A Survey of Prediction and Classification Techniques in Multicore Processor Systems

Cristinel Ababei, *Senior Member, IEEE* and Milad Ghorbani Moghaddam, *Student Member, IEEE*

Abstract—In multicore processor systems, being able to accurately predict the future provides new optimization opportunities, which otherwise could not be exploited. For example, an oracle able to predict a certain application's behavior running on a smart phone could direct the power manager to switch to appropriate dynamic voltage and frequency scaling modes that would guarantee minimum levels of desired performance while saving energy consumption and thereby prolonging battery life. Using predictions enables systems to become proactive rather than continue to operate in a reactive manner. This prediction-based proactive approach has become increasingly popular in the design and optimization of integrated circuits and of multicore processor systems. Prediction transforms from simple forecasting to sophisticated machine learning based prediction and classification that learns from existing data, employs data mining, and predicts future behavior. This can be exploited by novel optimization techniques that can span across all layers of the computing stack. In this survey paper, we present a discussion of the most popular techniques on prediction and classification in the general context of computing systems with emphasis on multicore processors. The paper is far from comprehensive, but, it will help the reader interested in employing prediction in optimization of multicore processor systems.

Index Terms—Multicore processor system , prediction, classification, exponential averaging, history predictor, autoregressive moving average (ARMA), Kalman filter, linear regression (LR), linear discriminant analysis (LDA), multinomial logistic regression, K-nearest neighbor (KNN), Bayes classifier, support vector machines (SVM), reinforcement learning (RL), online machine learning, neural network (NN), deep neural network (DNN), model predictive control

1 Introduction

Generally speaking, in modeling and in solving engineering problems we always do prediction in various ways. Models themselves (e.g., often embodied into simulation tools such as Wattch [1], McPAT [2], and HotSpot [3]) abstract away details of the physical system that is modeled and provide means to capture present and future behavior. Often times, solving problems implies making estimations about different figures of merit or attributes of the modeled system, such as performance or power consumption in the current state as well as in future states. Typically, optimization decisions then are made based on such estimations or predictions. When these decisions are made based on the current state estimations, the optimization approach is called *reactive*, because the system is designed to react to certain changes in system's behavior. On the other hand, when optimization decisions are made based on predicted future values of the attributes of interest, the approach becomes *proactive*, because measures are taken early on based on forecast values, thereby potentially achieving better optimizations. It is the proactive type of optimization approaches that we are particularly interested in the discussion presented in this paper, because such approaches usually employ explicit forms of *prediction techniques*—as the primary focus of this paper, in the general context of computing systems with emphasis on multicore processors—and because the reactive type encompasses the vast majority of all other works, which is too large to be discussed in a paper such this. We use the general term *computing system* to refer to processors themselves, systems that are built using processors (e.g., servers and smart phones or other mobile devices) but also systems of such systems (e.g., datacenters). Thus, we include in our discussion bus based and network-on-chip (NoC) based multicore processors (or chip multiprocessors, CMPs), laptops, servers, smart phones, and datacenters (DCs) or warehouse scale computers (WSCs). In addition, our objective is to emphasize an important trend: that of using predictions based on increasingly large data sets and where we believe the research community is headed to with such prediction based methods.

We build our discussion by considering the following: 1) the complexity of the technique as basic or advanced (i.e., machine learning based), 2) the particular component of the system to which the prediction technique is applied, such as bus, NoC, cores, and datacenters, 3) the particular figure of merit or design attribute that is the subject of prediction, and 4) the abstraction layer or layers, if cross-layer, where the technique of interest is implemented. The well known computing stack model generalized to also include the datacenter as the top most abstraction layer is shown in Fig. 1. The presented techniques are described with just enough details and diagrams to make the reading coherent and easy to follow without the need to interrupt and read from

• *The authors are with the Department of Electrical and Computer Engineering, Marquette University, Milwaukee, WI 53233. E-mail: {cristinel.ababei, milad.ghorbanimoghaddam}@marquette.edu.*

Manuscript received 13 Sept. 2017; revised 1 Oct. 2018; accepted 25 Oct. 2018. Date of publication 30 Oct. 2018; date of current version 10 Apr. 2019. (Corresponding author: Cristinel Ababei.)
Recommended for acceptance by R. Cumplido.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2878699

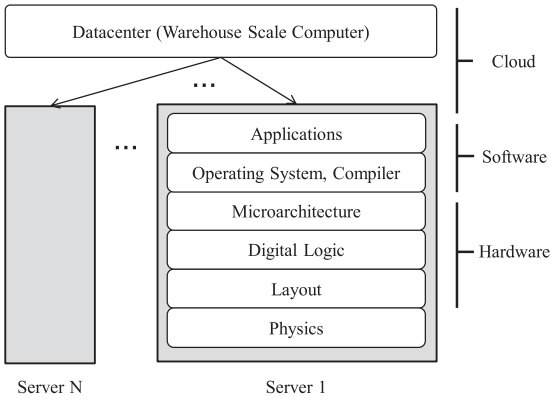

Fig. 1. Layered computing stack model generalized to include the warehouse scale computer as the top most abstraction layer. Note that the stack corresponding to a server node also applies to a mobile device such as a smart phone.

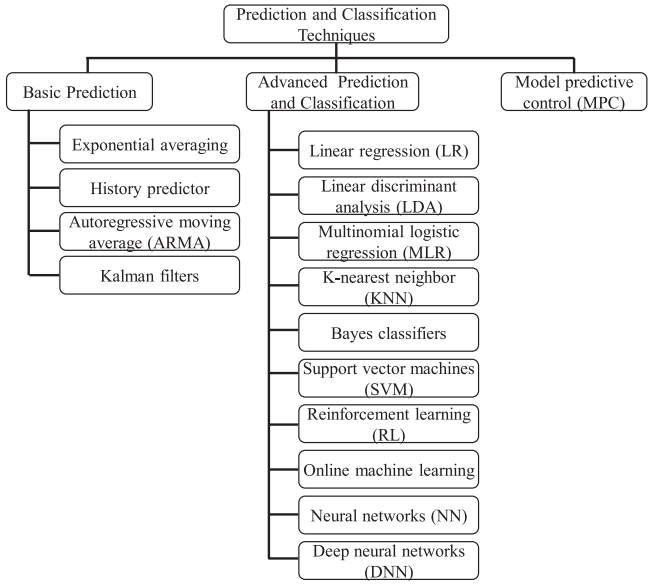

Fig. 2. Techniques discussed in this paper.

additional references. To aid in following the presentation, Fig. 2 presents a tree diagram that includes all the techniques discussed in this paper. Finally, this survey paper is far from being comprehensive. However, we hope it helps to create a good enough picture of what has been and especially what appears to be the most promising prediction techniques. It should serve as a good starting point for the reader interested in employing some form of prediction to be used in optimization solutions across layers in multicore processor systems.

2 Basic Prediction Techniques

In the discussion in this section, we present several prediction techniques from simple to more complex ones, while emphasizing one common underlying theme: each of these techniques exploits in one way or another the past history of the variable of interest. Most of these techniques have been employed at the lower layers from Fig. 1.

2.1 Exponential Averaging

One of the simplest methods for prediction is the exponential averaging. The exponential average predictor uses the following formula to estimate the current value of a variable of interest y at time t, $\hat{y}_t$:

$$\hat{y}_t = \alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}, \tag{1}$$

where y_{t-1} is the measured value of the variable at time $t-1$. α is a user defined weighting factor $0 \leq \alpha \leq 1$. Because the prediction at the current time t involves just the value from the previous step, $t-1$, the history window is of width two only. While this makes this technique easy to implement, its error margin increases considerably when predicting several steps ahead.

Due to its error margin issue, we did not find this prediction technique being used in any approach. Rather, it was used only as a basis for comparison when evaluating more sophisticated prediction techniques. It is included here for the sake of completeness.

2.2 History Predictor

This prediction technique uses a simple formula. The formula employs the previously predicted value $\hat{y}_{past}$ and the average value y_{curr} that is computed based on W samples over a pre-specified history window of length W. It is expressed as follows:

$$\hat{y}_{next} = \frac{W \times y_{curr} + \hat{y}_{past}}{W+1}, \tag{2}$$

where $\hat{y}_{next}$ is the predicted average value of y for the next window.

The primary advantage of the history predictor model is simplicity. Therefore, it is easy to implement in hardware, which is generally more efficient than software implementations. In the context of NoCs, for example, this technique can be used to predict congestion occurrence via proxies like buffer and link utilization. These predictions are then used to decide when to do proactive frequency throttling of selected NoC routers in order to lower the packet transmission rate between different routers, thereby reducing power consumption [4], [5]. In this way, one can develop dynamic voltage and frequency scaling (DVFS) schemes whose objective is to reduce energy consumption with minimal performance degradation. At higher levels of abstraction in the computing stack, history based prediction was employed in [6] to predict workload in a nine node cluster in order to design DVFS based power management schemes. An enhanced form of this prediction technique was proposed in [7], where the authors used a linear formula involving the previous N values of the temperature (as a band limited signal) of a multicore processor to predict the temperature in the near future at time $t + \delta t$.

A global branch predictor is similar to the history predictor. It is constructed mainly with a shift register, whose depth is the length of the recorded history, to store the last observed values [8]. The content of the register indexes a history table that holds previously observed patterns (e.g., thermal patterns if temperature is the predicted variable), with their corresponding next value predictions. Such a predictor was used by [9] to predict power phases in a laptop with a Pentium-M processor.

Similar to the exponential averaging, the history predictor also suffers from increasing errors with horizon. It has been used in several approaches though due to its simplicity and reasonably good results. It also has the advantage of being easily implemented in *Hardware* (Fig. 1).

2.3 Autoregressive Moving Average (ARMA) Model

These models capture autocorrelation in a time series that is assumed to be a stationary process [10], [11]. $ARMA(p, q)$ denotes a model with p autoregressive terms and q moving-average terms. It is described by the following equation:

$$y_t + \sum_{i=1}^{p} a_i y_{t-i} = e_t + \sum_{i=1}^{q} c_i e_{t-i}, \tag{3}$$

where, y_t is the value at time t. e_t is the noise or residual, which is assumed to be random and normally distributed. The autoregressive (AR) and moving average (MA) coefficients are a_i and c_i. An ARMA model is constructed in two phases: 1) identification and estimation and 2) model checking. The model parameters p and q are selected as small as possible to still fit training data reasonably well. The identification and estimation step requires a training data set used to compute the coefficients of the model. Coefficients can be computed by fitting the model using least squares regression in order to identify the parameters which result in minimum errors. In the second step, the model is checked to verify that the model residuals are random. This check can be done using the autocorrelation function. Once an ARMA model is constructed, equation (3) can be used for forecasting the value of y into the future, $\hat{y}_{t+l}$, $l = 1, 2, ...$ being the lead time, as a combination of past values.

The ARMA model described above was used by [11] for thermal management in multiprocessor systems-on-chip (SoCs). The authors showed that ARMA based models can be used to construct temperature predictors that were better than those constructed with exponential averaging, history predictors, or recursive least squares based prediction methods. The proposed technique predicted temperature five steps ahead (the equivalent of 500 ms in realtime) with satisfactory results. The temperature predictions then were used by the thread scheduler to assign threads to different cores in a way that balanced the thermal profile of the chip. Because this solution was implemented mostly in software, it resides in the *Software* abstraction layer from Fig. 1. A similar ARMA model based predictor for temperature was used by [12] to develop a user activity aware thermal management technique in smartphones. The work in [13] used a modified ARMA model to also predict temperature in heterogeneous mobile platforms and reported accuracy of 3 percent. Another example where this model was applied at the datacenter layer is the study in [14] for the purpose of predicting resource demand. The predictions were used to develop a stochastic load balancing scheme with probabilistic guarantees against resource overloading with virtual machine migration.

This model has better prediction accuracy (compared to the previously two discussed techniques) for longer horizons ahead. It can be implemented all in software, which makes it easy to be deployed in existing products. It suffers though from the need for (re)training.

2.4 Kalman Filters

The brief description here is adapted mainly from [15], [16]. The Kalman filter is an adaptive filter applied to predict the state x of a discrete-time controlled process. It uses a set of recursive equations and employs a feedback control mechanism in a way that minimizes the variance of the estimation error [15]. A Kalman filter is constructed in two phases. The first phase is called the *predict phase* and also called the time update phase. Here, the state x is predicted a priori as $\hat{x}_n^-$. The second phase is called the *update phase* and also called the measurement update phase. This is where the predicted $\hat{x}_n^-$ is updated a posteriori as $\hat{x}_n$.

In the predict phase, the filter uses the previous state $\hat{x}_{n-1}$ and the input u_{n-1} to project the state. It also uses the error covariance of the a posteriori error P_{n-1} and the process noise covariance Q to project the error covariance P_n^- for the a priori error. The two equations used in this phase are:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_{n-1} \tag{4}$$

$$P_n^- = AP_{n-1}A^T + Q, \tag{5}$$

where A is the state transition model of the system. B relates the state x to the optional control input u.

The update phase begins after the predict phase with the measurement of the actual state value at time n. It first computes the Kalman gain K_n. K_n is chosen to maximize P_n. Then, the current state matrix $\hat{x}_n$ and P_n are updated. The three equations utilized in this phase are:

$$K_n = P_n^- H^T (HP_n^- H^T + R)^{-1} \tag{6}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(z_n - H\hat{x}_n^-) \tag{7}$$

$$P_n = (1 - K_n H)P_n^-, \tag{8}$$

where R is the measurement noise covariance. H relates the observation or measurement z to the state x.

In the context of dynamic voltage scaling (DVS) for MPEG applications, the study in [16] proposed an extended Kalman filter to estimate the processing time of workloads. In our recent study in [17], we used a similar Kalman filtering approach to estimate the average cycles per instruction and the instruction count for the next control period inside a method for dynamic energy management for NoC based chip multiprocessors with 16 and 64 core architectures. We found that the Kalman filtering based predictions are very accurate and allow the proposed energy reduction heuristic to provide consistent energy savings under a given performance constraint for all benchmarks that we investigated. Also in the context of high performance processors, the authors of [18] proposed a sparse Kalman filter to estimate the states of a dynamical network system. They then applied their solution to the thermal model network of many-core processors to solve the problem of finding the minimum number of in-situ sensors that can be used for both thermal profile estimation and tracking of hotspots in dynamic thermal management solutions.

Kalman filtering is a time-tested technique that was used in numerous application domains due to its high accuracy. Many studies reported achieving the best results with Kalman filtering based approaches. Implementation is

straightforward and versions such as the Sparse Kalman filter [18] eliminate the need even to compute the estimation error covariance or the Kalman gains in real-time, which makes it even more efficient.

3 Advanced Prediction Via Classification Techniques

The description of various machine learning techniques presented here is based on information from several textbooks [19], [20], [21], [22], [23], [24] and other online resources [25]. In some cases, brief descriptions are adapted from other previous papers; when that is the case, the respective papers are cited appropriately.

Recently, there has been a resurgence of machine learning (ML). ML encompasses algorithms that can make predictions or classifications on new data after having constructed models based on training data [19]. It includes primarily three categories. The first type is called the *predictive/supervised learning* approach because it learns a mapping from inputs to outputs, when it is given a set of labeled input-output pairs called the training set. When the output is a categorical variable (from a set of classes), the problem is called *classification/pattern recognition*. When the output is a real valued variable, the problem is called *regression*. The *descriptive/unsupervised learning* is the second category. In this case, only the inputs data are given and the goal is to discover patterns, reason for which this approach is also called *knowledge discovery*. This is more challenging because it is unknown what patterns to search for [20]. As a third category, *reinforcement learning* constructs algorithms to learn how to act in setups with rewards/penalties.

It should be noted that technically most of the machine learning techniques discussed in this section are classification techniques. However, they are used for making predictions indirectly about various figures of merit in the immediate future. For example, classification of the workload of a multicore processor as low predicts that the power dissipation will be low. Similarly, classification of the lifetime reliability of a multicore processor as short predicts higher temperatures. In mobile devices, classification of the operation mode into one of several states translates into prediction of the need for data and location interface configurations, which in turn can be used for proactive measures to save energy. Finally, this is not a comprehensive treatment of these topics and for details on the vast number of machine learning algorithms, the reader should take a look at reference texts such as [19], [20], [23].

An interesting observation is that these techniques have been employed at the higher levels from Fig. 1. That is in part due to the need for training of the models. This requires storage and computational runtime spent during training. Thus, optimization solutions using these techniques are easier to implement for example at the *Software* layer in Fig. 1. Their cost is easier to justify for larger systems composed of multiple compute nodes, such as datacenters, which takes us to the *Cloud* layer in Fig. 1. Nevertheless, optimization solutions using techniques discussed in this section can be used at lower levels of abstraction as well, but at the design time (i.e., statically). For example, the study in [47] uses mining and support vector machine based techniques to predict routability of integrated circuits from placement data.

3.1 Linear Regression (LR)

The description presented here is mainly based on information from [19], [24], [26]. LR is a simple supervised learning approach. It models the relation between one or more independent variables x and the dependent variable y. Relationships are modeled using linear predictor functions. For example, a linear combination of fixed nonlinear functions of the following form can be utilized for multiple LR.

$$y = w_1 f_1(x) + w_2 f_2(x) + \cdots + w_n f_n(x), \tag{9}$$

where, x is the input vector, f_i, $i = 1, \ldots, n$ are known basis functions (e.g., square polynomials), and w_i, $i = 1, \ldots, n$ are unknown parameters that must be estimated from the data. For prediction purposes, LR is employed to fit a predictive model to the set of training observations (x, y). Then, the fitted model is used to make predictions of y for new instances of x [26].

In situations when data are not available all at once but arrive sequentially, it is useful not to restart from scratch the model estimation but simply to update the model on the basis of the newly collected data. This problem is solved by the so called recursive least squares (RLS) estimation. This technique works with a fixed history window and uses data from the window for retraining purposes. Its idea is still to use a polynomial whose coefficients are calculated using least squares estimation. In this way, to maintain good prediction accuracy, the RLS method updates the coefficients repeatedly as new data arrive.

The technique described above was used by [27] for temperature prediction of multicore processors. These predictions served as the basis for a predictive dynamic thermal management algorithm. The algorithm uses core temperatures and their application-specific variation to estimate the thermal profile/behavior. Then, it intervenes through appropriate measures that help to avert thermal emergencies. The study in [98] used recursive least squares to estimate and update a system model parameter matrix. These estimates are used then for DVFS in chip multiprocessors. A multivariate linear regression approach is used in [28] to obtain the coefficients of a linear model that is used to predict execution time of parallel applications MPI tasks executed on clusters of up to 320 nodes. The study [29] uses a constrained-posynomial function (learned through curve fitting) to approximate the power consumption of a many core processor, which generally, is a monotonically increasing function of the frequency. [30] proposed a regression model for the maximum temperature in 3D integrated chip multiprocessors. The temperature was predicted as a linear function of leakage power values. The predictions were used then in a design optimization technique whose objective was to increase the thermal yield.

Regression was applied in the study from [31] to train the popular McPAT power calculator for single-core processors. The paper presented a methodology to calibrate McPAT for a precise power model targeting post-silicon processors. The authors conducted experiments on McPAT against a Cortex-A15 within a Samsung Exynos 5422 SoC and reported mean percentage errors of 2 percent. At the *Cloud* layer, the study in [32] studied regression models to predict energy consumption in cloud datacenters. The authors

reported that regression methods performed better than other techniques including so-called linear and cubic models; they reported 95 percent prediction accuracy.

LR is a rather simple technique at the *Software* layers. Like all other techniques discussed in this section, it requires (re)training to update its coefficients. The linear espression from Equation (9) is efficient, which makes this technique easy to be used in realtime.

3.2 Linear Discriminant Analysis (LDA)

The description presented here is mainly based on information from [24]. LDA is a classification method. It does classification by assigning a new observation $X = x$ to one of K classes. It is a Bayesian approach in the sense that the assignment is done to the class for which the following posterior probability is the largest.

$$\begin{aligned} p_k(x) &= P(Y = k|X = x) \\ &= \frac{P(X = x|Y = k)P(Y = k)}{P(X = x)} = \frac{f_k(x)\pi_k}{P(X = x)}, \end{aligned} \tag{10}$$

where, instead of computing directly π_k and $f_k(x)$, they are estimated, thereby effectively developing an approximator of a Bayes classifier. This estimation is done under certain assumptions however about the form of $f_k(x)$. Usually, the assumption is that $f_k(x)$ is Gaussian; that is, the data in each class are normally distributed. In addition, it is assumed that the variance σ^2 is the same for all classes. In that case, it can be shown that to approximate $p_k(x)$ one can use the so called discriminant function [24]:

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k), \tag{11}$$

where μ_k is the mean for the kth class. Finally, once, μ_k, σ^2, and π_k are estimated as $\hat{\mu}_k$, $\hat{\sigma}^2$, and $\hat{\pi}_k$ based on the training data, then, the LDA classifier operates according to the following expression:

$$\arg\max_K \hat{\delta}_k(x). \tag{12}$$

The above LDA classification approach was employed in [33]. The authors provided a comparison of several machine learning algorithms, including LDA, to predict mobile device location interface and data configurations that reduce energy consumption. Based on three target variables (*Fine Location Required*, *Coarse Location Required*, *Data Required*), they partitioned the location interface and data configurations into eight classes. The idea then was to efficiently predict one of these eight classes using device context, spatial, and temporal input variables and, thus, to know when to shut down location and wireless radios to be able to save energy.

LDA is fast and relatively easy to implement in practice. It has been popular especially in situations with more than two classes. Sometimes it can provide results as good as more complex models.

3.3 Multinomial Logistic Regression Model

Multinomial or multi-class logistic regression generalizes logistic regression to cases with more than two outcomes [26]. For example, we can think of predicting the type of workload of a processor, say (low, medium, high), based on the given outcomes of several observations. In this case, the dependent variable that we want to predict is the workload. The independent variables or features can be observations such as instruction and activity counters, cache misses, etc.

In other words, multinomial logistic regression is a discriminative classifier applied to a multinomial variable. It predicts the probability distribution over a set of classes from a sample input to learn a direct mapping from the input sample to the output class. The logistic regression based classification is composed of two steps: 1) modeling to estimate the probability distribution of the different classes for a given input, and 2) parameter fitting to estimate the parameters of the logistic regression model.

Following the brief description and notation in [34], in the first step, the multinomial logistic regression model works with the assumption that the value of the variable of interest, $y \in [1, 2, \ldots, K]$, is predicted based on the N values of the input feature set, which are identified as $X = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{1\times N}$. The model is represented by the hypothesis h_β, with parameter $\beta \in \mathbb{R}^{(K-1)\times N}$. Then, it can be shown that for a given input feature set X, the logistic regression model outputs $h_\beta(X)$ is given by:

$$h_\beta(X) = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{K-1} \end{bmatrix} = \begin{bmatrix} \frac{e^{\beta_1^T \cdot X}}{\sum_{j=1}^{K} e^{\beta_j^T \cdot X}} & \ldots & \frac{e^{\beta_{K-1}^T \cdot X}}{\sum_{j=1}^{K} e^{\beta_j^T \cdot X}} \end{bmatrix}^T. \tag{13}$$

Equation (13) is used to select the output of the overall model as the class y given by the expression:

$$\arg\max_k \{p_k \,|\, \forall k \in [1, 2, \ldots, K]\}, \tag{14}$$

where p_k is the probability $p_k = Pr(y = k), k \in [1, 2, \ldots, K]$.

The second step of the multinomial logistic regression model is the estimation of the parameters β. That is done using a training set of M samples generated independently and identically. For each of these samples, the input feature X and the output class y are known a priori and the input-output pairs are identified as $(X_i, y_i), \forall i \in [1, 2, \ldots, M]$. The parameters β are calculated using maximum a posteriori estimation. Usually, the solution to the problem described by equation (14) is found by gradient-based optimization algorithms.

The study in [34] proposed such a multinomial logistic regression-based classification technique that classifies the workload (i.e., CPU cycles) at runtime into a fixed set of K classes. The variable of interest y is the workload while X specifies the workloads of the previous N video frames, where x_i is the workload of the i^{th} previous frame. In other words, the class of the next video frame is predicted based on the workloads of the N previous frames. Each workload class corresponds to a frequency that is predetermined using training data. At runtime, the classified frequencies are applied to the processing cores within an DVFS algorithm. Results obtained on an embedded multicore system running standard multimedia applications demonstrated an average of 20 percent reduction in energy consumption.

A multinominal logistic regression classifier was developed by [35], [36] based on data collected from performance counters during offline workload characterization. The classifier is used then during applictaion runtime to predict the workload and to select the frequency and thread packing such that performance is maximized under a given power cap. The study in [33] also presented a multinominal logistic regression or linear logistic regression based solution to make predictions about the states of mobile devices. It was found that this solution was outperformed by neural networks and K-nearest neighbor based solutions.

These models tend to have better performance than a series of binary logistic regressions as they can model synergistic relationships. However, they are somewhat complex and sensitive to outliers.

3.4 K-Nearest Neighbor (KNN)

The short description here is adapted mainly from [24], [37]. As a non-parametric supervised approach, the k-nearest neighbor (KNN) algorithm is commonly used for classification or regression problems. In the training phase, feature vectors and class labels are simply paired using k labels, where k is a user defined constant. The output of the algorithm depends on if KNN is used for classification or regression. In the former case, the output is a class membership. The classification of a new sample or object is done by placing it in the class that is shared by the largest number of k closest neighbors. To quantify closeness, the KNN algorithm employs a distance measure, such as the popular euclidean distance or Hamming distance.

One of the techniques studied in [33] used a KNN model, where the number of attributes defining the input feature space was 19 (including attributes such as day of week, device moving, and battery level) and the number of classes was 8. These eight classes corresponded to eight different combinations of three variables (*Fine Location Required*, *Coarse Location Required*, *Data Required*). However, the KNN model was outperformed by other models including support vector machines.

Despite its simplicity, the KNN algorithm can build classifiers that are very close in performance to the Bayes classifier. One of its limitations though is that it is sensitive to the local structure of the data, which makes the selection of k difficult. There have been however, various heuristics proposed to select a good k. Also, it can be computationally slow.

3.5 Bayes Classifiers

The brief description here is adapted mainly from [38], [39]. The Bayesian classifier is a supervised learning model. It uses a learning agent that builds a probabilistic model of the features that is then employed to predict the classification of new features or examples. The naive Bayes classifier assumes that the input features are conditionally independent. It is constructed by combining a naive Bayes probability model with a decision rule. For a new instance to be classified, denoted as $\mathbf{x} = (x_1, \ldots, x_n)$ representing the n features, the naive Bayes probability model assigns probabilities $p(C_k|x_1, \ldots, x_n)$ to this problem instance. This assignment is done for each of the K classes C_k. It can be shown that the conditional distribution over the class variable can be expressed as:

$$p(C_k|x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i|C_k), \tag{15}$$

where $Z = p(\mathbf{x})$ is a constant scaling factor that depends on $(x_1, \ldots, x_n)$. The classifier can then be constructed as the function that assigns a class label $\hat{y} = C_k$ according to the following equation:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \prod_{i=1}^{n} p(x_i|C_k). \tag{16}$$

The study in [40] proposed a Bayesian classifier for energy management. Only information about the occupancy state of the global service queue is used for learning to predict the system performance. The predicted performance is then used to select the frequency from a pre-computed policy table. The authors reported that this classifier was more efficient than other methods.

While it was proven to provide good results, this model has not been very popular so far. We include it here again for the sake of completeness.

3.6 Support Vector Machines (SVMs)

An SVM is a model employed by supervised learning methods used in classification and regression [23], [24]. Typical models comprise linear combinations of fixed basis functions, which need to be adapted to the data in order to be able to apply these models to large scale problems [19]. To do that, the SVM model defines basis functions centered on the training data and later during training selects just a subset of them. In the case of the two-class classification problem, the following linear model is used:

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b, \tag{17}$$

where $\boldsymbol{\phi}(\mathbf{x})$ represents a transformation in the feature space. $\mathbf{w}$ is a parameter vector and b is a bias. The classification of a new input $\mathbf{x}$ is done by the sign of $y(\mathbf{x})$. The training data is a set of pairs of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$ and their targets $t_1, t_2, \ldots, t_M$, $t_n \in \{-1, 1\}$. Under the assumption that training data are linearly separable in the feature space, then, there are multiple values of $\mathbf{w}$ and b for which Equation (17) gives $y(\mathbf{x}) > 0$ for pairs with $t_n = +1$ and $y(\mathbf{x}) < 0$ for pairs with $t_n = -1$. The SVM model chooses those parameter values that maximize the so called margin, which is the smallest distance between the decision boundary, Equation (17), and any training sample. In this way, the SVM training algorithm constructs a line for binary classification or a hyperplane for higher dimensionality. If the earlier assumption on separability is not valid, the SVM model can use nonlinear kernel functions to map the original space to a higher-dimensional space, where separation can be made linear.

In the context of increasingly popular heterogeneous platforms with multiple CPUs and GPUs, the study in [41] presented an efficient OpenCL task scheduling algorithm to schedule multiple kernels from multiple programs. The scheduler is based on a model constructed with an SVM classifier that predicts speedup of a kernel based on its static code structure. Results were reported on two different

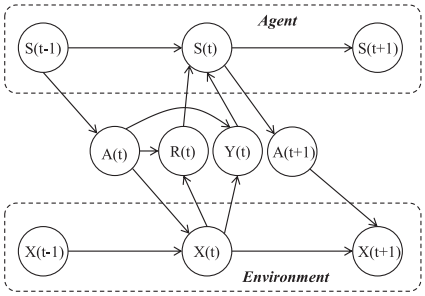

Fig. 3. Model of reinforcement learning problem adapted from [50].

systems, Intel Core i7 4-core CPU + NVIDIA GeForce GTX 590 GPU and AMD HD 7970 GPU. This SVM based prediction approach was improved upon by the work in [42], who also tested their scheduler on a real system, the Intel Haswell Core i7-4790K CPU-GPU processor. At the datacenter layer, the authors in [43] developed an SVM model for temperature prediction in datacenters. The study in [33] investigated SVMs for energy optimization in smart phones. SVMs were found to provide the best prediction accuracy together with neural network models. The study in [44] applied several machine learning techniques, including SVMs and decision trees, to energy-efficient context sensing for mobile devices. The proposed models learn relations among different classes of sensors (e.g., *light-duty sensors* that are energy-efficient, implemented in software, and *heavy-duty sensors* that consume high energy and are implemented in hardware) and then, exploit those relationships to infer the status of high-energy-consuming sensors. If this inference says that the sensor is stable, then, the sensor is not triggered and the latest sensor value is utilized instead as the estimation, thereby saving the energy that the sensor would have consumed.

The study in [45] used an SVM based approach to predict aging induced delay in integrated circuits, including the Leon3 and OpenRISC processors. Their solution consisted of a runtime monitoring infrastructure that exploited space and time sampling of a reduced number of latches. Training was performed offline using support-vector regression and the prediction model was implemented in software. Other previous studies used SVM to predict embedded memory timing failures during the floorplanning stage [46], routability of integrated circuit placements [47], latency in networks-on-chip [48], network-on-chip configuration links [49].

The SVM model is very effective practically, and therefore, it has become very popular in modern machine learning. On the downside, the number of basis functions can increase with the size of the training data set [19] and the parameter selection is data dependent.

3.7 Reinforcement Learning (RL)

The description here was adapted primarily from [50], [51], [52]. In the RL approach, an agent operates in the environment with the goal to maximize the total accumulated reward. The RL model can be described with the help of Fig. 3. The agent gets an observation $Y(t)$ and a reward $R(t)$ at each discrete time step t. Next, the agent selects an action $A(t+1)$, from the set of actions $\mathbf{A}$. The selected action is sent back to the environment. The environment, in turn, transitions to the new state $X(t+1)$. At the same time, the reward $R(t+1)$ corresponding to the transition $(S(t), A(t), X(t+1))$ is calculated; where $S(t)$ is the agent state at time t from the set of possible states $\mathbf{S}$.

Both the environment and the agent are modeled as stochastic finite state machines. The agent receives observations and rewards as inputs. The outputs from the agent represent actions sent back to the environment. The policy function is $A(t) = \pi(S(t))$ and the state transition function is $S(t) = f(S(t-1), Y(t), R(t), A(t))$. The goal of the agent is to accumulate as much reward as possible. That can be done with a policy and agent state-update function that maximizes the expected value of the summation of rewards:

$$E[R(0) + \gamma R(1) + \gamma^2 R(2) + \cdots] = E\left[\sum_{t=0}^{\infty} \gamma^t R(t)\right], \quad (18)$$

where $0 \leq \gamma \leq 1$ represents the discount factor. This factor signifies that immediate reward is worth more than future reward [20].

Reinforcement learning has been very popular especially in developing energy and thermal management solutions for processors. Because these solutions were developed mostly in software, they are located on the *Software* layer in Fig. 1. For example, the study in [53] presents a reinforcement learning solution to the problem of adaptive thermal management in multicore systems with the goal to improve lifetime reliability. Q-Learning was used as the algorithm to learn the relation between the clock frequencies and temperatures of the cores and the mapping of threads to cores. In Q-Learning, a learning agent maintains a Q-Table with entries called Q-values that correspond state-action pairs. These entries are referred to also as Q-values. Based on readings from thermal sensors and performance counters, the operating system calculates the thermal stress and aging. The values of *stress* and *aging* represent the Q-Table states. In RL terminology, they model the *environment* basically. Actions are taken as dynamic changes of cores frequency that override OS thread mapping decisions. In this way, peak and average temperatures are controlled such that lifetime is improved.

Reinforcement learning based on Q-learning was used by the authors of [54] to develop an online power management technique for multicores. Their technique achieved autonomous management by dynamically adapting to the environment without prior information about the workload. Another Q-learning based dynamic voltage and frequency scaling algorithm is presented in [55]. Other studies, used various reinforcement learning based approaches to learn the optimal control policy of the VF pairs in many core processors for power optimization [56], user behavior with respect to the use of embedded network-on-chip platforms [57]. A Q-learning based I/O management was proposed in [58] to adaptively adjust the I/O output-voltage swing in 2.5D integrated many core microprocessors and memories, under communication power and bit error rate constraints. The authors of [59] presented a reinforcement learning based runtime manager for energy-efficient thermal management of embedded systems. The approach addressed thermal cycling and average and peak temperatures simultaneously. An online DVFS control strategy based on core-level modular reinforcement learning to adaptively select

appropriate operating frequencies for each individual core was proposed in [60]. An Q-learning based algorithm was proposed in [61] to identify V/F pairs for predicted workloads and given application performance requirements. The study in [62] investigated imitation learning and reported higher quality policies in the context of dynamic VFI control in many core systems with different applications running concurrently.

Note that RL based solutions do not make direct predictions of metrics like temperature, power, etc. Instead, predictions are made for the *state* in which the system is or will be and then *actions* are taken such that the system is geared towards desired states, which are characterized by desired values of the metrics of interest such as performance and lifetime reliability. Nevertheless, we are still dealing with prediction here.

Q-learning has been very popular because it is simple and robust to noise. On the limitations side, it may not be able to identify the optimal policy if the environment is not a Markov decision process.

3.8 Online Machine Learning

Online or sequential learning is used when data are provided sequentially (i.e., streaming data) [19]. It involves a sequence of consecutive steps to develop a mapping between data and corresponding labels. In each step, the learning agent is asked a question that is answered by employing a prediction technique. This prediction technique, also called a hypothesis, represents a mapping between questions and acceptable answers. The learning agent receives the correct answer after each predicted answer. A loss function is utilized to quantify the discrepancy between prediction and the correct answer. In this way, the total accumulated loss after a set of question answer rounds measures the performance of the online learning algorithm. In achieving the goal of minimizing this total loss, the agent can dynamically update the hypothesis in order to improve its chances of giving the correct answer in future steps.

Online learning was employed by the techniques proposed in [63], [64], [65] to select the most appropriate frequency for the processing cores based on the workload characteristic of a given application. For example, the study in [63] introduces a DVFS technique for a multi-tasking framework. The authors proposed a control algorithm to characterize the behavior of a given task and to select the best voltage-frequency (VF) pair setting. The characterization employs runtime statistics such as IPC and cache hit/miss ratio. The chosen or predicted VF pair is expected to minimize both the energy consumption and performance delay. Implemented as a software technique, this policy is lightweight and has negligible overhead.

The study in [66] proposed an online learning temperature management technique for multicore systems. The objective of the technique is to reduce the adverse effects of temperature variations and hotspots. It achieves that by employing online learning, based on switching experts [67], to choose the best policy from among a given set of expert policies for the current workload characteristics. This online learning solution facilitates realtime adaptation to react to the changing workload. This adaptation consists of the selection of the policy with the desired trade-off between thermal profile and performance.

Due to its inherent architecture, this approach can offer good accuracies. Thanks to its realtime adaptation ability, it can naturally estimate workloads that were not encountered before.

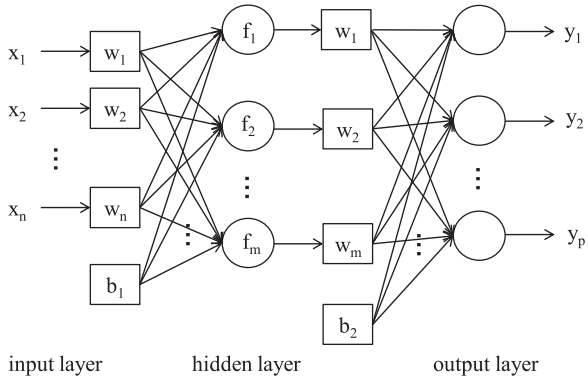

Fig. 4. Architecture of a two-layer neural network model.

3.9 Neural Network (NN) Models

A popular machine learning approach is the neural network or multi-layered perceptron model. Because NNs are very good at identifying trends and discovering patterns in complex data, they have been utilized in numerous applications (e.g., pattern recognition and data classification). An NN model is essentially formed by connecting a number of neurons (also called processing elements), typically organized on several layers. For example, the block diagram of a two-layer neural network is shown in Fig. 4. In this model, each layer implements the transfer function:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{19}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of a given layer. $\mathbf{W}$ is the matrix of weights of size $m \times n$ with n being the number of inputs and m being the number of neurons in the layer. $\mathbf{b}$ is the bias vector of size $m \times 1$. Examples of layer transfer functions include *tansig* and *purelin*. For example, the former one is given by: $f(u) = 2/(1 + e^{-2u}) - 1$.

An NN model must always be trained first. Training requires a set of known input and output data pairs, which sometimes is difficult to obtain. It is done by an algorithm that uses the training data to estimate numerical values for weights and biases. Once trained, the NN model can be utilized to provide estimations on new data of interest. Such estimations are usually more accurate when the training is done with sufficiently large training data sets.

NN models have been used to predict temperature [68], [69] in chip multiprocessors (CMPs) to construct thermal management strategies. The study in [70] uses a similar NN model to predict the lifetime reliability of CMPs. The reliability predictions are applied to dynamic reliability management. The neural network oracle can utilize as inputs the current temperatures, supply voltages, and clock frequencies of each tile as well as their variation trends. These studies used NN models with two layers similar to the example from Fig. 4. The authors of [71] develop an Artificial Neural Network (ANN) based mechanism for

network-on-chip (called uncore) power management. The offline training of the ANN is augmented with a simple proportional integral (PI) controller as a second classifier. The ANN is used to predict the NoC utility (defined as the performance sensitivity to the NoC), which is then used to make DVFS decisions that lead to improvements in the energy-delay product. ANNs have been used also in branch prediction techniques [72] as well as to predict congestion hotspots in networks-on-chip [73]. In [75], NN models were used to estimate power and thermal profile in NoCs based on the utilization of NoC nodes and links. These predictions were then used to configure the global optimal NoC, which was considered a reconfigurable communication resource. An NN model was used to predict core temperatures in supercomputers in the study from [76]. The predictions were used to develop a preemptive fan control mechanism and a thermal-aware load balancing algorithm. Experiments were reported on an IBM cluster with POWER8 processors, each node in the cluster with 2 sockets, and each socket with 10 physical cores. Results reported that peak fan power can be reduced by 61%. Another application of NN models was reported in [77]. The authors used measured performance and power data from real GPU hardware to train an NN model and to capture how applications scale as the GPU's configuration is changed. The model was then used to estimate the performance and power of new applications at different GPU configurations and was reported to offer within 15 percent accuracy.

An NN based model with eight outputs for different interface configurations of a mobile device was presented in [33] to do classification. Such classification is used as the basis for setting the mobile device into that configuration state with the goal of reducing energy consumption. It was reported that the NN model provides together with the SVM model the best prediction accuracy. Other studies have used NN models for matching (i.e., predicting the best microarchitecture) processor microarchitecture in energy harvesting systems to dynamically adjust the microarchitecture to achieve the maximum forward progress [74].

Main advantages of the NN model include: it can model intricate nonlinear relationships and can capture useful meaning even from imprecise data. However, NN models usually suffer from long computational runtimes required for training. Also, model overfitting can become an issue.

3.10 Deep Neural Networks (DNNs)

The description here was adapted primarily from [20], [22], [78]. Structurally, a DNN model is a multi-layer perceptron (MLP), which is just a feedforward Artificial Neural Network with many hidden layers. The main difference compared to traditional NNs is that DNNs have more hidden layers. That helps DNNs to capture more complex nonlinear relationships [79]. A turning point in the world of deep learning took place in 2006, when Hinton and colleagues [80], [81] showed that deep belief networks (DBNs) can serve as the basis for DNNs pretraining. They showed that one can effectively pretrain a DNN one layer at a time. That can be done by handling individual layers as unsupervised restricted Boltzmann machines (RBM) separately. Then, the entire stack of layers can be fine-tuned using supervised backpropagation. Moreover, the pretraining can also be followed by other discriminative learning techniques to further fine-tune the weights. During this process, a final layer is added to the DNN [82] as shown in Fig. 5. The variables on this final layer are the desired outputs from the training data. These outputs of the final layer will be used directly for classification purposes.

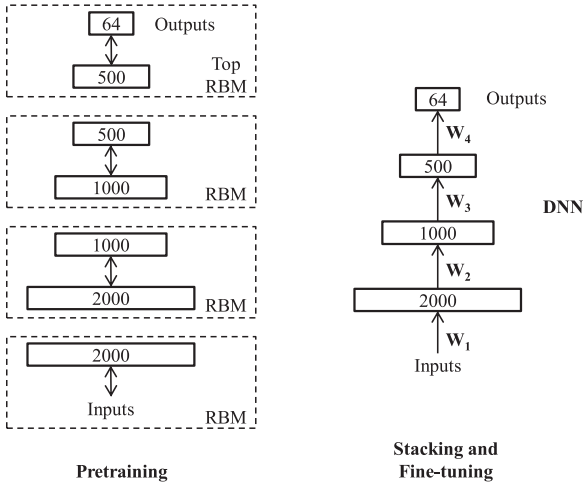

Fig. 5. Block diagram of the DNN model formed by a stack of restricted Boltzmann machines. The number of units in layers are for illustration purposes.

These developments were crucial to the advent of deep learning, which received a lot of attention recently. For example, the successful application of DNN models in speech recognition at an industry scale provided recognition rates that improved with 30 percent compared to the Gaussian mixture model (GMM) based traditional methods. Such improvement was considered as "the most dramatic change in accuracy since 1979" [83]. As another example, DNNs were used for traffic sign classification and achieved a better-than-human recognition rate of 99.46 percent [84]. These developments represented a remarkable moment that triggered the resurrection of DNNs, which have been shown to lead to some of the best results in several different application domains [85].

While very popular in other application domains, DNN models have been used less so far in multicore processors systems. Nevertheless, at the highest level in Fig. 1, in the context of datacenters—where huge amounts of data points are generated continuously by large numbers of sensors—recently, the study in [86] proposed a DNN model to model plant performance and to predict power usage effectiveness (PUE) with very good accuracy. Their DNN model was constructed with five hidden layers and fifty nodes per hidden layer. The training data set contained 19 input variables (which included the number of running cooling towers, the total server IT load, the outdoor wind speed, and others) and one output variable as the PUE. All these variables represented about two years of operational data collected as 184,435 time samples at 5 minute steps. Testing and validation at Google's datacenters, the DNN model was shown to be an effective approach to exploit existing sensor data to model datacenter performance and to identify operational parameters that improve energy efficiency and reduce the PUE [86].

DNNs have a high modeling power because they use hidden layers with many neurons. However, that comes

with the price of increased computational complexity during training. This may be one of the reasons why it has not been very popular especially at lower levels in the computing stack from Fig. 1. In addition, sometimes, the performance achieved with DNNs is not as good as that of some probabilistic models.

3.11 Additional Machine Learning Approaches

In this section, we briefly discuss other predictions and classification approaches that could not be fit in any of the categories discussed earlier. Machine learning techniques for performance and power modeling of single-core processors were discussed in [87]. More specifically, to predict the performance of a workload on a target platform, the study in [88] introduced a statistical learning approach. The model was verified for the ARM CPU model (five-stage in-order) and the authors reported an average accuracy of 90 percent. Decision tree learning—a predictive model that represents observations as branches leading to conclusions, i.e., leaves, about the attribute's value - was used to construct a temperature prediction model for black-box IPs in [89]. Another type of machine learning technique, belief rule based expert systems (BRBES), was used in [90] to predict energy efficiency in datacenters. Consisting of two components, knowledge-base and inference engine, the BRBES model was reported to perform better than ANNs. The study in [91] used ridge regression to predict the number of packets to be injected into routers of NoCs built with photonic interconnects in heterogeneous multicore CPU + GPU processors. The predictions were used to develop a proactive method to determine the amount of laser power needed in a specified reservation window at each router.

The extreme learning machine algorithm proposed in [92] exploited history of signal quality and strength to predict location of mobile devices. The authors found that their approach outperformed KNN based solutions. Also in the context of mobile device, a supervised learning based prediction model was proposed in [93] to predict spatial context. A nonparametric predictive modeling scheme implemented using boosted regression trees was proposed in [94]. The objective was to capture the correlation between processor configurations, workloads, and execution phases and exploit that toward improving various performance metrics including the architectural vulnerability factor. A recent research trend is in compositional structures in social dynamics. The focus is on data generated by social media applications. For example, the study in [95] proposed a new probabilistic model called recursive convolutional Bayesian model to model signatures of social dynamics. They explored the potential of their model for supervised learning in social applications. As such, they reported predictions made for the users count of a hashtag on a Twitter dataset and for the average number of checkins on a daily basis for a business on Yelp for a year. This class of prediction approaches are located on the top most abstraction layer, *Cloud* in Fig. 1, because they operate in the cloud by working with data generated by datacenter level applications. Similarly, most of the recommender systems (e.g., Netflix, Amazon, etc.), where the users' preferences are predicted, operate at the *Cloud* layer in Fig. 1. We do not review these approaches in this paper due to lack of space.

4 Other Prediction Approaches

4.1 Model Predictive Control (MPC) Techniques

MPC is a special approach to prediction, which is not explicit as in most other cases. It is an optimal control technique for linear dynamic systems [96]. Its objective is to maximize a certain performance metric. It is called predictive because the problem is formulated over a period of a certain number of steps with the start point being the current time. The solution to the problem provides the sequence of future feedback control actions (e.g., frequency settings for the cores of a multicore processor). These actions can be derived by numerical solvers embedded directly in the control algorithm. Alternatively, the actions can be precomputed statically, at design time, stored in a look-up table, which is then referenced during realtime operation.

The model predictive control was used for thermal management to achieve smooth control with minimal performance loss in [97]. An optimal control theory based algorithm was proposed in [98] for the chip-level power control of a multicore processor with the objective that the temperature of each core be maintained below a specified threshold. Another example is the study in [99], where the authors developed thermal management policies for chip multiprocessors. Using DVFS as a control mechanism, these policies manipulate the time-varying workloads and thermal profile in a way that improves the thermal balancing of the CMP die. A workload aware approach is proposed in [100] based on control theoretic principles. At the datacenter layer, the study in [101] presented ThermoRing, a model-predictive control based scheduling strategy to reduce cooling costs in data centers.

While not a straightforward prediction/classification technique, MPC was successfully used to develop closed-loop control approaches for thermal management in multicore processors. This technique is a good example of a cross-layer technique at both *Hardware* and *Software* layers.

4.2 Others

Here, we discuss several prediction techniques that we could not fit in any of the categories discussed in this paper. Exploiting a cause-effect rationale, the study in [102] developed an NoC traffic prediction method by looking at the application cache coherence behavior. Implemented in *Hardware*, the technique achieved 87 percent accuracy. Similarly, but in the context of cloud datacenters, the study in [103] introduced a so-called heat imbalance model that is used to predict future temperature trends. These predictions are then used to develop a proactive thermal-aware virtual machine (VM) allocation algorithm that minimizes energy consumption for computation. Using information about the current temperature and power, the study in [104] proposed a low overhead future temperature predictor in heterogeneous multiprocessor systems-on-chip (MPSoC). The proposed technique is based on a compact thermal model of the chip, which captures the temperature dynamics and the relation between the temperature, cores power consumption, and thermal characteristics of the system via a non-homogeneous system of differential equations. The predictor was used to develop temperature aware scheduling techniques that can avoid proactively power states that could leading to future thermal emergencies.

5 Discussion

A summary of the predicted variables or attributes of interest in the reviewed literature is presented in Table 1. In addition, we are making the following observations.

- We observe that as we move to higher levels of abstraction in the computing stack model from Fig. 1, we find that increasingly complex or sophisticated prediction methods are employed. While for example at the NoC router level simple history predictors would be good enough, at the datacenter level already deep neural network (DNN) models are able to capture the relationship between and the impact of 19 different normalized input variables on the power usage effectiveness. Noteworthy is that such increasingly sophisticated models require usually large training datasets and long training times too. Large training datasets in turn means storage too, which is not something that we could afford for (realtime) prediction methods at lower levels in the computing stack.
- Therefore, a question that is interesting to ask is how far down in the computing stack can we go with DNN models such that we could benefit from their power of modeling intricate relationships across long histories of operation while keeping the model size (complexity and required storage) small enough to justify the benefits from using such models? While there have been previous studies that used neural network (not deep) models, it is still unclear what the answer to the above question is.
- At lower levels in the computing stack, Kalman filtering technique, as a time tested solution, does a very good job at predicting the near future based on a relatively short recent history. It can easily be implemented in software and hardware. This technique is a very good compromise between effectiveness and computational and storage complexity.
- Support vector machine (SVM) models have been employed by many previous studies. This is a very popular technique that has been used for predicting many different figures of merit (see Table 1). While this technique does require training data and static (i.e., design time) model construction and optimization, we attribute its popularity to its practical effectiveness. It does a good job for the practical applications for which it has been used.
- While simpler prediction techniques (such as those discussed in the first part of this paper) will continue to be utilized in rather simple design optimizations at the lower levels in the computing stack, we project that more complex techniques (like those discussed in the second part of this paper) will be increasingly employed, especially as we move more and more towards datacenter/warehouse scale computer or even exascale computing. At such "cloud computing" levels of abstraction, a lot of data is usually collected and stored anyway. So, it is natural that DNN models and data mining are looked at because of their ability to capture relationships and predict behavior that was not possible before. Such improved models and prediction techniques can be utilized to provide optimization opportunities not seen before. Along this line, we see already substantial work done on the topic of recommender systems, where the users' preferences are predicted. Similarly, opinion and sentiment analysis has received a lot of attention [105].
- A different class of optimizations is that where the human user takes a central role. These user centric techniques have been advocated especially for mobile embedded computing platforms [106], [107], [108], [109], [110], [111]. These techniques model and then predict the user behavior to identify optimization opportunities for reducing energy consumption without degrading user-perceived performance. Here, we note that the distinction between user behavior and workload behavior is rather vague. That is because the way a certain computing system, such as a smart phone, is exercised with workload reflects the user behavior too. So, one can argue that workload behavior captures the user implicitly. There is however another user-related variable—that of psychology of users—which can be exploited towards further energy savings. For example, the study in [111] exploits psychological changes during the low battery phase of mobile devices of different users to design a quality of experience (QoE) aware frequency governor. The role of the governor is to dynamically change the processor frequency in order to operate at the best QoE at for different users in different environments during low battery phases. We expect more work along this line will emerge to benefit from optimization opportunities from angles not fully exploited yet.

Furthermore, we present Table 2 to indicate the usage of the various prediction/classification techniques across different layers of the computing stack. Thus, in this table, the columns represent the layers from Fig. 1 and the rows represent the techniques discussed throughout this paper. Layers from Fig. 1 are grouped within the columns in order to keep the table compact and because the implementation of the prediction techniques tends to span across these layers. An entry in this table indicates what attribute(s) of interest the prediction technique (i.e., row) was employed for prediction/classification at the layer(s) indicated by column. Note that, techniques appearing in the column *OS/Compiler, Apps* that involve DVFS or some form of system (re)configuration generally require support at the *Hardware* level too. In other words, these techniques are usually spanning multiple layers in the computing stack. However, we included entries in the *Dig. Logic, Microarch.* column where the cross-layer aspect was clearly stated in the surveyed prior works. In addition, note that for a given layer, we do not specify what technique is the best in terms of accuracy or impact on design optimization because most often than not these techniques are used to predict/classify different attributes—and this makes such comparison difficult.

Instead, we present Table 3 as a quick look-up table that summarizes the pluses and minuses of each major technique that we discussed in this paper. In addition, in Fig. 6, we summarize what we observed as being the most popular prediction techniques. The popularity of the techniques in this figure was qualitatively measured in terms of 1) the

TABLE 1
Summary of Discussed Prediction Techniques

	Power Energy	Workload	Perf. utiliz. IPC	Temp. CMPs	NoC buffer/link utiliz.	NoC congest. hostspots	NoC Utility Latency	Reliab. CMPs	Aging	VF Setting	Memory timing	Routability	Others
Exponential averaging				[11]									
History predictor	[9] (power phase)	[6] (cluster)			[4], [5]								
ARMA			[11]	[11]									[18] (thermal state)
Kalman filtering		[16]	[17]	[18]									
Linear regression	[30], [30]		[28] (exe. time)	[27]									[98] (model param.)
LDA													[33] (mobile config.)
Multinomial logistic regr.		[34], [35], [36]											[33] (mobile config.)
KNN													[33] (mobile config.)
Bayes classifier													[40] (perf. state)
SVM							[48] (latency)		[45]		[46]	[47]	[49] (NoC config.)
Reinforcement learning													[53], [55], [56] (DVFS policy) [57] (user beh.)
Online learning										[63], [64], [65]			[66] (policy) [67] (expert)
Neural Networks	[68], [69]					[73]	[71] (utility)	[70]					[74] (uarch)
DNN													[86] (PUE)
Others	[106],–[111] (energy via user beh.)												[91] (num. packets) [98] (pred. ctrl.) [111] (psych. traits)

Representative examples are shown only.

TABLE 2
Prediction Techniques were Applied at Different Abstraction Layers

Technique	Physics, Layout	Dig. Logic, Microarch.	OS/Compiler, Apps	Datacenter
Exponential averaging		Temp. CMPs		
History predictor		NoC buffer/link util.	Workload, Temp., Power phase	
ARMA		Perf. utiliz. IPC	Temp. CMPs	Resource demand
Kalman		CPI, Instr. count	CPI, Instr. count	
filtering		System states, Temp. CMPs	Processing time	
		Perf. utiliz. IPC		
Linear		Temp., Model param.	Temp., Exe. time	Exe. time
regression		Power consump.	Power consump.	Energy
LDA		Device config.	Device config.	
Multinomial			Workload	
logistic regr.			Device config.	
KNN			Device config.	
Bayes classifier			System perf.	
SVM	Routability of IC	Aging induced delay	Device config., Sensor status, NoC latency	Temp.
		NoC latency	Aging induced delay, Mem. timing err. NoC config. links	
Reinforcement			Stress/Aging-Clock freq., Power-Clock freq.	
learning			Error rate-I/O voltage swing Energy/Power-DVFS, Thermal profile-DVFS	
Online			Workload-Clock freq.	
learning			Temp./Hotspots-Workload policy	
Neural		Temp., Lifetime reliability	Temp., Lifetime reliability, Branch pred.	Temp.
Networks		NoC utility, NoC congestion	NoC utility, Device config.	
DNN				PUE
Model Pred. Control		Power, Temp.	Workload, Thermal profile	Power, Temp.

An entry indicates predicted attributes.

TABLE 3
Summary of Pros and Cons of the Surveyed Prediction Techniques

Technique	Pros	Cons
Exponential averaging	Simple, easy to implement	Error margin increases with horizon
History predictor	Easy to implement	Error margin increases with horizon
ARMA	Can be implemented all in SW	Identification and estimation require training
	Good accuracy for a few steps ahead	
Kalman	Very good accuracy	Somewhat complex to implement
filtering	One of the best accuracy-cost points	
Linear	Good prediction accuracy	Requires training
regression		May need to update coefficients repeatedly as new data arrive
LDA	Fast, relatively easy to implement	Requires training
	Can provide results as good as more complex models	
Multinomial	Good performance performance	Requires training
logistic regr.	Can model synergistic relationships	Somewhat complex, sensitive to outliers
KNN	Simple, great accuracy	Sensitive to the local structure of the data
		Can be computationally slow
Bayes classifier	Good accuracy, efficient	Requires training
SVM	Effective practically, very popular	Number of basis functions can increase with size of training data
		Parameter selection is data dependent
Reinforcement	Simple and robust to noise, popular	May not be able to identify optimal policy if environment
learning		is not a Markov decision process
Online	Relatively simple, good prediction accuracy	Medium complexity of implementation
learning	Good realtime adaptation	
Neural	Can model intricate nonlinear relationships, popular	Long computational runtimes required for training
Networks	Can capture useful meaning even from imprecise data	Model overfitting can become an issue
DNN	High modeling power	Increased computational complexity during training
		Network topology design can be tricky
Model Pred. Control	Good accuracy over several steps ahead	Not an explicit prediction technique

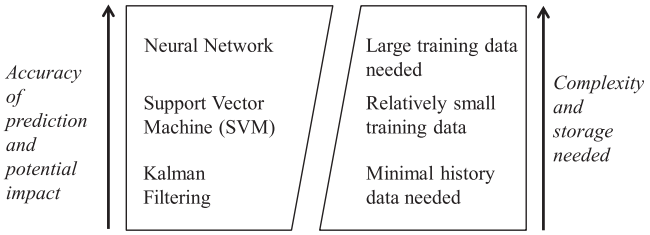

Fig. 6. Summary of the most popular prediction techniques that we identified in this paper. The more complex techniques are applicable at higher levels in the computing stack.

number of attributes predicted/classified by these techniques (seen as number of columns with example references in the corresponding rows in Table 1), 2) the number of different layers in the computing stack where these techniques were employed, and 3) the overall number of previous studies that seemed to prefer these techniques.

6 Conclusion

We presented a survey of some of the most popular prediction and classification for prediction techniques employed across multiple levels of the computing stack. These prediction techniques have been employed from predicting simple attributes of interest such as buffer utilization in networks-on-chip to predicting complex relationship affecting the power usage effectiveness in datacenters. Aside from discussing some of the most popular prediction techniques and emphasizing some of their advantages and disadvantages, our objective was to also identify trends in the way prediction techniques are used most recently. We see increasingly complex and sophisticated models such as deep neural networks being employed at higher levels, such as datacenters. User behavior and psychology is another direction that has been looked at recently in search for additional optimization opportunities that have not been explored before. If support vector machine models have been probably the best so far, it is likely in our opinion that deep neural networks to become the new norm at least at higher levels of abstraction, i.e., cloud computing type of applications and the supporting hardware infrastructure. It is hoped that this survey will provide useful initial guidance to the reader that may be interested in employing prediction/classification techniques in optimization solutions across layers in multicore processor systems.

Acknowledgments

This work was supported by the Dept. of Electrical and Computer Engineering at Marquette University. We thank Richard J. Povinelli for feedback on an earlier draft of this paper as well to the anonymous reviewers whose feedback helped to significantly improve the quality of this presentation. Finally, this survey paper is far from being comprehensive. However, we hope it helps to create a good enough picture of what has been, and especially what appears to be, the most promising prediction techniques. It should serve as a good starting point for the reader interested in employing some form of prediction to be used in optimization solutions across layers in multicore processor systems.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *Proc. 27th Int. Symp. Comput. Archit.*, 2000, pp. 83–94.

[2] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, timing modeling framework for multicore and many core architectures," in *Proc. 42nd Annu.ACM/IEEE Int. Symp. Microarchitecture*, 2009, pp. 469–480.

[3] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling method for CMOS VLSI systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.

[4] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. Int. Symp. High-Perform. Comput. Archit.*, 2003, pp. 91–102.

[5] C. Ababei and N. Mastronarde, "Benefits and costs of prediction based DVFS for NoCs at router level," in *Proc. 27th IEEE Int. System-on-Chip Conf.*, 2014, pp. 255–260.

[6] R. Ge, X. Feng, W.-C. Feng, and K. W. Cameron, "CPU MISER: A performance-directed, run-time system for power-aware clusters," in *Proc. Int. Conf. Parallel Process.*, 2007, pp. 18–18.

[7] R. Z. Ayoub and T. Simunic Rosing, "Predict and act: Dynamic thermal management for multi-core processors," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2009, pp. 99–104.

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, The Hardware/Software Interface*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[9] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2006, pp. 359–370.

[10] Autoregressivemoving-average model, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model.

[11] A. K. Coskun, T. S. Rosing, and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1503–1516, Oct. 2009.

[12] Y. Kim, F. Paterna, S. Tilak, and T. S. Rosing, "Smartphone analysis and optimization based on user activity recognition," in *Proc. Int. Conf. Comput. Aided Des.*, 2015, pp. 605–612.

[13] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 3, pp. 544–557, Mar. 2018.

[14] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, and Y. Pan, "Stochastic load balancing for virtual resource management in datacenters," *IEEE Trans. Cloud Comput.*, p. 1, Feb. 2016. doi: 10.1109/TCC.2016.2525984.

[15] G. Welch and G. Bishop, "An introduction to the Kalman filter," University of North Carolina at Chapel Hill, 2001. [Online]. Available: http://www.cs.unc.edu/\;tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf

[16] S.-Y. Bang, K. Bang, S. Yoon, and E.-Y. Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1334–1347, Aug. 2009.

[17] M. G. Moghaddam and C. Ababei, "Dynamic energy management for chip multiprocessors under performance constraints," *Elsevier Microprocessors Microsyst.*, vol. 54, pp. 1–13, 2017.

[18] S. Sarma and N. Dutt, "Minimal sparse observability of complex networks: application to MPSoC sensor placement and run-time thermal estimation & tracking," in *Proc. Conf. Des. Autom. Test Eur.*, 2014, Art. no. 329.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[20] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.

[21] M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of Machine Learning, MIT Press, 2012.

[22] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*. New York, NY, USA: Academic, 2015.

[23] Ethem Alpaydin, *Introduction to Machine Learning*. Cambridge, MA, USA: MIT Press, 2010.

[24] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. New York, NY, USA: Springer, 2013.

[25] Machine learning, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning

[26] Multinomial logistic regression, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Multinomial_logistic_regression

[27] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," *ACM/IEEE Des. Autom. Conf.*, 2008, pp. 734–739.

[28] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 144–157, Jan. 2013.

[29] E. Cai, D. C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning-based power/performance optimization for many-core systems with extended-range voltage/frequency scaling," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1318–1331, Aug. 2016.

[30] D.-C. Juan, S. Garg, and D. Marculescu, "Statistical peak temperature prediction and thermal yield improvement for 3D chip-multiprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 4, Aug. 2014, Art. no. 39.

[31] W. Lee, Y. Kim, J.H. Ryoo, D. Sunwoo, A. Gerstlauer, and L.K.John, "PowerTrain: A learning-based calibration of McPAT power models," *IEEE Int. Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 189–194.

[32] Z. Zhou, J.H. Abawajy, F. Li, Z. Hu, M.U. Chowdhury, A. Alelaiwi, and K. Li, "Fine-grained energy consumption model of servers based on task characteristics in cloud data center," *IEEE Access*, vol. 6, pp. 27080–27090, Aug. 2018. doi: 10.1109/ACCESS.2017.2732458.

[33] B. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, and Y. Xiang, "Context-aware energy enhancements for smart mobile devices," *IEEE Trans. Mobile Comput.*, vol. 13, no. 8, pp. 1720–1732, Aug. 2014.

[34] A. Das, A. Kumar, B. Veeravalli, R. A. Shafik, G. V. Merrett, and B. M. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2015, pp. 43–48.

[35] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and thread packing under power caps," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2011, pp. 175–185.

[36] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Identifying the optimal energy-efficient operating points of parallel workloads," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 608–615.

[37] k-nearest neighbors algorithm, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[38] Naive Bayes classifier, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[39] D. Poole and A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. Cambridge, U.K.: Cambridge Univ. Press, 2010. [Online]. Available: http://artint.info/index.html

[40] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 9, pp. 1395–1408, Sep. 2010.

[41] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms," in *Proc. 21st Int. Conf. High Perform. Comput.*, pp. 1–10, 2014.

[42] K. Dev and S. Reda, "Scheduling challenges and opportunities in integrated CPU+GPU processors," in *Proc. ACM/IEEE Symp. Embedded Syst. Real-Time Multimedia*, 2016, pp. 1–6.

[43] Z. Wu, X. Li, P. Garraghan, X. Jiang, K. Ye, and A. Y. Zomaya, "Virtual machine level temperature profiling and prediction in cloud datacenters," *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 735–736.

[44] X. Li, H. Cao, E. Chen, and J. Tian, "Learning to infer the status of heavy-duty sensors for energy-efficient context-sensing," *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 2, Feb. 2012, Art. no. 35.

[45] A. Vijayan, A. Koneru, S. Kiamehr, K. Chakrabarty, and M. B. Tahoori, "Fine-grained aging-induced delay prediction based on the monitoring of run-time stress," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD)*, vol. 37, no. 5, pp. 1064–1074, May 2018.

[46] W.-T. J. Chan, et al., "Learning-based prediction of embedded memory timing failures during initial floorplan design," *Asia South Pacific Des. Autom. Conf.*, 2016, pp. 178–185.

[47] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "BEOL stack-aware routability prediction from placement using data mining techniques," *IEEE Int. Conf. Comput. Des.*, 2016, pp. 41–48.

[48] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2013, pp. 354–357.

[49] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty, "Optimizing 3D NoC design for energy efficiency: A machine learning approach," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, 2015, pp. 705–712.

[50] K. Murphy, A brief introduction to reinforcement learning, 1998. [Online]. Available: http://www.cs.ubc.ca/\;murphyk/Bayes/pomdp.html

[51] R. Sutton and A. S. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA, USA: MIT Press, 1998.

[52] Reinforcement learning, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning

[53] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proc. 51st ACM/EDAC/IEEE Des. Autom. Conf.*, 2014, pp. 1–6.

[54] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, 2013, Art. no. 24.

[55] Y.-G. Chen, W.-Y. Wen, T. Wang, Y. Shi, and S.-C. Chang, "Q-Learning based dynamic voltage scaling for designs with graceful degradation," in *Proc. Symp. Int. Symp. Phys. Des.*, 2015, pp. 41–48.

[56] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2015, pp. 1521–1526.

[57] C.-L. Chou and R. Marculescu, "Designing heterogeneous embedded network-on-chip platforms with users in mind," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 9, pp. 1301–1314, Sep. 2010.

[58] S. Manoj P . D., H. Yu, H. Huang, and D. Xu, "A Q-learning based self-adaptive I/O communication for 2.5D integrated many-core microprocessor and memory," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1185–1196, Apr. 2016.

[59] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 2, pp. 1–25, Jan. 2016.

[60] Z. Wang, Z. Tian, J. Xu, R. Maeda, and H. Li, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," *ACM/IEEE Asia South Pacific Des. Autom. Conf.*, 2017, pp. 684–689.

[61] D. Biswas, V. Balagopal, R. Shafik, B. Al-Hashimi, and G. Merrett, "Machine learning for run-time energy optimization in many-core systems," in *Proc. ACM/IEEE Des. Autom. Test Eur. Conf. Exhib.*, 2017, pp. 1588–1592.

[62] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. VLSI Syst.*, vol. 24, no. 9, pp. 2488–2501, Sep. 2017.

[63] G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multitasking systems using online learning," in *Proc. Int. Symp. Low Power Electron. Design.*, 2007, pp. 207–212.

[64] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," in *Proc. 13th Int. Symp. Quality Electron. Des.*, 2012, pp. 747–754.

[65] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2014, pp. 1043–1055.

[66] A. K. Coskun, T. S. Rosing, and K. C. Grosss, "Temperature management in multiprocessor SOCs using online learning," in *Proc. 45th Annu. Des. Autom. Conf.*, 2008, pp. 890–893.

[67] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth, "Using and combining predictors that specialize," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 334–343.

[68] R. Jayaseelan and T. Mitra, "Dynamic thermal management via architectural adaptation," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2009, pp. 484–489.

[69] Y. Ge, Q. Qiu, and Q. Wu, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 10, pp. 1758–1771, Oct. 2012.

[70] A. Y. Yamamoto and C. Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors," *Microprocessors Microsyst.*, vol. 38, no. 1, pp. 53–63, Feb. 2014.

[71] J. Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in Artificial Neural Networks for CMP uncore power management," *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 308–319.

[72] G. Steven, R. Anguera, C. Egan, F. Steven, and L. Vintan, "Dynamic branch prediction using neural networks," in *Proc. Euromicro Symp. Digit. Syst. Des.*, 2001, pp. 178–185.

[73] E. Kakoulli, V. Soteriou, and T. Theocharides, "Intelligent hotspot prediction for network-on-chip-based multicore systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 3, pp. 418–431, Mar. 2012.

[74] K. Ma, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Dynamic machine learning based matching of nonvolatile processor microarchitecture to harvested energy profile," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, 2015, pp. 670–675.

[75] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao, "Neuro-NoC: Energy optimization in heterogeneous many-core NoC using neural networks in dark silicon era," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2018, pp. 1–5.

[76] B. Acun, E. K. Lee, Y. Park, and L. V. Kale, "Neural network-based task scheduling with preemptive fan control," in *Proc. 4th Int. Workshop Energy Efficient Supercomputing*, 2016, pp. 77–84.

[77] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 564–576.

[78] Deep learning, Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning

[79] Y. LeCun, "Learning invariant feature hierarchies," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 496–505.

[80] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554, Jul. 2006.

[81] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Sci.*, vol. 313. no. 5786, pp. 504–507, Jul. 2006.

[82] Li Deng and Dong Yu, "Deep learning: Methods and applications," *NOW Found. Trends Signal Process.*, vol. 7, no. 3/4, Jun. 2014.

[83] J. Markoff, "Scientists see promise in deep-learning programs," *New York Times*, (2012, Nov.). [Online]. Available: https://www.nytimes.com/2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html

[84] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Netw.*, vol. 32, pp. 333–338, 2012.

[85] Y. LeCun, Y. Bengio, and G.E. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[86] J. Gao, "Machine learning applications for data center optimization," *Google White Paper*, 2014. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf

[87] L.K. John, "Machine learning for performance and power modeling/prediction," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2017, 1–2.

[88] X. Zheng, P. Ravikumar, L. K. John, and A. Gerstlauer, "Learning-based analytical cross-platform performance prediction," in *Proc. IEEE Int. Conf. Embedded Comput. Syst.: Archit. Model. Simul.*, 2015, pp. 52–59.

[89] D. Lee, T. Kim, K. Han, Y. Hoskote, L. K. John, and A. Gerstlauer, "Learning-based power modeling of system-level black-box IPs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2015, pp. 847–853.

[90] M. S. Hossain, S. Rahaman, A.-L. Kor, K. Andersson, and C. Pattinson, "A belief rule based expert system for datacenter PUE prediction under uncertainty," *IEEE Trans. Sustainable Comput.*, vol. 2, no. 2, pp. 140–153, Apr.-Jun. 2017.

[91] S. Van Winkle, A. K. Kodi, R. C. Bunescu, and A. Louri, "Extending the power-efficiency and performance of photonic interconnects for heterogeneous multicores with machine learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 480–491.

[92] T. Mantoro, A. Olowolayemo, and S.O. Olatunji, "Mobile user location determination using extreme learning machine," in *Proc. 3rd Int. Conf. Inf. Commun. Technol. Moslem World*, 2011, pp. D25–D30.

[93] T. Anagnostopoulos, C. Anagnostopoulos, S. Hadjiefthymiades, M. Kyriakakos, and A. Kalousis, "Predicting the location of mobile users: A machine learning approach," in *Proc. Int. Conf. Pervasive Serv.*, 2009, pp. 65–72.

[94] B. Li, L. Duan, and L. Peng, "Efficient microarchitectural vulnerabilities prediction using boosted regression trees and patient rule inductions," *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 593–607, May 2010.

[95] H.-K. Peng and R. Marculescu, "Multi-scale compositionality: identifying the compositional structures of social dynamics using deep learning," *PLOS*, Apr. 2014. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0118309

[96] Jan Maciejowski, *Predictive Control with Constraints*, 1st ed., Englewood Cliffs, NJ, USA: Prentice Hall, 2000.

[97] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *Proc. Eur. Conf. Circuit Theory Des.*, 2009, pp. 711–714.

[98] X. Wang, K. Ma, and Y. Wang, "Adaptive power control with online model estimation for chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1681–1696, Oct. 2011.

[99] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 170–183, Jan. 2013.

[100] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and user experience-aware dynamic reliability management in multicore processors," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 2.

[101] X. Zhao, T. Peng, X. Qin, Q. Hu, L. Ding, and Z. Fang, "Feedback control scheduling in energy-efficient and thermal-aware data centers," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 46, no. 1, pp. 48–60, Jan. 2016.

[102] R. Hesse and N. D. E. Jerger, "Improving DVFS in NoCs with coherence prediction," in *Proc. 9th Int. Symp. Netw.-on-Chip*, 2015, Art. no. 24.

[103] E. K. Lee, H. Viswanathan, and D. Pompili, "Proactive thermal-aware resource management in virtualized HPC cloud datacenters," *IEEE Trans. Cloud Comput.*, vol. 5, no. 2, pp. 34–248, Apr.-Jun. 2017.

[104] S. Sharifi, D. Krishnaswamy, and T. S. Rosing, "PROMETHEUS: A proactive method for thermal management of heterogeneous MPSoCs," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1110–1123, Jun. 2013.

[105] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *NOW Found. Trends Inf. Retrieval*, vol. 2, no. 1/2, pp. 1–135, 2008.

[106] A. Mallik, B. Lin, G. Memik, P. A. Dinda, and R. P. Dick, "User-driven frequency scaling," *Comput. Archit. Lett.*, vol. 5, no. 2, pp. 16–16, 2006.

[107] B. Lin, A. Mallik, P. A. Dinda, G. Memik, and R. P. Dick, "User- and process-driven dynamic voltage and frequency scaling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2009, pp. 11–22.

[108] C.-L. Chou and R. Marculescu, "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 1, pp. 78–91, 2010.

[109] S. Pasricha, B. K. Donohoo, and C. Ohlsen, "A middleware framework for application-aware and user-specific energy optimization in smart mobile devices," *Pervasive Mobile Comput.*, vol. 20, pp. 47–63, Jul. 2015.

[110] A. Shye, Y. Pan, B. Scholbrock, J. S. Miller, G. Memik, P. Dinda, and R. Dick, "Power to the people: Leveraging human physiological traits to control microprocessor frequency," *ACM/IEEE Int. Symp. Microarchitecture*, 2008, pp. 188–199.

[111] K. Yan, X. Zhang, and X. Fu, "Characterizing, modeling, and improving the QoE of mobile devices with low battery level," in *Proc. ACM/IEEE Int. Symp. Microarchitecture*, 2015, pp. 713–724.

Cristinel Ababei (SM'14) received the PhD degree in electrical and computer engineering from the University of Minnesota, Minneapolis, in 2004. He is an assistant professor with the Department of ECE, Marquette University Prior to that, from 2012 to 2013, he was an assistant professor with the Department of EE, SUNY at Buffalo. Between 2008 to 2012, he was an assistant professor with the Department of ECE, North Dakota State University. From 2004 to 2008, he worked for Magma Design Automation, Silicon Valley. His current research interests include electronic design automation of systems-on-chip with emphasis on reliability and energy consumption, datacenters, parallel computing, and FPGAs. He is a senior member of the IEEE.

Milad Ghorbani Moghaddam (S'16) received the BS degree from Ferdowsi University of Mashhad, Iran, in 2008 and the MSc degree from Isfahan University of Technology, Iran, in 2011, both in computer engineering. Currently, he is working toward the PhD degree in the Department of Electrical and Computer Engineering, Marquette University, Milwaukee, Wisconsin. His main research interests include lifetime reliability of chip multiprocessors and full system simulators. He is a student member of the IEEE.