Runtime Application Behavior Prediction Using a Statistical Metric Model

Ruhi Sarikaya, *Senior Member*, *IEEE*, Canturk Isci, *Member*, *IEEE*, and Alper Buyuktosunoglu, *Senior Member*, *IEEE*

Abstract—Adaptive computing systems rely on accurate predictions of application behavior to understand and respond to the dynamically varying characteristics. In this study, we present a Statistical Metric Model (SMM) that is system- and metric-independent for predicting application behavior. SMM is a probability distribution over application patterns of varying length and it models how likely a specific behavior occurs. Maximum Likelihood Estimation (MLE) criterion is used to estimate the parameters of SMM. The parameters are further refined with a smoothing method to improve prediction robustness. We also propose an extension to SMM (i.e., SMM-Interp) to handle sudden short-term changes in application behavior. SMM learns the application patterns during runtime, and at the same time predicts the upcoming application phases based on what it has learned up to that point. We demonstrate several key features of SMM: 1) adaptation, 2) variable length sequence modeling, and 3) long-term memory. An extensive and rigorous series of prediction experiments show the superior performance of the SMM predictor over existing predictors on a wide range of benchmarks. For some of the benchmarks, SMM reduces the prediction error rate by $10X$ and $3X$, compared to last value and table-based prediction approaches, respectively. SMM's improved prediction accuracy results in superior power-performance tradeoffs when it is applied to an adaptive dynamic power management scheme.

Index Terms—Workload behavior prediction, statistical modeling, adaptive computing

1 Introduction

Current generation microprocessors deploy adaptive management techniques to cope with the inherent variability within an application as well as across different applications. Typically, these techniques use changing phase behavior of an application at certain time epochs to dynamically tradeoff metrics of interest and performance. In most cases, the techniques are reactive in that they are enabled once an application phase transition occurs. In cases where the application phase behavior is very dynamic, reactive systems can result in further performance degradation and can also lead to instability in the system.

A more interesting alternative is the proactive management of the system, which requires accurate prediction of the application behavior by using past behavior patterns. Previously, such proactive methods have been proposed, where pattern history tables [16], program flow behavior [27], [24], and statistics over recent performance characteristics [10] are used for prediction. These techniques predict application behavior by tracking recently observed patterns or statistics in the observed application features, which can be used to guide dynamic management decisions. While these predictors can be effective in some scenarios, they do not address two critical issues; 1) predicting global long range patterns, and 2) modeling patterns of variable length. Fig. 1 shows an example of an actual execution trace from the `applu` benchmark. Here, the two boxed regions show an exemplary repetitive execution. The table-based predictor used in this example has a pattern length of 8 samples. The two captured repetitive regions show an 8-sample pattern, where the second occurrence experiences a single fluctuation. While the table-based predictor learns the behavior from the first pattern, this small fluctuation actually leads to a pattern mismatch. As a result, the table-based predictor backs off to last value predictor, and the following two consecutive samples labeled as i and j are mispredicted. However, an advanced predictor that is resilient to pattern fluctuations, can actually discern this repetition and may correctly predict both i and j from prior observations. This pattern is discovered by the proposed predictor by modeling variable size patterns and is highlighted with the smaller enclosed regions (i.e., S-shaped areas in green in Fig. 1).

In this paper, we propose a Statistical Metric Model (SMM) for metric prediction. SMM estimates the probability of the next phase based on prior observations and also alleviates the shortcomings of the prior predictors with its ability to model patterns of different length and long-term global patterns. A comprehensive set of experiments demonstrates the effectiveness of the SMM predictor in comparison to the previously proposed predictors. SMM predictor, when applied to dynamic power management,

- *R. Sarikaya is with Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. E-mail: ruhi.sarikaya@microsoft.com.*
- *A. Buyuktosunoglu is with the IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Rte 134, Yorktown Heights, NY 10598. E-mail: alperb@us.ibm.com.*
- *C. Isci is with the IBM TJ Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532. E-mail: canturk@us.ibm.com.*

Manuscript received 16 May 2011; revised 13 Oct. 2011; accepted 17 Dec. 2011; published online 16 Jan. 2012.
Recommended for acceptance by J. Xue.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-05-0326.
Digital Object Identifier no. 10.1109/TC.2012.25.

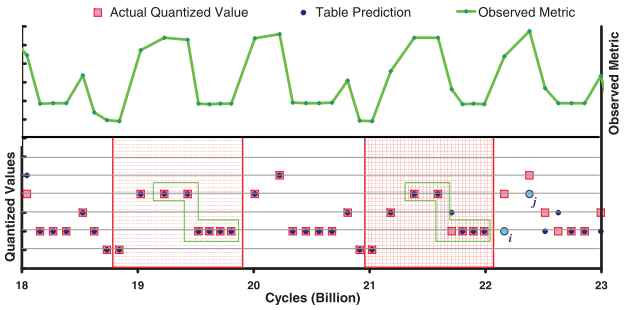

Fig. 1. Vulnerability of a table-based predictor to fluctuations in observed metrics.

results in lower prediction error rate[1] and thus better power-performance tradeoffs compared to the existing predictors. Our contributions can be summarized below. Note that this work expands our recent work [22] and [23] in several ways which are highlighted in the last three bullets:

- We propose a new data-driven SMM, which is inspired from natural language modeling field.
- We show that SMM is an adaptive model, where each time it observes a new sample it updates its parameters to incorporate the changes in metric behavior. The more it learns by observing new samples the better gets its prediction performance.
- We show that unlike table based and last value predictors, SMM has the ability to model long-term patterns in the data.
- We demonstrate that when SMM is applied to dynamic power management, it results in better power-performance tradeoffs compared to the existing predictors.
- We propose SMM-Interp, which is an extension of SMM to emphasize local short-term patterns in modeling, while keeping long-term modeling and prediction capacity intact.
- We present a detailed analysis and requirements for the online model training and prediction. We show that SMM can be implemented within the operating system software in context switch time granularities with no visible performance impact.
- We compare SMM predictor to widely used last value and table-based predictors, and show its superior performance.

The rest of the paper is organized as follows: Section 2 gives an overview of the prior work on program phase prediction. Section 3 describes the predictors that serve as the baseline in our work. Section 4 provides the foundation and formulation of SMM predictor. Section 5 gives a detailed description of the computational overhead for SMM. Section 6 describes SMM-Interp, which is an extension of SMM. Section 7 presents methodology. Section 8 demonstrates the experimental evaluation of SMM and its application to dynamic power management, and Section 9 offers our conclusions.

1. Prediction error rate (e) is defined as the average absolute difference between the predicted (p) and true (t) metric value divided by the total metric sample size (N), $e = \frac{1}{N}\sum_{i=1}^{N}|p_i - t_i|$. Normalized prediction error (e_n) is computed by dividing the absolute prediction error in the predicted phases of the tracked workload metrics to the maximum absolute range (R) of the predicted metrics. $e_n = \frac{1}{N}\sum_{i=1}^{N}\frac{|p_i - t_i|}{R}$. Such normalization helps to understand prediction accuracy for different architectural metrics that might otherwise have orders of magnitude differences in their absolute quantities, such as, for example, IPC versus last-level cache miss rates.

2 Related Work

Tracking, characterizing, and predicting application characteristics have been the focus of a large body of research. These studies leverage various characterization metrics including performance monitoring counters, programmatical flow, and system statistics. One line of research explores characterization of observed behavior via runtime statistic collection and architectural or system-level simulations [14], [15], [26], [1], [3], [9], [12]. These techniques mainly focus on interpreting specific application execution behavior and detecting some indicative characteristics. Other work [2], [4], [8], [5], [30] uses system statistics to guide dynamic adaptations such as power and thermal management. While these techniques provide significant insight to application behavior and its impact on dynamic management decisions, they do not explore online prediction of future behavior. The resulting dynamic management techniques can be considered as reactive approaches. Our proposed approach aims to provide the necessary means for proactive adaptations.

A significant number of study also targets at predicting future application behavior [10], [16], [13], [27], [19], [20], [21], [29], [32]. For example, Duesterwald et al. [10] describe different simple statistical and table-based predictors for within- and across-metric predictions of performance monitoring information. However, the statistical predictors they considered were based on simple averages of the recently observed metric values. For example, an Average(N) predictor chooses the average over the last N values and aMode(N) predictor chooses the most frequently occurring value among the last N values. An exponentially weighted moving average (EWMA) predictor places more emphasis on the most recent data. None of these predictors were using probabilistic approaches. Prior studies [10], [13] also looked at the comparative performance of different prediction methods, commonly demonstrating that the table-based predictors tend to perform significantly better than purely simple statistical approaches for projecting dynamically varying workload characteristics. SMM, in comparison, is a combination of the statistical models and the table-based, pattern-tracking methods. SMM's grammar models perform in function similar to a range of table-based predictors with different model orders, and the pattern statistics help to choose the right model order and the corresponding table for next phase prediction. Vandeputte et al. [29] compare last value, burst, and run-length encoder predictors with additional improvements via confidence counters and conditional update configurations. They demonstrate that 2-level burst predictor with confidence and conditional update is the best performing configuration among the evaluated predictors. Sarikaya and Buyuktosunoglu [20], [21] describe an optimal prediction technique based on a predictive least squares minimization. Isci et al. [13] develop a table-based runtime predictor to predict future behavior from past pattern characteristics. Zhou et al. [32] monitor memory access patterns and estimate memory behavior of workloads for energy efficient memory allocation. Sherwood et al. [27]

describe microarchitectural phase predictors based on repetitive program flow behavior. Lau et al. [19] further refine these phase classifications by defining transition phases to mitigate the effects of small fluctuations and in turn to improve prediction accuracy. They classify the application behavior during a transition period between two stable phases into a "transition phase" rather than attributing it to a particular actual phase. This transition-based approach also helps to improve prediction accuracy by addressing an orthogonal problem, i.e., pruning the outliers due to transitions or bursts and thus, arriving at a cleaner and reliable phase characteristic information. SMM, in comparison, uses a language modeling-based approach to track repetitive application behavior with variable pattern lengths, which can be comprised of many phases. Shen et al. [24], [25] detect such repetitions from reuse distance patterns for dynamic memory configurations via profiling and instrumentation. This work leverages grammar compression to construct phase hierarchies and to identify variable-length phases, and uses binary rewriting to insert markers into the studied programs. In summary, all these studies provide useful prediction techniques suitable for different applications. This paper, on the other hand, shows the benefit of statistical metric modeling for tracking variable pattern history lengths and modeling long term patterns, which have not been addressed by previous studies.

3 Baseline Predictors

In order to demonstrate the comparative advantages of the SMM predictor, we first provide a brief description of some of the existing prediction approaches employed in the literature. These predictors serve as the baseline for our work.

3.1 Last Value Predictor

Last value prediction is perhaps the simplest metric prediction method. Last value predictor assumes that the most recently observed behavior is representative of the future application characteristics. Such a prediction approach tends to be very effective for slow-moving application characteristics, in essence, predicting a time-shifted version of the original trace. However, this predictor has limited accuracy for rapidly varying application characteristics.

The most appealing feature of the last value predictor is its implementation simplicity. The last value predictor needs to monitor and store only the most recent application sample. The prediction algorithm simply sets the prediction output to the most recent observation. Some variations on the last value approach involve moving averages and exponential weighting of the observed history. Compared to this predictor, other elaborate prediction schemes generally introduce additional complexity and overhead in terms of storage and computation to carry out the prediction largely without a consistent performance gain.

3.2 Table-Based Predictor

An alternative to last value type predictors is the table-based predictor. While last value and its variants look at the statistics of last or recent application behavior, table-based prediction tries to understand and leverage the patterns in execution flow to predict future application behavior. The key motivation for such an approach is the inherent repetitive application characteristics. An example of these

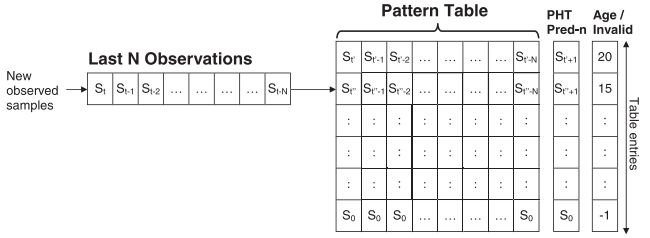

Fig. 2. Table-based predictor structure.

predictors, namely a global phase history table predictor [13] is depicted in Fig. 2. Such a predictor can be implemented either in software or hardware depending on the desired prediction quanta and the time granularity of the employed adaptation policy [13], [10], [27].

The global register is the heart of the table-based predictor and it tracks the most recently observed (e.g., N) sample characteristics. At each sampling period, this register records the last observed sample. The contents of this register are used to index into a pattern table, which holds a certain number of previously encountered patterns. The predictions are also deterministically encoded into this table and are performed for each previously observed pattern. An age entry is used to track the reuse time of different entries for a least recently used (LRU) replacement policy. After a prediction is performed, the register contents are added to the table by either replacing the oldest entry or by inserting into an available invalid entry.

4 Statistical Metric Model

4.1 A Natural Language Modeling-Based Perspective for SMM

We see a resemblance between how a metric sequence and a natural language are generated. In a natural language, we generate words sequentially to construct sentences. The SMM treats the metric samples as the words in a language and builds a language model for each benchmark. Note that as metric samples are real numbers one has to quantize them into a set of discrete values (like vocabulary in a language), which are called "quantization bins." In a natural language, words do not follow each other randomly because of the underlying grammar. There is an underlying structure defined by the grammar, which determines the order in which we bring words together to make meaningful sentences. Our intuition is that, like in natural languages, we can treat the metric modeling as a language modeling problem. We assume that there is an underlying structure in each benchmark, and if indeed there is such an underlying structure (e.g., repetitive patterns) SMM can reveal and model this structure. However, if there is not any structure, that is to say, the benchmark is a completely random sequence of numbers then SMM will not do a worse job than any other predictor, as long as it is trained on sufficiently large data. In the rest of the manuscript, we will often draw parallels between natural language modeling and SMM to explain the concepts used in this work. Next, we introduce the foundations of SMM, which will pave the way for a mathematical formulation.

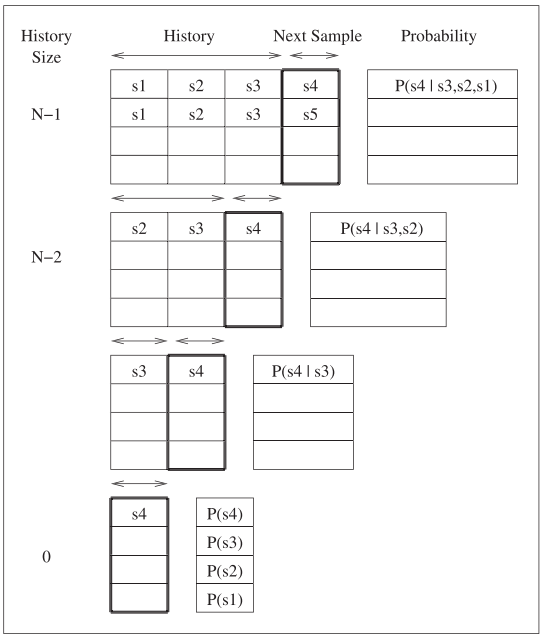

Fig. 3. Description of the statistical metric model with back off for $n = 4$.

4.2 Foundations of SMM

SMM models the conditional distribution on the identity of the ith (quantized) sample given the identities of all previous (quantized) samples in a metric sequence. Therefore, the next sample depends on all the previous samples. In general, this is a true statement but such a model will suffer from parameter estimation problems. Therefore, we have to make a computationally convenient approximation that a sample depends only on the previous $n-1$ samples, where n depends on the amount of available data to estimate the model parameters. Recalling the natural language modeling analogy, what word we will speak next typically depends more on the most recently spoken words than the words we have spoken a while ago. With this approximation, SMM becomes an n-gram model [17], which is a special class of Markov models. Here, n refers to the maximum length of the metric patterns (e.g., patterns of $n = 4$ samples as given in Fig. 3). N-gram models have been studied intensively since their introduction in early eighties and have been widely used in speech and natural language processing [7]. The parameters of n-gram models are estimated from a large training corpus. The models produce a reasonable nonzero probability for every word in the vocabulary. The probability of the nth sample is conditioned on the previous $n-1$ samples. Unlike natural language models, which are built offline only once and then used for the application without any update, the SMM is built and updated on runtime as many times as the observed metric samples. As such, it requires online model training.

SMM has two sets of entries: the finite sequences (i.e., patterns) of length n and the associated probabilities with each sequence. The SMM model of order $n = 4$ is shown in Fig. 3. The model contains sequences of length 1 to n where the last sample in each sequence is the output given the remaining $n-1$ history samples. For example, the first entry has the (s_1, s_2, s_3) as the history for the next sample s_4 with the probability $P(s_4|s_3, s_2, s_1)$. The probabilities are called the model parameters that are estimated from the observed data. As we highlighted before, the SMM consists of models of lower order m $(1 \leq m \leq n)$, allowing variable length sequence modeling and increasing the likelihood of finding a matching subsequence for a given finite sequence.

The prediction power of SMM increases as n increases. However, this comes at the expense of poor parameter estimation, which could hurt the performance. Following extensive experimentation, $2 \leq n \leq 6$ are found to work best for natural language models. In this work, for program behavior prediction we use $n = 8$ to have a fair comparison with prior work [13]. However, we also provide an evaluation of SMM performance with smaller n. By restricting the conditioning information to the previous $n-1$ samples, we are making a simplifying assumption. Although samples further back in history potentially also have an influence on the identity of the next sample, higher order models provide diminishing returns due to the fact that the number of parameters in the SMM model is exponential in n.

4.3 Formulation of SMM

Mathematically, SMM is a probability distribution, $P(s)$, over L samples $S = s_1, s_2, \ldots, s_L$, that attempts to reflect the frequency with which each finite sequence $s = s_1, s_2, \ldots, s_l$ $(l < L)$ occurs in a metric. $P(s)$ is expressed as the product of the probabilities of the samples that compose the sequence, with each sample probability is conditional on the identity of the last $n-1$ samples.

$$\begin{aligned} P(s) &= P(s_1)P(s_2|s_1)\cdots P(s_l|s_{l-1}\ldots s_1) \\ &= \prod_{i=1}^{l} P(s_i|s_{i-1}, s_{i-2}, \ldots., s_1), \end{aligned} \tag{1}$$

where $P(s_2|s_1)$ is the conditional probability of observing sample s_2, given the previous sample was s_1. We can approximate the probability $P(s)$ by limiting the dependence on all the previous words to only most recently observed ones $(n-1)$:

$$P(s) = \prod_{i=1}^{l} P\left(s_i|s_1^{i-1}\right) \approx \prod_{i=1}^{l} P\left(s_i|s_{i-n+1}^{i-1}\right), \tag{2}$$

where s_i^j denotes samples $s_i, \ldots, s_j$. For the sake of simplicity, bigram $(n = 2)$ case is considered below. The extension of formulation to higher order models are trivial. By setting $n = 2$, we make the approximation that the probability of a sample only depends on the identity of the immediately preceding sample, hence we can approximate $P(s)$ as:

$$P(s) = \prod_{i=1}^{l} P(s_i|s_{i-1}). \tag{3}$$

The individual conditional probability distributions can be estimated with Maximum Likelihood Estimation (MLE) technique by using relative frequencies:

$$P_{ML}(s_i|s_{i-1}) = \frac{C(s_{i-1}, s_i)}{C(s_{i-1})}, \tag{4}$$

where $C(x)$ denotes the number of times the sequence x occurs in the metric. The conditional distributions are multiplied to estimate $P(s)$.

4.4 Improving Probability Estimates via Model Smoothing

In the absence of sufficiently large observations, the maximum likelihood (ML)-based relative frequency estimates do not result in reliable probability estimates. For example, for a model with $n = 8$ and a vocabulary (i.e., quantization bins) size of 20, we can potentially have $20^8 = 25.6$ billion unique sequences of length 8. However, in practice, all applications combined exhibit less than 10K unique patterns, which is negligible compared to 25.6 billion (which corresponds to about 300 days of continuous running of an application with no two sequence (i.e., pattern) of length 8 are the same). However, in our experiments we sample the data at every 10 msec, and the conventional probability estimate for each unseen sequence would be zero. However, just because an event has never been observed so far does not mean that it cannot occur in the future. To address these issues with the MLE probability estimates, we apply "model smoothing" to the relative frequencies to make sure that each probability estimate is larger than zero. Various smoothing techniques can be devised to ensure that the probability estimates are greater than zero for samples which do not occur in the training data. One simple smoothing technique is to pretend each bigram occurs one more than it actually did, leading to;

$$P_{+1}(s_i|s_{i-1}) = \frac{C(s_{i-1}, s_i) + 1}{C(s_{i-1}) + |V|}, \tag{5}$$

where $V = \{1, 2, \ldots, 19, 20\}$ is the set of all unique quantization bins being considered. This has the desirable quality of preventing zero bigram probabilities. However, this scheme has the flaw of assigning the same probability to say, "(10,10)" and "(20,20)" (assuming neither occurred in the metric data so far and the overall metric mean is 10), even though intuitively the former seems more likely because the sample 10 is much more frequently observed than 20.

Another smoothing method is to interpolate higher order n-gram models with lower order n-gram models, because when there is insufficient data to estimate a probability in the higher order model, the lower order model can often provide useful information.

$$P_{int}(s_i|s_{i-1}) = \lambda P(s_i|s_{i-1}) + (1 - \lambda)P(s_i), \tag{6}$$

where $P_{int}(s_i|s_{i-1})$ is the probability distribution for the interpolated model and $P(s_i) = C(s_i)/L$ is the unigram model. The interpolated model achieves the behavior so that bigrams which involve common words are assigned higher probabilities [17], [6]. The weighting of the linear interpolation is estimated by maximizing the probability of "held-out" data which differs from the data used to original estimate the n-gram frequencies. This type of smoothing was shown to obtain competitive performance when the training data are small, as is the case with the metrics considered here.

Among all the smoothing methods *absolute discounting* appears to be most popular in the natural language processing area. The higher order distribution is created by subtracting a fixed discount $D < 1$ from each nonzero count. We use an interpolated version of the absolute discounting [7], which was shown to provide the best performance in natural language modeling applications. For finite metric sequences with nonzero counts, this distribution has the following general form:

$$P_{int}\left(s_i|s_{i-n+1}^{i-1}\right) = \frac{C\left(s_{i-n+1}^{i}\right) - D}{\sum_{s_i} C\left(s_{i-n+1}^{i}\right)} + \alpha\left(s_{i-n+1}^{i-1}\right) P_{int}\left(s_i|s_{i-n+2}^{i-1}\right), \tag{7}$$

where $P_{int}(s_i|s_{i-n+2}^{i-1})$ is the lower order smoothing distribution. Normalization constraints fix the value of $\alpha(s_{i-n+1}^{i-1})$:

$$\alpha(s_{i-n+1}^{i-1}) = D\frac{n_{1+}(*, s_{i-n+1}^{i-1})}{\sum_{s_i} C(s_{i-n+1}^{i})}, \tag{8}$$

where $n_{1+}(*, s_{i-n+1}^{i-1})$ represents the number of bins for which $C(s_{i-n+1}^{i}) > 0$.

4.5 Advantages of SMM

SMM has three key features which set it apart from the previous work; 1) it is adaptive, 2) it has the ability to model sequences (i.e., patterns) of variable length, 3) it has long term memory.

The first feature allows runtime learning and provides ability to model changing application characteristics. The second feature provides robustness for metric prediction and contributes to accuracy. We show in Fig. 3 that SMM has the ability to match patterns of variable length m, where $(1 \leq m \leq n)$. This is a major issue with table-based predictor, as its patterns are fixed in length. SMM takes a pattern of length n and checks whether there is such an entry in the model, if so, it uses the probability for that entry. If not, then it checks models of lower order for the matching subsequences by iteratively limiting the history size. For a given history, the predicted next sample is selected based on its probability.

The third feature provides ability to model patterns seen long time ago. Last value predictor, table-based predictor and others attempt to use short term dependencies, focusing on the most recent samples to predict the next sample. This of course is useful but not sufficient to model long term dependencies where repetitive patterns go beyond the window of past n samples. The benefit of these features are demonstrated by a set of experiments in the experimental section of the paper.

5 Computational Complexity

In workload behavior prediction task, the entire sequence of execution is not observed when the SMM is used for prediction. In fact, in practice monitored system execution is a continuous process that produces a stream of workload characteristics without any particular beginning or end. The metric data that are of interest to the SMM predictor keeps coming from the beginning of a system startup to the next system halt, which can span days or months. Because of this, the SMM must be trained/updated constantly. The key question here is whether the computational complexity of model training/updating presents a challenge on the practicality of the SMM.

There are three aspects to the SMM's overhead: 1) model training, 2) prediction of the likely next sample using the

model, 3) model storage. We previously touched these issues [23] without going into detail, but here we will elaborate on them.

In the proposed approach, all of the three items (model training, prediction ,and model storage) corresponding to the SMM overhead are done online for each observed sample during an execution of an application (e.g., `bzip`, `applu`, etc.). SMM is essentially a set of tables of different length, where each entry in the table has three components: 1) history, 2) next sample (to be predicted), and 3) probability of predicting the next sample given the history. Therefore, predicting the next sample using SMM is the most straightforward among the three overheads. For a given history we simply look up the SMM tables and pick the sample that has the highest probability for the prediction.

5.1 SMM Training

The computational overhead for SMM training can be estimated given the following three parameters: 1) number of quantization bins (i.e., vocabulary V), 2) length of finite sequence (n), 3) how often we need to update the model. If SMM parameters updated at every sample, computation is bounded by $O(kVn)$ division and multiplication operation, where k is a small constant ($k \leq 3$). In our experiments, we have $V = 20$ quantization bins, $n = 8$ and we update the SMM parameters at every sample. For such a setting, the computational overhead for each model update would be less than 1K multiply and divide operations. Thus, the compute time overheads is on the order of microseconds and it can be implemented within the operating system software in context switch time granularities with no visible performance impact. We refer the reader to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2012.25, for further explanation of the model training overhead.

5.2 SMM Storage

SMM size (i.e., model storage) is more of a theoretical issue than the practical one. For a quantization bin size of 20 (i.e., $V = 20$) and model order of $\mathrm{n} = 8$, the sequence space can have as many as 20^8 distinct sequences/patterns. In natural language modeling the model size could be a real concern, as, for example, in English the vocabulary size could be as large as 1M. A simple trigram language ($n = 3$) model in theory could be as many as ($1M^3$) patterns. However, trigram and even higher order models are built and stored in a small amount of disk space. This is because of the fact that in practice only a tiny fraction of this possible sequence space is observed, due to the underlying structure in the data. When we are forming sentences, what we have said in the past (i.e., history) greatly limits what we will say next from 1M alternatives to possibly as few as a hundred words. The same is largely true for SMM applied to the metric modeling.

Obviously, the number of quantization bins (V) and the model order (n) are two parameters that can be adjusted to control the model size. In the paper, as we will show in Fig. 11 that we can decrease n from 8 to 6 (going from a space of $20^8 \rightarrow 20^6$ distinct sequences) without sacrificing the performance. Likewise, we can reduce the quantization bins from say 20 to 8 (going from a space of $20^6 \rightarrow 8^6 = 262K$ sequences) without significantly hurting the performance. Moreover, there are very effective model pruning techniques [11] that keep the model size under control without sacrificing model performance. The idea underlying the pruning methods is to remove higher order sequences from SMM (i.e., sequence tables) whose probabilities fall below a threshold and instead rely on lower order sequences to predict the next word. In our experiments with the SPEC CPU2000 data set on average the SMM model storage for model order 8 is around 10 KB, which is in the same ballpark as the table-based predictor.[2] While the largest storage requirement of 21 KB was observed for `mgrid`, there are 15 benchmarks with storage requirements of less than 5 KB.

6 Integrating Local and Global Metric Behavior

Despite all the advantages SMM does not account for the temporal proximity of the patterns during the parameter estimation and prediction. Next, we will present the case for the need for "local" metric modeling, and later combine local and global metric modeling within a new model called SMM-interp (i.e., interpolated SMM). SMM-interp has the ability to emphasize recently observed patterns while performing global metric modeling and prediction.

6.1 Argument for the Local Metric Modeling

The main idea underlying the SMM predictor concerns an important limitation of all the Markov models. The main limitation of the Markov models is their inability to reflect short-term or recently observed local patterns. Again going back to the natural language analogy [18], suppose that we are reading a document and we just read the word sequence "*the new ...*," and that the word *phone* followed these two words 5 percent of the time in the text we read so far, while the word "window" followed them 1 percent of the time. SMM will assign *phone* a probability of 0.05 and *window* a probability of 0.01. For an isolated sentence, this would be the reasonable choice to make. But now suppose that we just read the word *window* in the previous sentence (i.e., topic effect) and there was no mention of the word *phone*. We believe that a human would then assign overwhelmingly higher probability to the word *window*. A word used in the immediate past (say the last 100-200 words or so) is much more likely to be used soon than either its overall frequency in the English language or any of the popular Markov models would predict. The same argument obviously applies to computer systems, as several architectural components are actually targeted toward exploiting such temporal locality principles such as caches and branch history tables. Furthermore, as the common loop structures and the control flow of applications often exhibit such temporal repetitive execution, the recent behavior is commonly a stronger indicator of future application execution. Several prior studies on dynamic optimization, workload characterization, hot-code identification, and control-flow analysis illustrate and exploit these temporal characteristics and demonstrate the value of recency-awareness in workload tracking and dynamic adaptations.

2. Each bin can be represented with a byte and each entry has a model order of 8. For a 1,024 entry table the total storage is $1{,}024 \cdot 8 = 8$ KB.

Therefore, modeling and integrating short-term shifts in metric sequence frequencies to global metric modeling and prediction might perform better than the pure Markov model embodied in the SMM predictor. Following this analogy, we can design a predictor that has both a cache and a Markov component. In fact, there are several ways of implementing this. One way is to build a second—what we call "cache" SMM predictor using only the most recently observed say 100-500 samples. The cache component of our combined model would estimate the probability of a metric sequence from its recent frequency of use. The combined model would use a weighted average of the probabilities from the global and cache component of the SMM predictors in calculating the most likely expected sample probability. Another way is to use table based and last value predictors as "cache" component and combine their prediction with the SMM prediction to improve prediction accuracy. In this work, we implemented the latter.

6.2 Combining Local and Global Prediction

The SMM as we described so far does global metric modeling and prediction, since it treats all observed patterns equally whether they are observed very recently or a while ago. However, we believe that the recently observed pattern should be treated favorably and should play a more important role than patterns observed a while ago. We therefore combine the local short-term prediction and global prediction to improve overall prediction accuracy. This combination leads to a new form of SMM, named SMM-Interp, which interpolates global and local short-term predictions. As pointed out in the previous section we employ table based and last value predictors as the short-term and very short-term predictors, respectively.

There are several ways to integrate short-term and global prediction results. We choose "performance-based voting," where we consider last value, table based and SMM predictors as three different predictors in a pool, and track their past performance to decide on the final prediction output. By including two short-term predictors in the pool and treating all the predictors equally we are biasing the final prediction results in favor of the recently observed patterns. The decision criterion is simple but effective; pick the output of the predictor as the final prediction output if it provided the smallest error in predicting the most recent sample. However, it is not always clear which predictor achieved the lowest prediction error, as they may all have predicted the most recent sample accurately, or in general, their errors are equal. In this case, we take into account their overall prediction performance so far. That is, we prefer the predictor with the lowest accumulated prediction error.

7 Methodology

To evaluate the SMM predictor, and the other two baseline predictors we monitor the dynamic runtime characteristics of the SPEC CPU2000 suite applications running on an IBM POWER4 [28] server platform. We use hardware performance counters to track application behavior using the AIX Performance Monitoring API (PMAPI). Our performance monitoring framework has negligible impact on system behavior, and captures overall application characteristics including all library and system calls performed by the application thread. The performance counters available in our system enables simultaneous monitoring of eight performance events with negligible overhead. For most of our evaluations we mainly focus on two key architectural metrics, instructions per cycle (IPC) and memory access rate. These two metrics are also commonly used in prior studies to demonstrate well-known application phase characteristics. They are widely available across architectures and have been employed in prior studies to guide workload-adaptive dynamic power management. All the presented prediction approaches, however, are not tied to a specific metric and can be applied to other architectural metrics of interest, based on the desired adaptation or monitoring goals.

In all of our predictor implementations, we use a 10 ms sampling period, in line with OS scheduling time scale. At each sampling point, our monitoring interface snapshots the performance counter registers and the difference of the current accrued counts from the prior sample indicates the current performance sample. From the elapsed counts and cycles, we derive our performance metrics of interest, which are then transferred to the prediction logic. As the raw measures are somewhat continuous values in the possible IPC and memory rate range, using the raw measures for pattern- or grammar-based prediction has limited returns. This is because, these approaches rely on labels rather than actual exact quantities. Therefore, we also apply a quantization step to the derived performance measures to bin them into specific ranges. For our evaluations, we use a relatively fine-grain binning approach with 20 quantization bins. This represents the high end of the experimented granularities, limiting within-bin quantization error range to 5 percent. We record the obtained quantized performance measure samples in a sample buffer, which is then used for the prediction and generating the patterns for the table-based and SMM predictors. Following this, each prediction algorithm performs its prediction for the succeeding application behavior.

8 Experimental Results

To evaluate SMM performance, we run a set of benchmarks on a real system and monitor performance characteristics via performance counters and predict future workload behavior via SMM and other predictors. We compare the SMM predictor to a table-based and last value predictors. We also investigate SMM predictor in-depth for its performance in relation to various parameters. In order to provide a fair comparison, we also perform a sensitivity analysis for the table-based predictor using different table sizes. Fig. 4 presents the normalized mean prediction errors for different table sizes. Our results confirm the previous studies [16] in that using table sizes larger than 1,024 does not provide significantly different prediction results with the only exception of `mgrid` where there is further improvement. Therefore, in our experiments we use a table-based predictor with a fixed size of 1,024 entries.

8.1 Metric Variation

Workloads with significant variability are particularly challenging for metric predictors. For slowly varying workloads last value predictor is very effective but for benchmarks with rapidly changing characteristics, it performs unfavorably. It is thus essential to understand the

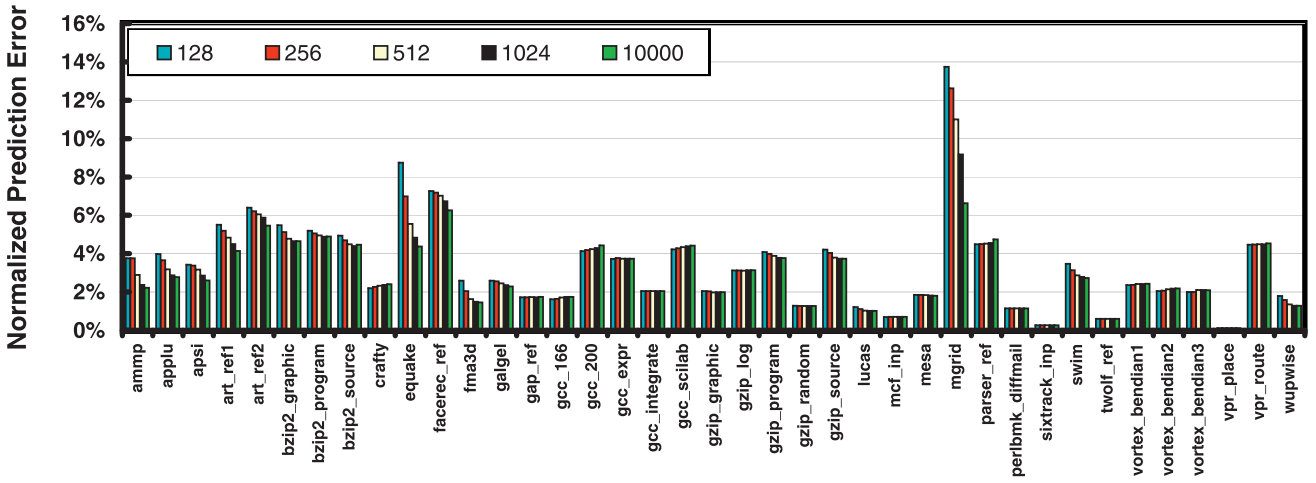

Fig. 4. Normalized mean prediction error for IPC using table-based predictor with table sizes of 128, 256, 512, 1024, 10000.

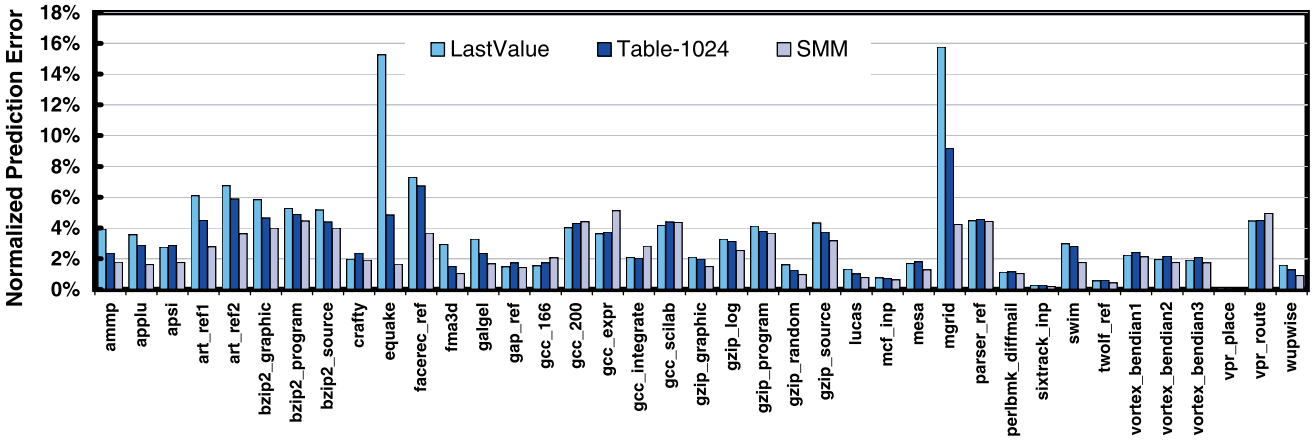

Fig. 5. Improvements with the SMM predictor, compared to last value and table-based predictors for IPC prediction.

variability and "predictability" of different workloads to properly judge the performance of various predictors.

We present two measures in Fig. 6 which show the available variation in the workloads used in this work. The upper plot shows the average *sample-to-sample* variation in the tracked metrics normalized to the overall dynamic range of the workload. In the lower plot, the magnitude of variation is decoupled from the *occurrence* of a variation by profiling how often two consecutive samples belong to different phases. This measure is useful to understand the deviation of each workload from a purely *flat* workload. In the figure, the benchmarks are sorted in increasing variability. Thus, workloads toward the right end exhibit the highest variability with the last six workloads showing more than 5 percent average sample-to-sample variation.

8.2 Metric Tracking Using SMM Predictor

SMM has several parameters that can have significant effect on the performance. The model order n (i.e., maximum sequence/pattern length) is one such parameter. The optimal n achieving the best performance depends on the specific benchmark and available data to train the SMM. Here, we set $n = 8$ for fair comparison, because the table-based predictor, which is considered as one of the baseline methods uses a sequence length of 8 [16]. As we will demonstrate that a smaller n value could be just as good in terms of performance. We also use the last value predictor both as a baseline to compare and as a back-off predictor for the table-based predictor. For SMM using larger model orders can allow us modeling longer patterns but the underlying model parameters may not be robustly estimated. However, using smaller model orders may not have enough predictive power to model any patterns embedded in the metric sequence. We believe that $6 \leq n \leq 8$ is a reasonable compromise between these two competing goals.

We provide prediction error rates for all the benchmarks comparing last value, table-based and SMM predictors in Fig. 5. The SMM predictor improves prediction errors by 19 percent on average over all the experimented workloads compared to the table-based predictor. This improvement is even further emphasized, with 43 percent relative reduction in prediction error for the top five highly varying applications, namely as `mgrid`, `equake`, `facerec`, `art_ref1`,

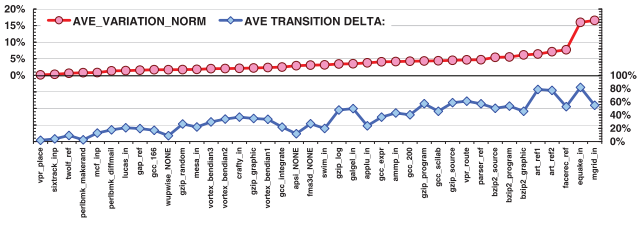

Fig. 6. IPC variation in benchmarks.

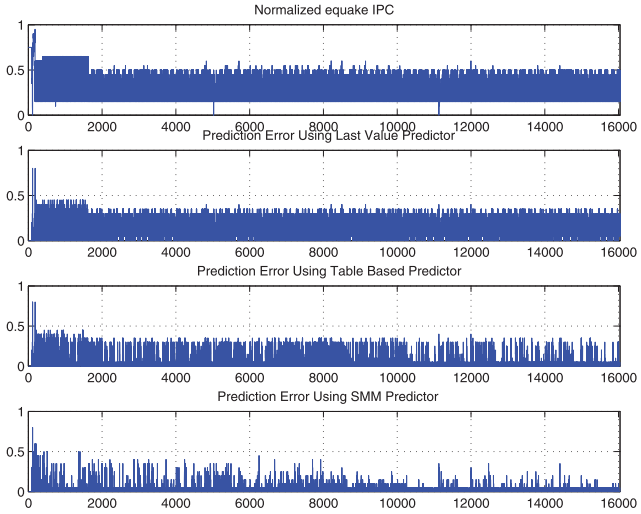

Fig. 7. Normalized IPC (to its max) prediction errors for different predictors over the quantized samples.

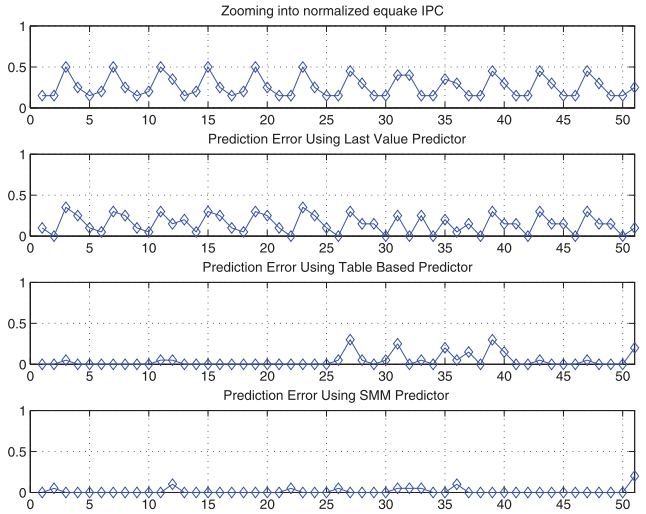

Fig. 8. Normalized IPC prediction errors for different predictors over an execution segment.

art_ref2. In comparison to the last value predictor, SMM improves prediction errors on top five highly varying applications by 63 percent. The largest improvements of about 10-fold (15.3 percent versus 1.6 percent) and 3-fold (4.8 percent versus 1.6 percent) are achieved compared to the last value and table based predictors, respectively, on the equake benchmark.

Fig. 7 shows how the prediction performance of the different predictors change over time, with the equake benchmark. In the first panel the entire IPC sequence of equake, normalized to its max, is plotted. In the other panels, the prediction errors are plotted for the last value, table-based and SMM predictors. We clearly observe from the plots that the table-based predictor outperforms the last value predictor and the SMM predictor outperforms both predictors. As expected, as the amount of data increases the SMM parameter estimation becomes more reliable. As a result the SMM prediction performance improves (i.e., making smaller prediction errors) toward the end of the figure. Online learning and adaptation are two critical features that an adaptive predictor must have. SMM has the ability to learn and adapt itself constantly. As it learns and adapts itself to the changing phase behavior, the prediction accuracy improves. On the other hand, in the same figure, the errors for last value predictor appears to be evenly distributed across the time scale. This is expected, as this simple predictor does not employ any adaptation based on previously observed application behavior. The table-based predictor's performance also improves slightly with increasing amount of data, since it also aims to discern the patterns in application behavior. However, the improvements in the prediction accuracy do not come close to those with the SMM predictor, since the SMM predictor can successfully model both long-term and varying-duration application characteristics.

We can gain insight about different predictors by looking at their detailed prediction results. In the first panel of Fig. 8, a segment of the normalized equake IPC data is plotted. This segment of the data is somewhat periodic with significant differences in values between consecutive samples. In the second panel, the corresponding normalized prediction error is plotted using the last value predictor. As expected the errors are almost on the same scale as the data samples due to large sample-to-sample variations. The table-based predictor models the periodicity in the data to some extent, as seen in the beginning and end of the plot. However, each time there is a slight variation in the observed behavior, the fixed-pattern-based approach fails and the predictor backs off to the last value predictor as seen between samples 25 and 40. However, SMM is very robust to small variations in the patterns and is able to provide accurate prediction.

We emphasize the ability of SMM to model pattern of different length in Fig. 9. In the figure, we highlight two patterns with boxes, which contain identical samples of length 6. Even though, the first five samples are the same in both boxes, table-based predictor does not have any match simply because it stores samples of length 8. When there is no match, then it backs off to the last value predictor, which does not produce an accurate prediction, as pointed out by the arrows. However, SMM backs off to lower order models and finds the matching pattern of length 6, which is observed in the first box, and uses it to perform accurate

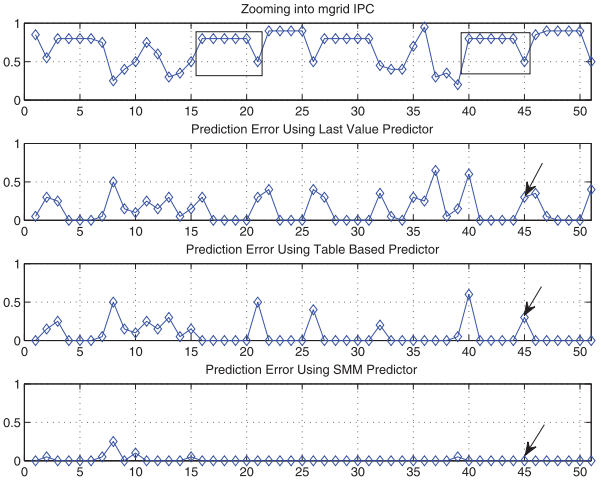

Fig. 9. Benefit of backing off to lower order models for SMM.

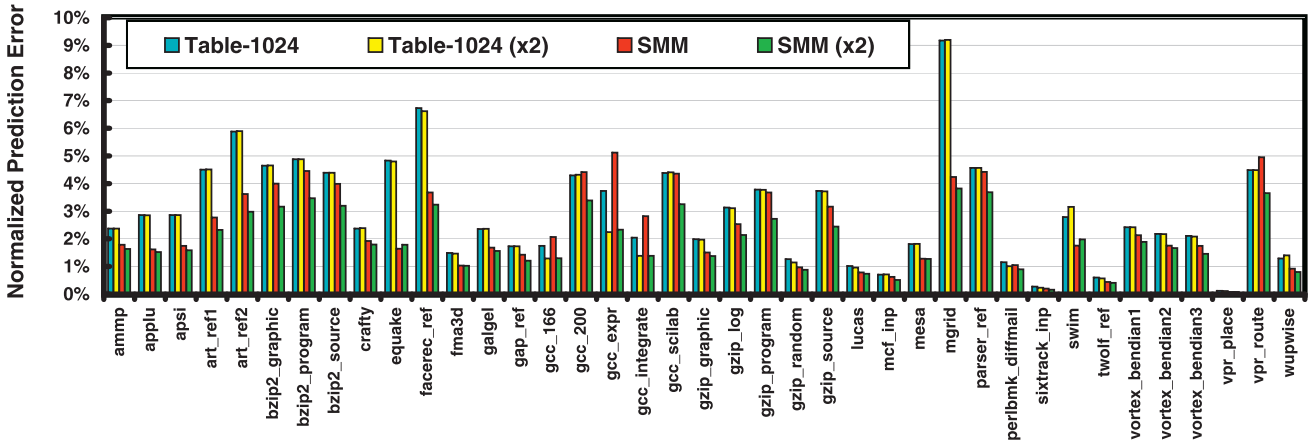

Fig. 10. Improvement in SMM predictor accuracy with longer benchmark execution times.

prediction in the second box. This is of course the case if the next sample 10 (normalized value in the figure is 0.5) has the highest probability among all the possible outputs $(1, \ldots, 20)$ given the past five samples in the box.

Another key feature of SMM predictor is its ability to model long range patterns. In order to demonstrate this feature, we extend the runtime duration of the experimented benchmarks by a factor of 2 (shown as x2 in Fig. 10) by concatenating workloads back to back. By doing so, we introduce long term patterns to the data. In fact, what we are doing is not hypothetical, in practice it can very well be observed, where one application finishes another one starts and the first application starts to run again after the second application. With concatenated data, we provide the predictors with runtime histories which are twice as long as the original runs. In Fig. 10, we plot the normalized mean prediction errors for the original benchmark runs and the extended benchmark runs. We also plot the results for the table-based predictor in both cases. Not surprisingly, the prediction performance of the table-based predictor shows almost no improvement except for two benchmarks. Those two benchmarks are `gcc_exp` and `gcc_integrate`, which are both very short in duration. SMM, on the other hand, provides significant improvements for almost all the benchmarks. The only exception is `swim`, where we observe a slight increase in prediction error of both SMM and table-based predictors. It is not surprising to see that SMM performance improves on the extended-duration benchmarks, since SMM has a memory of observing all the prior patterns. SMM starts to improve the prediction in the second half of the duplicated benchmark data. The average improvement across all benchmarks is 15 percent for the SMM predictor, whereas it is only 3 percent for the table-based predictor. This shows the substantial benefits of the SMM predictor in the longer term.

Even though we report the prediction experiments for IPC, we also predicted memory access rates using all three predictors. We observe similar results, where table-based predictor outperforms the last value predictor by 18 percent on average and SMM predictor outperforms table-based predictor by 29 percent across benchmarks.

The strength of the SMM predictor comes from its probabilistic nature, which sets it apart from the existing predictors. While last value or table-based predictors rely only on recent history and discard old pattern information in favor of new observations, SMM does not only rely on this temporal dependence but also takes into account previously observed patterns when it is making the prediction. When capacity requirements dictate discarding some patterns, pruning is done not by temporal proximity, but with respect to the probabilities of the patterns.

8.3 Model Order versus Performance

We mentioned that the performance of SMM predictor largely depends on the model order (i.e., n) and the available data to estimate the model parameters. Smaller model order limits the predictive power of the SMM predictor. However, using larger model orders may adversely affect the parameter estimation step, which in turn may degrade the prediction performance. We are not aware of a "grand" recipe that would tell us the optimal model order given the amount of training data. However, one should keep in mind that even though the metric data sizes for specific applications are fixed, in reality SMM is designed to operate continuously during system uptime, with a very large set of running metric samples. In that case using larger model order may benefit the prediction performance.

In Fig. 11, we provide prediction errors for all benchmarks with model orders: {4,6,8}. The results indicate that each benchmark can be described better with a specific model order and may achieve the smallest prediction accuracy. In general, as the model order increases, overall prediction accuracy also increases leading to lower prediction errors. Typically, models with larger model order are good in describing the fine structure in the data. However, in the absence of large data, this comes at the expense of poor generalization for the unseen future observations, which are to be predicted. There are few exceptions to this overall trend. For example, model order 4 ($\mathrm{n} = 4$) for `parser_ref` achieves slightly lower prediction error than model order 8 ($n = 8$). We also observe that the improvements with larger model order appears to be leveling off for $n \geq 6$ for most of the benchmarks with the exception of `mgrid`, where there is further possible reduction in prediction error with larger model sizes.

8.4 Metric Tracking Using SMM-Interp Predictor

We observed in Fig. 5 that SMM predictor performed worse than the last value predictor for several bechmarks, namely gcc_* series. One common feature that distinguishes

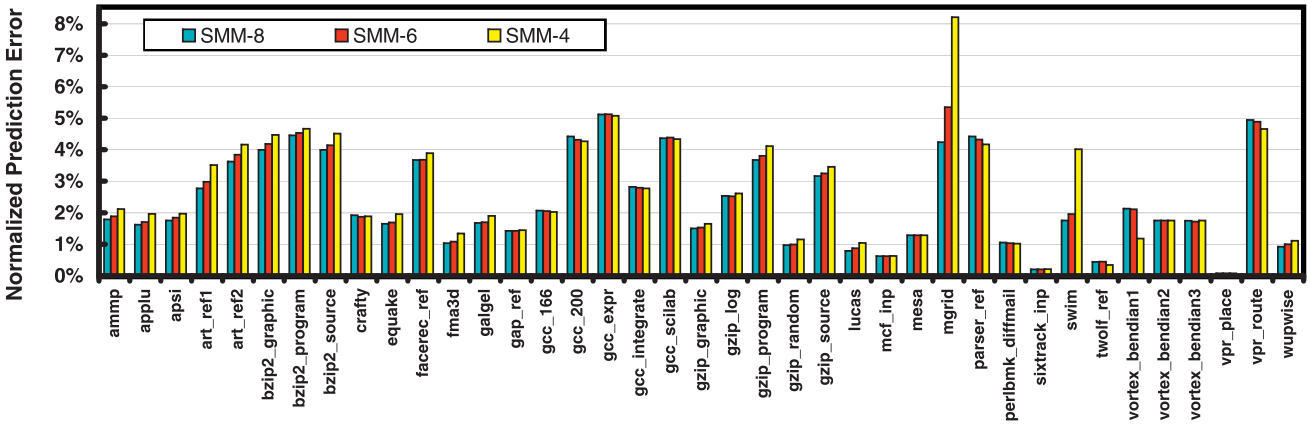

Fig. 11. SMM prediction performance for different model orders of 4, 6, and 8.

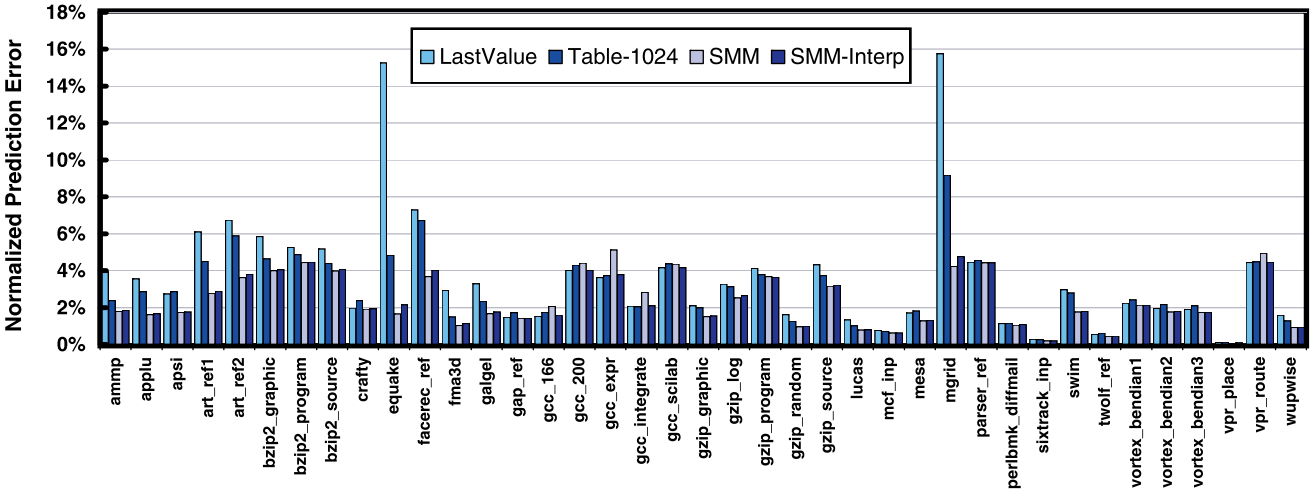

Fig. 12. Comparison of the SMM-Interp with the SMM, last value and table-based predictors.

gcc_∗ series from the other benchmarks is their fairly short duration. SMM needs relatively long samples to discover the underlying patterns in the data and also to robustly estimate model parameters, which are probabilities.

In Fig. 12, SMM-Interp is compared against SMM, table-based and last value predictors. The results show that SMM-Interp improves the prediction accuracy for all the *gcc_∗* series, while keeping the performance accuracy virtually the same for the other benchmarks. These results confirm that SMM-Interp has the ability to model local short-term changes in the data by steering prediction to exploit them.

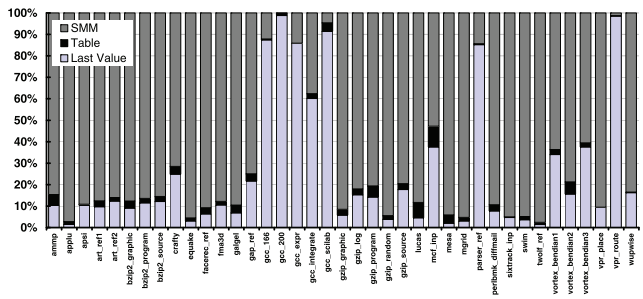

Fig. 13. Model usage statistics across benchmarks within the SMM-Interp scheme.

In Fig. 13, we present the predictor usage distribution across SMM-Interp components, namely SMM, table based and last value predictors. SMM predictor is picked as the best predictor more often than the other predictors for 32/39 of the benchmarks. The "gcc" benchmarks are all short in duration for the SMM to have reliable parameter estimates. As such, it is not surprising to see the last value predictor emerging as the best predictor. For `parser_ref` and `vpr_route`, we believe there was not any significant repetitive pattern in the data, otherwise SMM would have discovered the repetitive patterns.

Using table based and last value predictors in the prediction process along with SMM satisfies our goal of tracking local short-term changing patterns of a metric sequence. SMM-Interp incorporates both global pattern predictor embodied in SMM and an added *cache* component embodied in last value and table-based predictors, which tracks short-term fluctuations in metric behavior.

8.5 Application of SMM to Power Management

SMM predictor can be employed in various adaptive management setting that can benefit from proactively tuning hardware or software to the expected application characteristics. Some of these potential settings include dynamically configurable hardware such as cache partitioning or pipeline scaling, and power optimization techniques such as power gating and voltage, frequency scaling. Here,

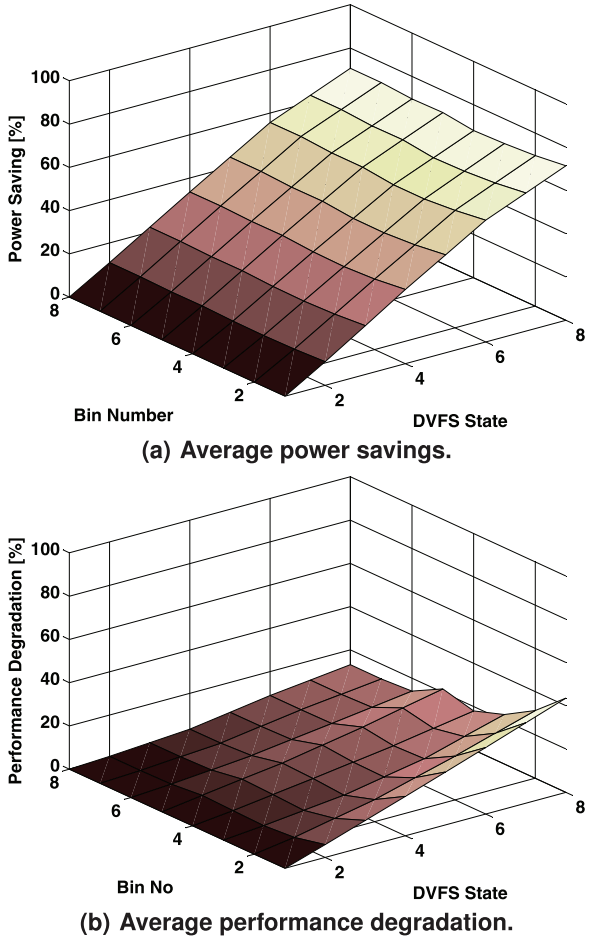

Fig. 14. Power savings and performance degradation at different DVFS states for different bins.

we explore the application of the SMM predictor to dynamic power management, where we specifically look into proactively configuring processor power states. We use SMM predictor to predict the memory access rate of applications, which is the commonly used architectural measure for guiding dynamic voltage and frequency scaling (DVFS) settings. Such workload-behavior-based DVFS is well studied in prior work [30], [31]. These show that the memory access rates of applications are a strong indicator of the appropriate DVFS state that an application can run with limited performance degradation. Applications with higher amount of memory accesses exhibit higher potential for running at lower frequencies with much less impact on application performance.

The primary approach for dynamic DVFS management is to categorize application execution into different characteristic regions, or *phases*, and to assign different phases to the corresponding DVFS settings. To accomplish this, we first profile the memory access rates of a range of different applications. This helps determine the *search space* of DVFS related application characteristics. Based on the observed aggregate view, we categorize different execution characteristics into different representative bins. We reference actual power and performance measurements collected at different DVFS states [13] to define the bin boundaries and to characterize the power-performance tradeoffs of running each representative bin at different DVFS settings. The number of bins is dictated by the

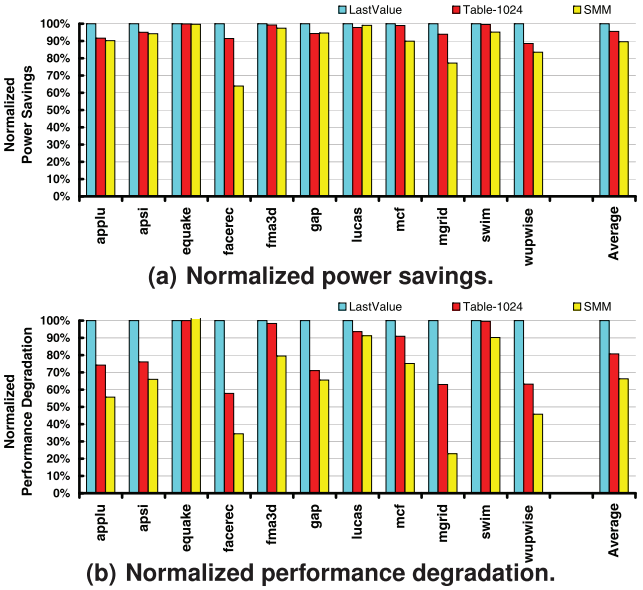

Fig. 15. Power savings and performance degradation achieved by the three prediction methods.

number of available DVFS states. Therefore, we use eight bins, corresponding to eight DVFS states. Then, at runtime, as the SMM predictor predicts the next application phase, the predicted phase, i.e., bin_i, dictates the corresponding processor DVFS setting i.

We show the corresponding power/performance tradeoffs for the chosen eight bins in Figs. 14a and 14b. Here, $bin1$ corresponds to an execution region with minimal memory accesses and $DVFSstate1$ represents the highest DVFS setting. When the execution behavior is similar to $bin1$ running the application in DVFS states other than 1 result in significant performance degradation. In contrast, using a higher DVFS state for an execution region with higher memory accesses—i.e., higher bin number—is much more beneficial due to the smaller performance degradation impact. In this memory bound case running the application at a lower DVFS setting still improves performance slightly. However, this comes at a significantly higher power cost and lost potential power savings.

We again use the last value predictor and the table-based predictor as baselines in our evaluations, and show the relative improvements with the SMM predictor. We predict application behavior at runtime at fixed time intervals, and use the resulting predictions to guide the future DVFS settings for the following period. For the presented cost-benefit analysis, we monitor the achieved power savings and the associated performance degradation at the end of each interval compared to baseline execution with no dynamic power management.

For all three predictors and for all applications, we collect the power savings and performance degradation achieved compared to the baseline without power management, and use this for our comparative evaluation of SMM. We accumulate the overall power savings and performance degradation throughout the execution of the applications and determine the average power savings and the experienced performance degradation for each application. Figs. 15a and 15b depict these results. While we have included all applications in our experiments, the figures

show only a subset of these. The excluded benchmarks perform similarly across predictors for one of the two reasons: all the excluded benchmarks either exhibit very low variability (thus very simple prediction such as last value predictor suffices) or they are highly CPU-bound, consistently operating at the highest DVFS setting phase.

In Figs. 15a and 15b, we depict the normalized power savings and performance degradations relative to the last value predictor. Here, we choose to show the tradeoffs achieved with the other predictors relative to the last value predictor as the main purpose of this work is to underline the additional benefit achieved with the SMM predictor compared to the other predictors. Among the three predictors, it is actually the last value predictor, which achieves somewhat higher power savings, followed by the table-based predictor. While this may sound counter to what one might expect, the real potential of the SMM predictor is seen in the performance degradation figures, as shown in Fig. 15b. Here, the distinction among the three predictors is much more significant, where the table-based predictor performs better than the last value predictor and the SMM predictor significantly outperforms both predictors. This outlines the benefit of the SMM predictor's higher prediction accuracy. While the SMM predictor achieves slightly lower power savings, with less than 10 percent difference compared to last value and less than 5 percent compared to the table-based predictor, it significantly reduces the performance impact. The SMM predictor reduces overall performance degradation by 34 percent compared to the last value predictor and by 19 percent compared to the table-based predictor.

9 Conclusions

We presented a new predictor called SMM for predicting dynamically varying program behavior. SMM is a probabilistic adaptive model. It has the ability to learn application characteristics at runtime and to capture patterns at different scales in application behavior. SMM has three key features, which set it apart from the existing predictors. First, it models long term global patterns in application behavior. Second, the predictor can respond to variable-length patterns, and thus it is resilient to small fluctuations in the observed patterns. Last, the SMM predictor has the ability to adapt itself; as it learns more it predicts better. We present a series of experiments that demonstrates these strengths as well as its superior accuracy. These studies show that the SMM predictor reduces prediction errors by up to 10X and 3X compared to the last value and table-based predictors, with an average improvement of more than 60 and 40 percent, respectively, for highly varying benchmarks. We also introduced an interpolation scheme; SMM-Interp, which weights the recently observed patterns favorably. SMM-Interp provides improvements over SMM for benchmarks that are short in duration while keeping the performance intact for the longer benchmarks. We also show the application of the SMM predictor to dynamic power management, where improved prediction accuracy with the SMM predictor achieves superior power-performance tradeoffs compared to the other predictors.

References

[1] D. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster, "Dynamically Tuning Processor Resources with Adaptive Processing," *Computer*, vol. 36, no. 12, pp. 43-51, Dec. 2003.

[2] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's Law through EPI Throttling," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2005.

[3] R. Balasubramonian, D.H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," *Proc. IEEE 33rd Ann. Int'l Symp. Microarchitecture (MICRO)*, 2000.

[4] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner, "Event-Driven Energy Accounting for Dynamic Thermal Management," *Proc. Workshop Compilers and Operating Systems for Low Power (COLP)*, 2003.

[5] W.L. Bircher, M. Valluri, J. Law, and L.K. John, "Runtime Identification of Microprocessor Energy Saving Opportunities," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED)*, 2005.

[6] P.E. Brown, S.A. Della Pietra, V.J. Della Pietra, J.C. Lai, and R.L. Mercer, "An Estimate of an Upper Bound for the Entropy of English," *Computational Linguistics*, vol. 18, no. 1, pp. 31-40, 1992.

[7] S. Chen and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," Technical Report TR-10-98, Computer Science Group, Harvard Univ., 1998.

[8] K. Choi, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling Based on Workload Decomposition," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED)*, 2004.

[9] A. Dhodapkar and J. Smith, "Managing Multi-Configurable Hardware via Dynamic Working Set Analysis," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2002.

[10] E. Duesterwald, C. Cascaval, and S. Dwarkadas, "Characterizing and Predicting Program Behavior and Its Variability," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT)*, 2003.

[11] J. Goodman and J. Gao, "Language Model Size Reduction by Pruning and Clustering," *Proc. Int'l Conf. Spoken Language Processing (ICSLP)*, 2000.

[12] M. Huang, J. Renau, and J. Torrellas, "Positional Adaptation of Processors: Application to Energy Reduction," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2003.

[13] C. Isci, G. Contreras, and M. Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management," *Proc. IEEE 39th Ann. Int'l Symp. Microarchitecture (MICRO)*, 2006.

[14] C. Isci and M. Martonosi, "Identifying Program Power Phase Behavior Using Power Vectors," *Proc. Int'l Workshop Workload Characterization (WWC)*, 2003.

[15] C. Isci and M. Martonosi, "Detecting Recurrent Phase Behavior under Real-System Variability," *Proc. Int'l Symp. Workload Characterization (IISWC)*, 2005.

[16] C. Isci, M. Martonosi, and A. Buyuktosunoglu, "Long-Term Workload Phases: Duration Predictions and Applications to DVFS," *IEEE Micro*, vol. 25, no. 5, pp. 39-51, Sept./Oct. 2005.

[17] F. Jelinek and R.L. Mercer, "Interpolated Estimation of Markov Source Parameters from Sparse Data," *Proc. Workshop Pattern Recognition in Practice*, E.S. Gelsema and L.N. Kanal, eds., 1980.

[18] R. Kuhn and R. Mori, "A Cache-Based Natural Language Model for Speech Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 6, pp. 570-583, June 1990.

[19] J. Lau, S. Schoenmackers, and B. Calder, "Transition Phase Classification and Prediction," *Proc. Int'l Symp. High Performance Computer Architecture (HPCA)*, 2005.

[20] R. Sarikaya and A. Buyuktosunoglu, "Predicting Program Behavior Based on Objective Function Minimization," *Proc. Int'l Symp. Workload Characterization (IISWC)*, 2007.

[21] R. Sarikaya and A. Buyuktosunoglu, "A Unified Prediction Method for Predicting Program Behavior," *IEEE Trans. Computers*, vol. 59, no. 2, pp. 272-282, Feb. 2010.

[22] R. Sarikaya, C. Isci, and A. Buyuktosunoglu, "Program Behavior Prediction Using a Statistical Metric Model," *ACM Sigmetrics*, vol. 38, pp. 371-372, 2010.

[23] R. Sarikaya, C. Isci, and A. Buyuktosunoglu, "Runtime Workload Behavior Prediction Using Statistical Metric Modeling with Application to Dynamic Power Management," *Proc. Int'l Symp. Workload Characterization (IISWC)*, 2010.

[24] X. Shen, Y. Zhong, and C. Ding, "Locality Phase Prediction," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.
[25] X. Shen and C. Zhang, and C. Ding, and M. Scott, and S. Dwarkadas, and M. Ogihara, "Analysis of Input-Dependent Program Behavior Using Active Profiling," *Proc. Workshop Experimental Computer Science*, 2007.
[26] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.
[27] T. Sherwood, S. Sair, and B. Calder, "Phase Tracking and Prediction," *Proc. Int'l Symp. Computer Architecture (ISCA)*, 2003.
[28] J.M. Tendler, J.S. Dodson, S. Fields, H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM J. Research and Development*, vol. 46, no. 1, pp. 5-25, 2002.
[29] F. Vandeputte, L. Eeckhout, and K.D. Bosschere, "A Detailed Study on Phase Predictors," *Proc. 11th Int'l Euro-Par Conf.*, 2005.
[30] A. Weissel and F. Bellosa, "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," *Proc. Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2002.
[31] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D.W. Clark, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," *Proc. IEEE 38th Ann. Int'l Symp. Microarchitecture (MICRO)*, 2005.
[32] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic Tracking of Page Miss Ratio Curve for Memory Management," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.

Ruhi Sarikaya received the BS degree from Bilkent University, Turkey in 1995, the MS degree from Clemson University, SC in 1997, and the PhD degree from Duke University, NC in 2001 all in electrical and computer engineering. He was a research staff member and a team leader in the Human Language Technologies Group at IBM T.J. Watson Research Center. He is currently a manager/principal scientist at Microsoft Corporation. He has published more than 60 technical papers in refereed journal and conference proceedings and holder of 13 patents. At IBM he has received numerous awards for his work including an Outstanding Technical Achievement Award and two Research Division Awards. Prior to joining IBM in 2001 he was a researcher at the Center for Spoken Language Research (CSLR) at the University of Colorado at Boulder for two years. He also spent the summer of 1999 at the Panasonic Speech Technology Laboratory, Santa Barbara, CA. Over the years he has been involved in the organization and technical committees of a number of conference and workshops. He also gave keynote speeches in several conferences. Currently, he has been serving as associate editors of *IEEE Transactions on Speech, Audio and Language Processing* and *IEEE Signal Processing Letters*. His past and present research interests span statistical modeling, speech recognition, natural language processing, machine learning, machine translation and digital signal processing. He is a senior member of the IEEE, ACL, and ISCA.

Canturk Isci received the BS degree in electrical engineering from Bilkent University, the MSc degree with Distinction in VLSI system design from the University of Westminster, and the PhD degree in computer engineering from Princeton University. He is a research staff member in the Distributed Systems Department at the Thomas J. Watson Research Center. His research interests are virtualization, data center energy management, and microarchitectural and system-level techniques for workload-adaptive and energy-efficient computing. He is a member of the IEEE.

Alper Buyuktosunoglu received the PhD degree in electrical and computer engineering from the University of Rochester. Currently, he is a research staff member in Reliability and Power-Aware Microarchitecture Department at IBM T. J. Watson Research Center. He has been involved in research and development work in support of IBM p-series and z-series microprocessors in the area of power-aware computer architectures. His research interests are in the area of high performance, power/reliability-aware computer architectures. He has over 35 pending/issued patents, has received several IBM-internal awards, has published over 45 papers, and has served on various conference technical program committees in these areas. He is a senior member of the IEEE and is currently serving on the editorial board of *IEEE MICRO*.