Learn. Res.*, 6:1579–1619, December 2005.
10. George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. HoldenDay, San Francisco, 1976.
11. Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. Workload characterization: A survey revisited. *ACM Comput. Surv.*, 48(3):48:1–48:43, February 2016.
12. Bice Cavallo, Massimiliano Di Penta, and Gerardo Canfora. An empirical comparison of methods to support QoS-aware service selection. In *proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems*, pages 64–70. ACM, 2010.
13. E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proc. of KDD-11*, pages 1082–1090, 2011.
14. Peyton Cook and Lyle D Broemeling. Analyzing threshold autoregressions with a Bayesian approach. *Advances in Econometrics*, 11:89–108, 1996.
15. Alysha M. De Livera, Rob J. Hyndman, and Ralph D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.
16. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal Of The Royal Statistical Society, Series B*, 39(1):1–38, 1977.
17. John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.
18. Sheng Di, Derrick Kondo, and Walfredo Cirne. Host load prediction in a google compute cloud with a bayesian model. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 21:1–21:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
19. Sheng Di, Derrick Kondo, and Walfredo Cirne. Google hostload prediction based on Bayesian model with optimized feature combination. *J. Parallel Distrib. Comput.*, 74(1):1820–1832, 2014.
20. P.A. Dinda. Online prediction of the running time of tasks. In *10th IEEE International Symposium on High Performance Distributed Computing, 2001. Proceedings*, pages 383–394, 2001.

21. P.A. Dinda. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 17(2):160–173, February 2006.
22. Peter A. Dinda. A Prediction-Based Real-Time Scheduling Advisor. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002), 15-19 April 2002, Fort Lauderdale, FL, USA, CD-ROM/Abstracts Proceedings*, 2002.
23. Qia Ding. Long-term load forecast using decision tree method. In *Power Systems Conference and Exposition, 2006. PSCE '06. 2006 IEEE PES*, pages 1541–1543, Oct 2006.
24. R.F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, pages 987–1007, 1982.
25. J Friedman, T Hastie, and R Tibshirani. *The elements of statistical learning*. 2001. 00571.
26. Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference*. CRC, 2003.
27. Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, IISWC '07, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.
28. Manish Godse, Umesh Bellur, and Rajendra Sonar. Automating QoS Based Service Selection. In *proceedings of the IEEE International Conference on Web Services*, pages 534–541. IEEE, 2010.
29. Zhenhuan Gong and Xiaohui Gu. PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing. In *2010 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 24–33, August 2010.
30. Zhenhuan Gong, Xiaohui Gu, and J. Wilkes. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *2010 International Conference on Network and Service Management (CNSM)*, pages 9–16, October 2010.
31. Bruce Hansen. Testing for linearity. *Journal of Economic Surveys*, 13(5):551–576, 1999.
32. Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. *Concurrency and Computation - Practice and Experience, John Wiley and Sons, Ltd.*, 26(12):2053–2078, 2014.
33. Magnus R. Hestenes and Eduard Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, December 1952.
34. L. Hu, X. L. Che, and S. Q. Zheng. Online system for grid resource monitoring and machine learning-based prediction. *IEEE Transactions on Parallel and Distributed Systems*, 23(1):134–145, Jan 2012.
35. Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15 – 17, 1976.
36. Rob Hyndman, Anne Khler, Keith Ord, and Ralph Snyder, editors. *Forecasting with Exponential Smoothing : The State Space Approach*. Springer Series in Statistics. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2008.
37. Rob J Hyndman, Maxwell Leslie King, Ivet Pitrun, and Baki Billah. Local linear forecasts using cubic smoothing splines. Monash Econometrics and Business Statistics Working Papers 10/02, Monash University, Department of Econometrics and Business Statistics, 2002.
38. Charles D. Kirkpatrick II and Julie Dahlquist. *Technical Analysis: The Complete Resource for Financial Market Technicians*. FT Press, November 2010.
39. Eamonn J. Keogh and Jessica Lin. Symbolic Aggregate approXimation (SAX) Homepage.
40. A. Khan, X. Yan, Shu Tao, and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium (NOMS)*, pages 1287–1294, April 2012.
41. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

42. Ali Lahouar and Jaleleddine Ben Hadj Slama. Random forests model for one day ahead load forecasting. In *Renewable Energy Congress (IREC), 2015 6th International*, pages 1–6, March 2015.
43. Pavel Laskov, Christian Gehl, Stefan Krüger, and Klaus-Robert Müller. Incremental support vector learning: Analysis, implementation and applications. *J. Mach. Learn. Res.*, 7:1909–1936, December 2006.
44. Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 497–506, New York, NY, USA, 2009. ACM.
45. WK Li and K Lam. Modelling asymmetry in stock returns by a threshold autoregressive conditional heteroscedastic model. *The Statistician*, pages 333–341, 1995.
46. KS Lim. On the stability of a threshold ar(1) without intercepts. *Journal of Time Series Analysis*, 13(2):119–132, 1992.
47. O. J. Mengshoel, R. Desai, A. Chen, and B. Tran. Will we connect again? machine learning for link prediction in mobile social networks. In *Proc. of Eleventh Workshop on Mining and Learning with Graphs*, Chicago, IL, August 2013.
48. Ole J Mengshoel, Avneesh Saluja, and Priya Sundararajan. Age-layered expectation maximization for parameter learning in bayesian networks. In *Proc. of the Fifteenth International Conference on Artificial Intelligence and Statistics*, 2012.
49. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
50. Hema Prem and N. R. Srinivasa Raghavan. A support vector machine based approach for forecasting of network weather services. *Journal of Grid Computing*, 4(1):89–114, 2006.
51. Dorian Pyle, Text Design, Morgan Kaufmann Publishers, Sixth Floor, and San Francisco. *Data Preparation for Data Mining*. 1999. 01347.
52. Jian qiang Li, Cheng lin Niu, Ji-Zhen Liu, and Jun jie Gu. The application of data mining in electric short-term load forecasting. In *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, volume 2, pages 519–522, Oct 2008.
53. J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
54. YC Raymond. An application of the arima model to real-estate prices in hong kong. *Journal of Property Finance*, 8(2):152–163, 1997.
55. E. Reed, A. Ishihara, and O. J. Mengshoel. Adaptive control of apache web server. In *Proc. of Feedback Computing '13*, San Jose, CA, June 2013.
56. Erik B Reed and Ole J Mengshoel. Scaling bayesian network parameter learning with expectation maximization using mapreduce. *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012.
57. Jerry Rolia, Xiaoyun Zhu, Martin Arlitt, and Artur Andrzejak. Statistical Service Assurances for Applications in Utility Grid Environments. *Performance Evaluation Journal*, 58(2+3):319–339, November 2004.
58. D. Ruta and B. Gabrys. Neural Network Ensembles for Time Series Prediction. In *International Joint Conference on Neural Networks, 2007. IJCNN 2007*, pages 1204–1209, August 2007.
59. S. Seneviratne and S. Witharana. A survey on methodologies for runtime prediction on grid environments. In *2014 7th International Conference on Information and Automation for Sustainability (ICIAfS)*, pages 1–6, December 2014.
60. P. K. Sundararajan, E. Feller, J. Forgeat, and O. J. Mengshoel. A constrained genetic algorithm for rebalancing of services in cloud data centers. In *8th IEEE International Conference on Cloud Computing, CLOUD*, pages 653–660, 2015.
61. H. Tong and K.S. Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 245–292, 1980.
62. Howell Tong. *Threshold models in non-linear time series analysis*, volume 21. Springer, 1983.
63. Howell Tong. *Non-linear time series: a dynamical system approach*. Oxford University Press, 1990.

64. Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An Analytical Model for Multi-tier Internet Services and Its Applications. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 291–302, New York, NY, USA, 2005. ACM.
65. Paul E. Utgoff. Incremental induction of decision trees. *Mach. Learn.*, 4(2):161–186, November 1989.
66. T. Vercauteren, P. Aggarwal, Xiaodong Wang, and Ta-Hsin Li. Hierarchical Forecasting of Web Server Workload Using Sequential Monte Carlo Training. *IEEE Transactions on Signal Processing*, 55(4):1286–1297, April 2007.
67. Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomput.*, 72(10-12):2581–2594, June 2009.
68. Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005. 23937.
69. Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Gener. Comput. Syst.*, 15(5-6):757–768, October 1999.
70. Yongwei Wu, Yulai Yuan, Guangwen Yang, and Weimin Zheng. Load Prediction Using Hybrid Model for Computational Grid. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 235–242, Washington, DC, USA, 2007. IEEE Computer Society.
71. J. Xue, F. Yan, R. Birke, L. Y. Chen, T. Scherer, and E. Smirni. Practise: Robust prediction of data center time series. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 126–134, Nov 2015.
72. Hui Zhang, Guofei Jiang, K. Yoshihira, and Haifeng Chen. Proactive Workload Management in Hybrid Cloud Computing. *IEEE Transactions on Network and Service Management*, 11(1):90–100, March 2014.
73. Yuanyuan Zhang, Wei Sun, and Yasushi Inoguchi. Predicting Running Time of Grid Tasks Based on CPU Load Predictions. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 286–292, Washington, DC, USA, 2006. IEEE Computer Society.