**RC24899 (W0911-059) November 6, 2009**
**Computer Science**

# IBM Research Report

## Program Behavior Prediction Using a Statistical Metric Model

**Ruhi Sarikaya, Canturk Isci, Alper Buyuktosunoglu**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Program Behavior Prediction

# Using a Statistical Metric Model

**ABSTRACT**

Adaptive computing systems rely on accurate projections of program behavior to understand and respond to the dynamically-varying application characteristics. In this study, we propose a Statistical Metric Model (SMM) that is system- and metric–independent for predicting program phases. SMM is a probability distribution over workload patterns and it attempts to model how frequently a specific behavior occurs. Maximum Likelihood Estimation (MLE) criterion is used to estimate the parameters of the SMM. The model parameters are further refined with a smoothing method to improve the robustness of the proposed program model. Finite sequences of execution metrics, obtained from running workloads are used to train the SMM. The SMM learns the application patterns during runtime, and at the same time predicts the upcoming program phases based on what it has learned so far. The finite prediction horizon can be adjusted based on the available data, which is used to estimate the model parameters. An extensive and rigorous series of prediction experiments demonstrate the superior performance of the SMM predictor over existing predictors on a wide range of benchmarks. For some benchmarks the improvements in prediction error are as much as 10-fold and 3-fold respectively, compared to the existing last-value and table-based prediction approaches.

## 1 Introduction

Adaptive management of power and performance has utmost importance in current generation microprocessors. Today's microprocessors rely on adaptive power management schemes that try to adapt to changing behavior of applications. Typically, these schemes exploit changing phase behavior at certain time granularities of application to dynamically trade-off power and performance. In most cases, the techniques are reactive in that they are enabled once the phase transition occurs. Obviously, reactive systems do not predict the future values and durations of phases for an application. In cases, where the application phase behavior is very dynamic the reactive systems may lead to possible oscillations for the adaptive control system.

Ideally, one would like to predict future behavior of an application based on its past behavior and proactively manage the system to avoid instability. Obviously, such a scheme heavily relies on accurate prediction capability of application behavior. Prior work proposed such proactive methods based on simple pattern history tables [17], recent program flow behavior [22, 20], and statistics over recent performance characteristics [12]. These predictors aim to predict application behavior by keeping track of the recently observed patterns or statistics in the observed application features, which can be then used to guide dynamic management decisions.

While in general these predictors prove to be effective in many scenarios, there are two critical caveats due to i) their deficiency in predicting global long range patterns, and ii) their inability to model patterns of varying length. Figure 1 shows an example to why a simple table based predictor cannot provide a robust mechanism for workload behavior prediction. This figure shows an actual trace from the SPECCPU 2000 `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 following from the first pattern, this small fluctuation actually leads to a pattern mismatch. As a result, the table based predictor backs off to the last value predictor, and the following two consecutive samples labeled as $i$ and $j$ are mispredicted. However, an advanced predictor that can model varying size patterns, can actually discern the repetition at a smaller sequence length and can correctly predict both $i$ and $j$ from prior observations. This shorter pattern discovered by our proposed predictor is highlighted with the smaller enclosed patterns in the figure. As shown in Figure 1, the proposed predictor can be effective over both table based and other historical predictors. While both of the latter approaches are potentially prone to mispredictions due to fluctuations in pattern history, the proposed novel predictor can model patterns of varying length.

In this paper, we propose a Statistical Metric Model (SMM) for metric prediction. The SMM estimates the probability of a finite a sequence. The probabilities generated by the SMM are used to predict the most likely next sample value given the history pattern. The SMM does not have the shortcomings of the table based predictor. As such, it has the ability to model patterns of different length and it can effectively model long term patterns. A comprehensive set of experiments demonstrates the effectiveness of the SMM predictor in comparison to the previously proposed predictors.

The rest of the paper is organized as follows. Section 2 gives an overview of prior work for program phase prediction. Section 3 describes the baseline predictors that SMM is compared against. Section 4 provides a detailed description of the SMM predictor. Section 5 summarizes the methodology. An extensive series of experiments are reported in Section 6 followed by the conclusions in Section 7.

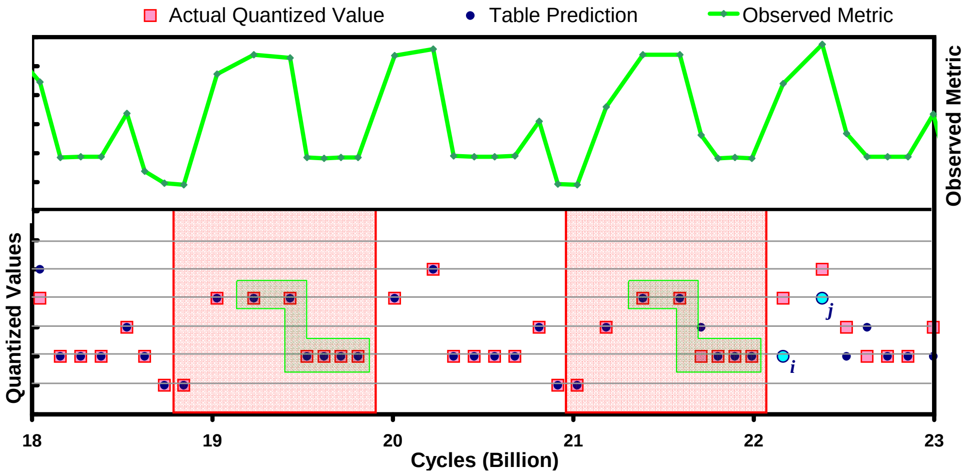

Figure 1: Vulnerability of table based predictor to the fluctuations in the metric sequence of fixed length.

## 2 Related Work

A large body of research focuses on tracking, characterizing and predicting application characteristics. These studies investigate characterization and prediction techniques for various metrics that include runtime performance monitoring counts, programmatical flow information and operating system statistics.

One line of research explores characterization of observed behavior via runtime statistic collection and architectural or system-level simulations [15, 16, 21, 5, 7, 11, 13]. These techniques mainly focus on interpreting specific workload execution behavior and detecting some indicative characteristics. Other characterization work [6, 8, 10, 9, 23] uses system statistics to guide dynamic adaptations such as power and thermal management. While, these techniques provide significant insights to workload behavior and its impact on dynamic management decisions, they do not explore the approaches for online prediction of future behavior. The dynamic management techniques resulting from these can be considered as reactive approaches. Our proposed approach aims to provide the necessary means for proactive adaptations.

In addition to the characterization studies, significant number of studies also target at predicting future application behavior [12, 17, 14, 22, 18, 19, 24]. Duesterwald et al. [12] describe different statistical and table based predictors for within- and across-metric predictions of performance monitoring information. Sarikaya et al. [19] describe an optimal prediction technique based on a predictive least squares minimization [19]. Isci et al. [14] develop a table based runtime predictor to predict future behavior from past pattern characteristics. Zhou et al. monitor memory access patterns to estimate memory behavior of workloads [24]. This approach is used for energy efficient memory allocation. Sherwood et al. describe micro–architectural phase predictors based on the repetitive characteristics of program flow behavior [22]. Shen et al. detect such repetitions from reuse distance patterns

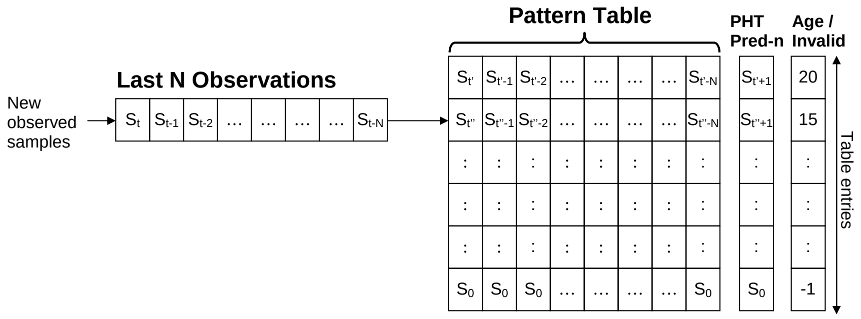
Pattern Table
PHT Pred-n
Age / Invalid
Last N Observations
New observed samples
$S_t$ $S_{t-1}$ $S_{t-2}$ … … … … $S_{t-N}$
$S_{t'}$ $S_{t'-1}$ $S_{t'-2}$ … … … … $S_{t'-N}$
$S_{t'+1}$
20
$S_{t''}$ $S_{t''-1}$ $S_{t''-2}$ … … … … $S_{t''-N}$
$S_{t''+1}$
15
$S_0$ $S_0$ $S_0$ … … … … $S_0$
$S_0$
-1
Table entries

Figure 2: Table based predictor structure.

for dynamic memory configurations via profiling and instrumentation [20]. In summary, all these studies provide useful prediction techniques suitable for different applications. This paper, on the other hand, shows the benefit of statistically incorporating the varying pattern history tracking. It also improves prediction accuracy over existing approaches by modeling long term patterns.

# 3 Baseline Predictors

This section gives a brief overview of the predictors that serve as the baseline in our work.

## 3.1 Last Value Predictor

One simple, yet effective metric predictor is the last value predictor. Last value predictor assumes that the next metric sample is the same as the last observed sample. The last value predictor can be very accurate for slowly varying metric sequences. However, its performance suffers greatly for rapidly varying metrics. The main appealing feature of the last value predictor is its simplicity from the computational and storage perspective. Other more elaborate predictors introduce computational and storage overheads to the prediction process.

## 3.2 Table Based Predictor

Table based predictors track the patterns in prior application history to deduce future workload characteristics. Such approaches rely on the repetitive application execution characteristics to produce a reliable model of future behavior. An example of these predictors, namely a global phase history table predictor [14] is depicted in Figure 2. Such a table based predictor can be implemented either in software or hardware depending on the desired prediction quantum and policy application time granularities [14, 12, 22].

The table based predictor consists of a global register that tracks a certain number of—specified as N— most recently observed 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 up to a certain number of previously encountered patterns. The predictions for the future behavior is also deterministically encoded into this table such that a prediction is 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 when the pattern table is full. 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 Intuition for SMM

The SMM is inspired from the way natural language is generated. In a natural language we use words to construct sentences. The SMM treats the metric samples as the words in a language and builds a language model for each metric. Note that, as metric samples are real numbers one has to quantize the real numbers into a set of discrete values, 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 metric, 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 metric 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 the natural language modeling and the SMM to explain the concepts used in this work.

## 4.2 Foundations of SMM

We consider a metric as a sequence of quantized sample values. From now on, we refer to a "quantized sample" simply as "sample". The SMM is a conditional distribution on the identity of the $i$th sample in a metric sequence, given the identities of all previous samples. In other words, the next sample in a metric 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 $n$ samples, where $n$ depends on the amount of available data to estimate the model parameters. Again, going back to natural language modeling analogy, in general, what word we will speak next, depends more on the most recent previous $n$ words than the words we have spoken a while ago.

The SMM we use is based on a class of Markov models, which is known as, the $n$-gram models [4]. $N$-gram models have received intensive research since their invention and have been widely used in speech and natural language processing [25]. The parameters of $n$-gram models are estimated from a large training text. The models produce a reasonable nonzero probability for every word in the vocabulary. From a theoretical perspective SMM is identical to the $n$–gram models. Here $n$ refers to the maximum length of the finite sequence of the metric samples (e.g. patterns of $n = 4$ samples as given in Figure 3). The probability of the $n$th 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 updated as many times as the observed metric samples.

The SMM model of order $n = 4$ is shown in Figure 3. The model has two set of entries: the finite sequences and the associated probabilities with each sequence. 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 $m$ $(1 \leq m \leq n)$, increasing the likelihood of finding a matching subsequence for a given finite sequence.

There is an inverse relationship between the predictive power of the SMM and robust parameter estimation. As $n$ increases the predictive power of SMM increases at the expense of unreliable parameter estimation, which in turn may start to hurt the predictive power of the model. 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. However, we also provide an evaluation of SMM performance with smaller $n$. By restricting the conditioning information to the previous seven $(n - 1)$ 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 $n$. Converting the entire sequence of metric samples that are observed up to now to a set of finite sequence of $n$ samples, allows us to compare any sequence with any other sequence in an efficient manner.

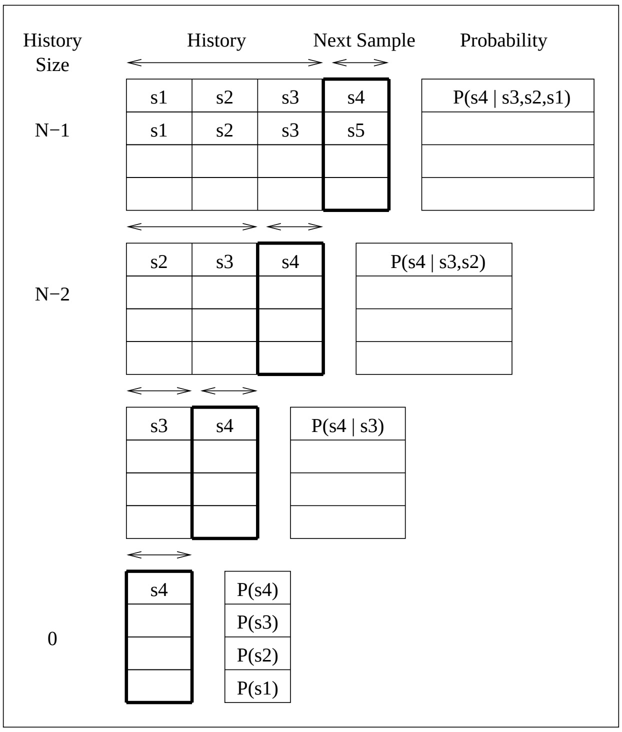

Figure 3: Description of the statistical metric model with back-off for $n = 4$.

## 4.3 Formulation of SMM

The SMM model is a probability distribution, $P(s)$, over L samples $S = s_1, s_2, ..., s_L$, that attempts to reflect the frequency with which each finite sequence $s = s_1, s_2, ..., s_l$ $(l < L)$ occurs in a metric.

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

In SMM, the probability of a string $P(s)$ 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. Without loss of generality, we can express the probability of a $s$, $P(s)$ as:

$$\begin{aligned} P(s) &= \prod_{i=1}^{l} P(s_i|s_1^{i-1}) \\ &\approx \prod_{i-1}^{l} P(s_i|s_{i-n+1}^{i-1}) \end{aligned} \tag{2}$$

where $s_i^j$ denotes samples $s_i, ..., s_j$. In order to simplify the description and formulation of the SMM we consider the case $n = 2$. The extension of formulation and results 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:

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

This probability distribution can be estimated with maximum likelihood estimation (MLE) technique:

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

where $C(x)$ denotes the number of times the sequence $x$ occurs in the metric. This is called the maximum likelihood (ML) estimate for $P(s_i|s_{i-1})$.

## 4.4 Motivation for Model Smoothing

There are fundamental problems with using ML based relative frequency counts due to data sparseness and robust model parameter estimation. We consider a specific example such as a bigram model ($n = 2$). Now assuming a total quantization level of V=20 bins, there are $20 \times 20 = 400$ bigrams. Setting $n = 8$, result in, potentially $20^8 = 25.6$ billion unique sequence of length 8. However, in our experiments we sample the data at every 10msec, and the longest metric data of SPECCPU 2000 consists less than 50K samples. In an unlikely case, where if there would be no identical sequence of 8 samples, this metric would generate 50K unique sequences of length 8. Clearly, this figure is negligible compared to 25.6 billion. As a result, the conventional probability estimate for each unseen event would be zero, and only a small fraction of all sequences of length 8 can be observed due to the small size of the training set.

While intuitive, the maximum likelihood estimate is a poor one when the amount of metric data is small compared to the size of the model being built, as is generally the case in metric modeling. For example, consider the situation where a pair of samples, say (1,20) does not occur in the metric data observed so far. Then, we have $P_{ML}(1, 20) = 0$, which is clearly inaccurate as this probability should be larger than zero. Just because an event has never been observed so far in the data does not mean it can not occur in the future. A zero bigram probability can lead to errors in metric prediction, as it disallows the bigram regardless of whether this bigram will be observed in the future. To overcome these shortcomings of the ML probability estimates, we have to apply some sort of "smoothing" to the relative frequencies and to make sure that each probability estimate is larger than zero. Smoothing is about modifying the conditional distributions away from pure relative frequency estimates in order to compensate for data sparsity.

### 4.5 Model Smoothing

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 once 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 $V$ 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.

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

where $P_{int}(s_i|s_{i-1})$ is the probability distribution for the interpolated model and $P_{ML}(s_i) = C(s_i)/L$ is the unigram model. The interpolated model achieves the behavior so that bigrams which involve common words are assigned higher probabilities [4, 1]. The weighting of the linear interpolation is estimated by maximizing the probability of new "held-out" data which differs from the data used to originally estimate the $n$-gram frequencies. In either case, this treatment normalizes the frequencies of formerly low-frequency $n$-grams to a significant level for the SMM. This type of smoothing was shown to obtain competitive performance when the training data is small, as is the case with the metrics considered here.

A more widely used set of smoothing methods is based on absolute discounting, which also involves interpolating higher order and lower models. However, instead of multiplying the higher order ML model with a factor, $\lambda$, 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 [25], 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:

$$\begin{aligned} P_{int}(s_i|s_{i-n+1}^{i-1}) &= \frac{C(s_{i-n+1}^{i}) - D}{\sum_{s_i} C(s_{i-n+1}^{i})} \\ &+ \alpha(s_{i-n+1}^{i-1}) P_{int}(s_i|s_{i-n+2}^{i-1}) \end{aligned} \tag{7}$$

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

## 4.6 Implementation of SMM

While our SMM-based prediction approach is inspired from natural language modeling, there are unique differences in our implementation stemming from the different characteristics of computational flow in computing systems. First, typically, the natural language model is built from a previously collected existing corpus once, and the same model is used for speech recognition or information retrieval without any model update. Even though the dialog or topic context information could be used to further improve the prediction of the words to be uttered at any point in the text/conversation, the information contained in the existing corpus dominates the probability of which word is to be expected next. Second, in natural language a sentence contains a finite number of words. However, in workload behavior prediction, 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 is of interest to the SMM predictor keeps coming from the beginning 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.

We can estimate the computational complexity of the SMM model training. For that, we need to know three parameters: i) number of quantization bins, ii) length of finite sequence ($n$), iii) how often we need to update the model. 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 bounded by $\mathcal{O}(kVn)$ division and multiplication operations, where $k$ is a small constant ($k \leq 3$). The storage requirements could be a more important concern for large $n$ value and very long metric sequences.

As the metric data becomes longer the SMM model size will increase as well. However, this is not a concern from a practical implementation point of view, as there are a number of effective methods that can keep the model size under control without compromising the model performance [2]. Moreover, the model size can inherently be controlled by limiting the number of quantization bins and the model order. Fewer quantization bins and smaller model order will lead to smaller model size.

Based on the described computational upper bounds for the model update, the overall computational overhead of the SMM is constrained to be less than a thousand multiply and divide operations. This translates to compute time overheads on the order of microseconds. Therefore, the proposed schemes can be implemented within the operating system software in context switch time granularities with no visible performance impact.

### 4.7 Advantages of SMM

By expressing various short term and long term sequential phenomena in terms of simple parameters in a statistical model, SMM provides an easy way to deal with complex computer metric prediction.

One of the important features of the SMM in comparison to the previous methods is its ability to model long term patterns. Last value predictor, table based predictor and others attempt to use short term dependencies, focusing on the previous $n - 1$ 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.

Another major advantage of the SMM is its ability to model patterns of variable length. We show in Figure 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 a given history, the predicted next sample is selected based on its probability.

## 5 Methodology

We perform the experiments on an IBM POWER4 [3] server platform with the AIX5L operating system. The results present per-thread behavior running in multi-user mode on a lightly loaded machine. We use the hardware performance counters, with a sampling tool that works on top of the AIX Performance Monitoring API (PMAPI). The sampler ties together counter values to a particular thread, including all library calls and system calls performed by that thread. The prediction is performed at 10 ms time scales. All the experiments are performed with the SPECCPU 2000 suite using reference datasets. All benchmarks are compiled with XLC and XLF90 compilers with the base compiler flags. The data in the experimental sections shows prediction performance with the instruction per cycle (IPC) metric. However, the same prediction approach can be applied to other architectural metrics. The

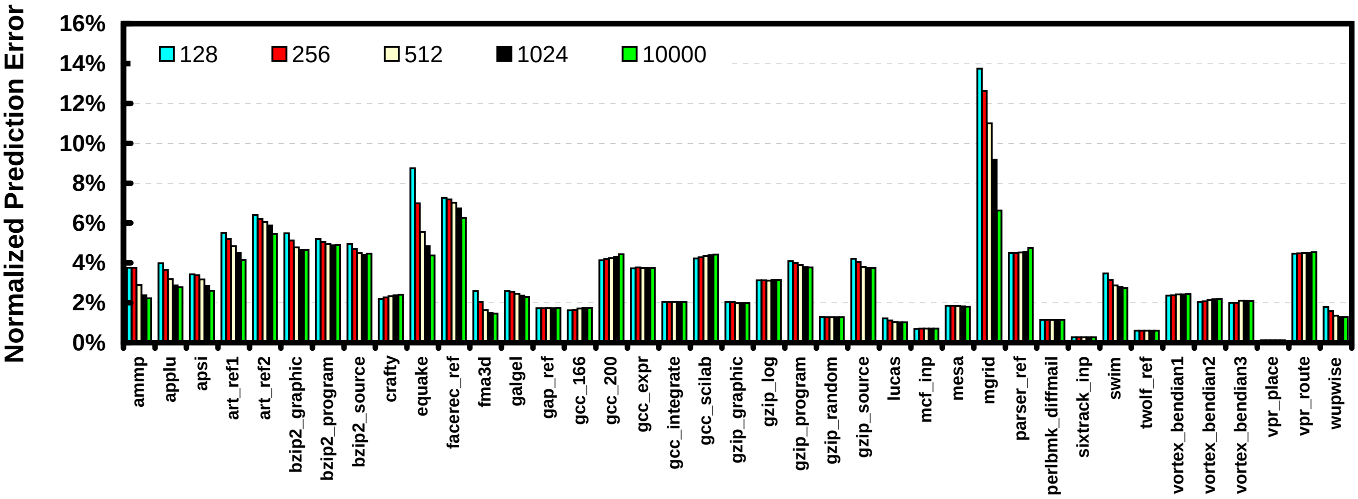

Figure 4: Normalized mean prediction error for IPC using table based predictor with various table sizes of 128, 256, 512, 1024, 10000.

results shown are based on a quantization level of 20 bins. This represents the high end of the experimented phase granularities, limiting within-bin quantization error range to 5%.

## 6 Experimental Results

We run a rigorous and extensive set of experiments to compare SMM, table based and last value predictors. We also investigate SMM predictor in-depth for its performance in relation to various parameters. Even though SMM predictor is applied to predict IPC, it has general applicability to predict any metric of interest with great accuracy.

Much of the past effort in the metric prediction area focused on ad-hoc methods that are tailored to work on specific benchmarks. Typically, these methods may perform very well for a few benchmarks in comparison to last value predictor. However for the rest of the benchmarks, they do not provide better prediction accuracy than the last value predictor. As such, they lack the generality and fail to work on other metrics, simply because they are not based on statistical estimation principles.

In order to provide a fair comparison with the baseline table based predictor, we perform a sensitivity analysis using different table sizes. Figure 4 presents the normalized mean prediction errors for different table sizes. Our results confirm the previous studies [17] in that using table sizes larger than 1024 does not provide significantly different prediction results. For most of the benchmarks as the table size increases the reductions in prediction error level off. A very large table size such as 10K does not change the results most applications. There is an appreciable improvement only for `mgrid` benchmark, while others such as `parser`, `gcc_166` and `crafty` show slight degradations in accuracy. In our experiments table size is set to 1024.

## 6.1 Metric Variation

The value of a good prediction scheme can be realized only if the underlying predicted workload characteristics show significant variability. In other words, if the metrics and the workloads of interest, show little or no variation in their temporal behavior, it is an overkill to design anything beyond a trivial predictor such as last value predictor. However, for benchmarks that exhibit rapidly changing characteristics, such a trivial predictor shall perform unfavorably. Therefore, we provide an evaluation on the predictability of the different applications used in this work.

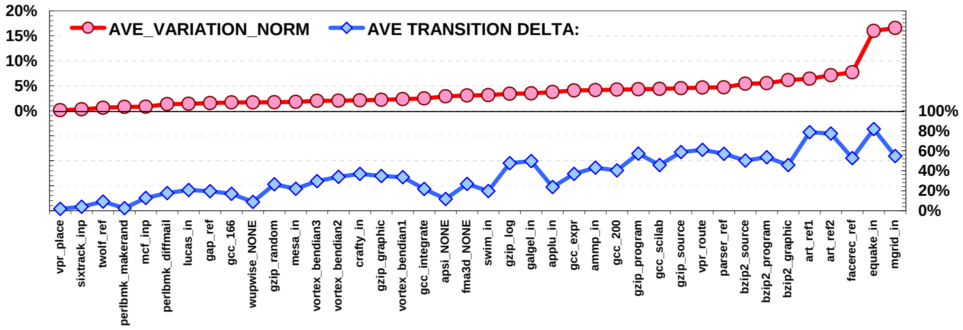

Figure 5: Variation in experimented benchmarks.

Figure 5 shows the available variation in the experimented benchmarks under two criteria. The upper plot shows the average *sample-to-sample* variation in IPC metrics. This variation is computed as the average sample-to-sample variation normalized with respect to the overall dynamic range of the workload. Benchmarks are sorted in increasing variability. Therefore, workloads towards the right end of the plot show the highest variability. The last six workloads show more than 5% sample-to-sample variation. In the lower plot, the magnitude of variation is decoupled from the *occurrence* of a variation. In this case, the IPC trace is quantized into 20 bins, and the percentage values show how often the two consecutive samples belong to different bins. This measure is useful to understand the deviation of each workload from a purely *flat* workload.

## 6.2 Metric Tracking Using SMM Predictor

The optimal model order $n$ depends on the specific benchmark and available data to train the SMM. In this study, we set $n = 8$, simply because the table based predictor, which is considered as one of the baseline methods uses a sequence length of 8. We also use the last value predictor both as a baseline to compare and as a backoff predictor for the table based predictor. 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

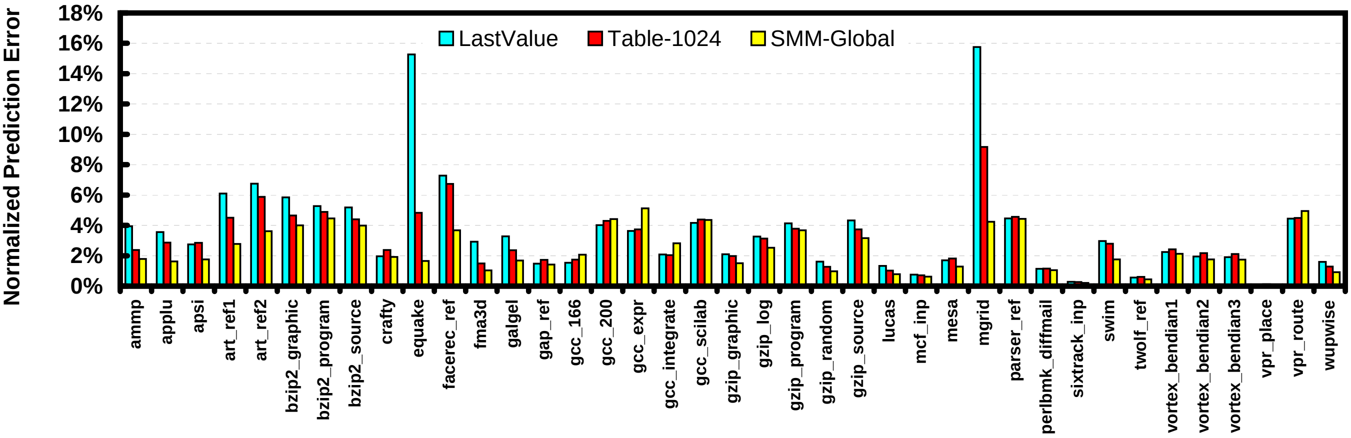

Figure 6: Improvements with the SMM predictor, compared to last-value and table-based predictors.

predictive power to model any patterns embedded in the metric sequence. We believe that $n = 8$ is a reasonable compromise between these two competing goals. In later sections, we also look at this tradeoff for different values of $n$.

Figure 6 shows a comparison of last value, table based and SMM predictors across all benchmarks. The SMM predictor improves prediction errors by 19% relative on average over all the experimented workloads compared to the table based predictor. This improvement is even further emphasized, with 43% relative reduction in prediction error for the top five highly-varying applications, namely as `mgrid`, `equake`, `facerec`, `art_ref1`, `art_ref2`. In comparison to the last value predictor, SMM improves prediction errors on top five highly-varying applications by 63%. The largest improvements of about 10–fold (15.3% vs. 1.6%) and 3–fold (4.8% vs. 1.6%) are achieved compared to the last value and table based predictors, respectively on the `equake` benchmark.

Figure 7 shows how the prediction performance of the different prediction approaches 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 towards 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

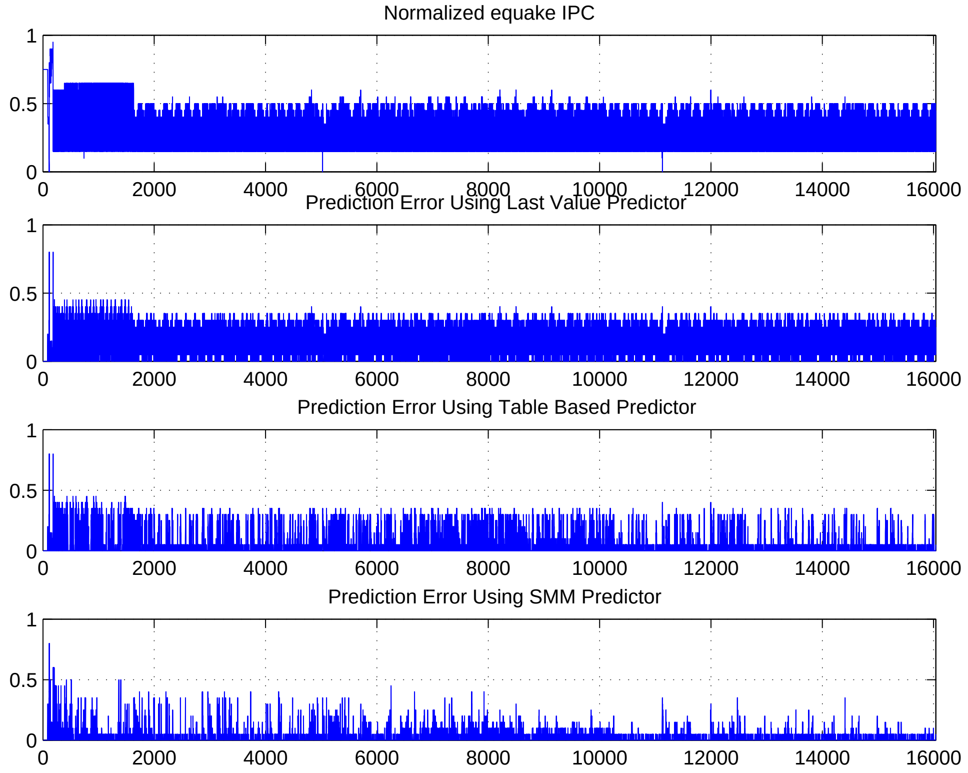

Figure 7: Normalized IPC prediction errors for different predictors over the quantized samples.

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 of the SMM predictor, as the SMM predictor can successfully leverage both long-term and varying-duration application characteristics.

We also zoom into a segment of the `equake` data to provide more concrete insights on how different predictors are performing. In the first panel of Figure 8, we plot a segment of the normalized `equake` IPC data. 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, where the errors are almost on the same scale as the data samples. In the third panel table-based predictor errors are displayed. 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.

Figure 9 underlines one of the main strengths of SMM as compared to the table based predictor. That is, SMM backs off to lower order models when the maximum sequence match is not found. In the figure, we highlight two patterns with boxes, which contain identical samples of length 6. Even though, the first 5 samples are the same

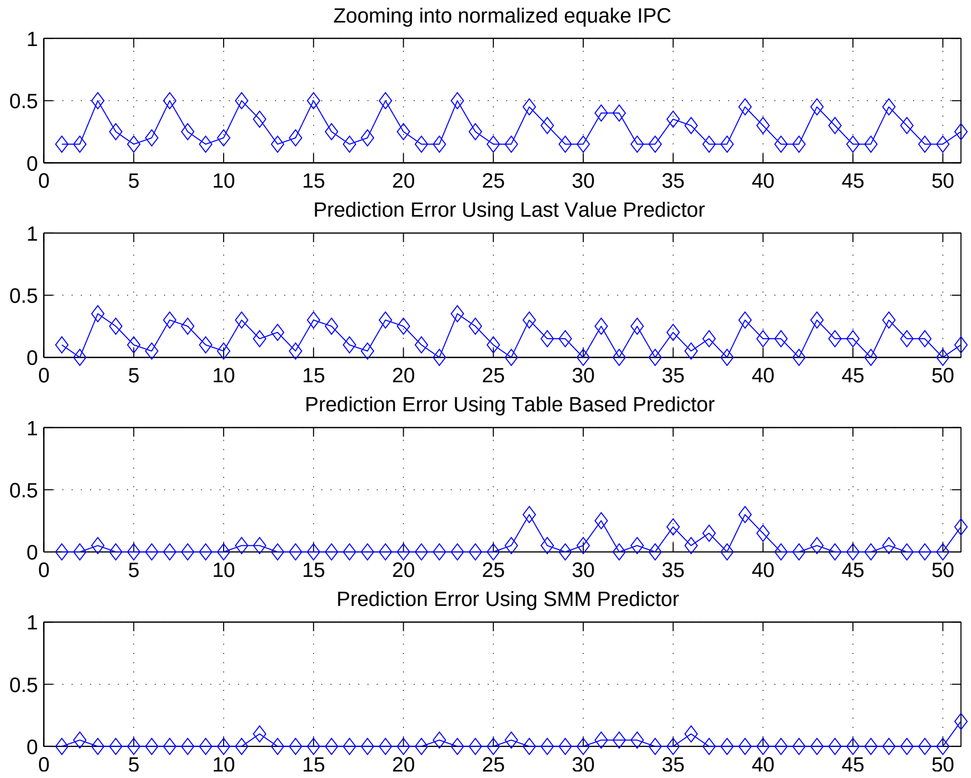

Figure 8: Normalized IPC prediction errors for different predictors over a segment of quantized samples.

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 good 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 prediction in the second box. This is of course the case, given the past 5 samples in the box, the next sample 10 has the highest probability among all the possible outputs (1,...,20). SMM prediction considers the conditional (on the history) probability of the each possible metric output and picks the one that has the highest probability.

The other main strength of the SMM predictor is its ability to model long range patterns, which improves the prediction. In order to underline this feature of SMM, we extend the runtime duration of the experimented benchmarks by a factor of 2 (shown as x2 in Figure 10) by concatenating workloads back to back. With this approach we provide the predictors with runtime histories which are twice as long as the original runs. In Figure 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 metric sample size. SMM, on the other hand, provides

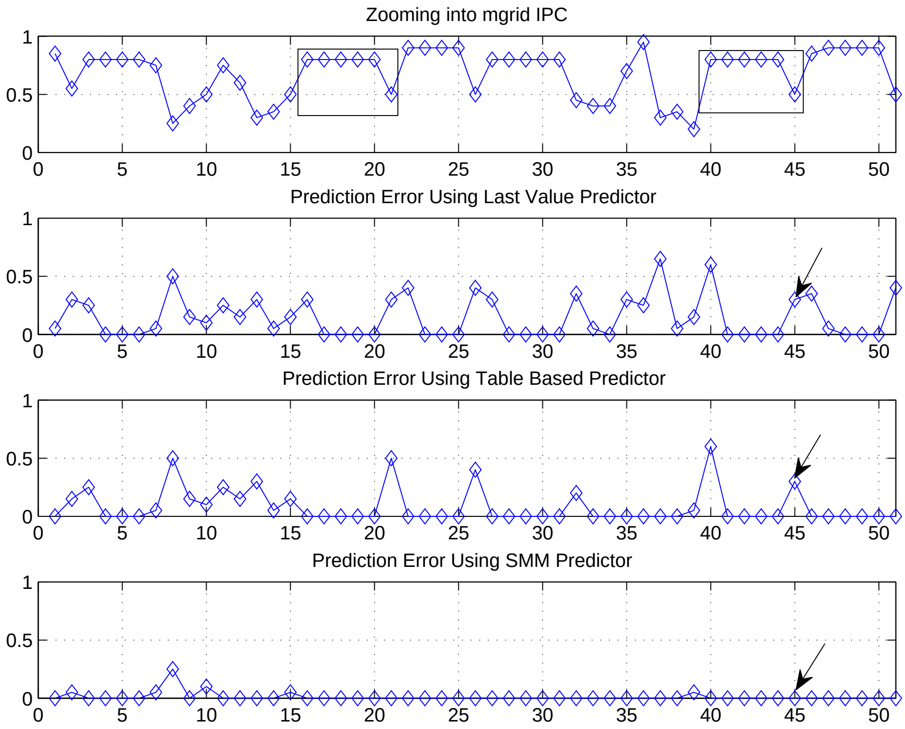

Figure 9: Benefit of backing off to lower order models for SMM.

significant improvements for almost all the benchmarks. The only exception is `swim`, where we observe a slight increase in prediction errors of both 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 characteristics before. SMM starts to improve the prediction in the second half of the duplicated benchmark data. The average improvement across all benchmarks is 15% for the SMM predictor, whereas it is only 3% for the table based predictor. This shows the substantial benefits of the SMM predictor in the longer term.

It is important to emphasize that the ability of the SMM predictor to capture long-term patterns is not directly proportional to its chosen model order/history size, or its quantization bins. The key strength of the SMM predictor stems from a fundamental difference from the existing predictors. While pattern-based or statistical predictors rely on recent history and discard old pattern information in favor of new observations, SMM actually eliminates this temporal dependence by relying on its probability-based history information. Thus, older but dominant patterns survive in SMM model, and provide significant influence in future predictions. SMM brings out two unique features: i) it can model patterns of varying length, ii) it can model long range patterns by tracking observed patterns with their respective probabilities. When capacity requirements dictate discarding some patterns, this pruning is done not by temporal proximity, but with respect to the strength of the patterns as indicated in the SMM pattern tables.

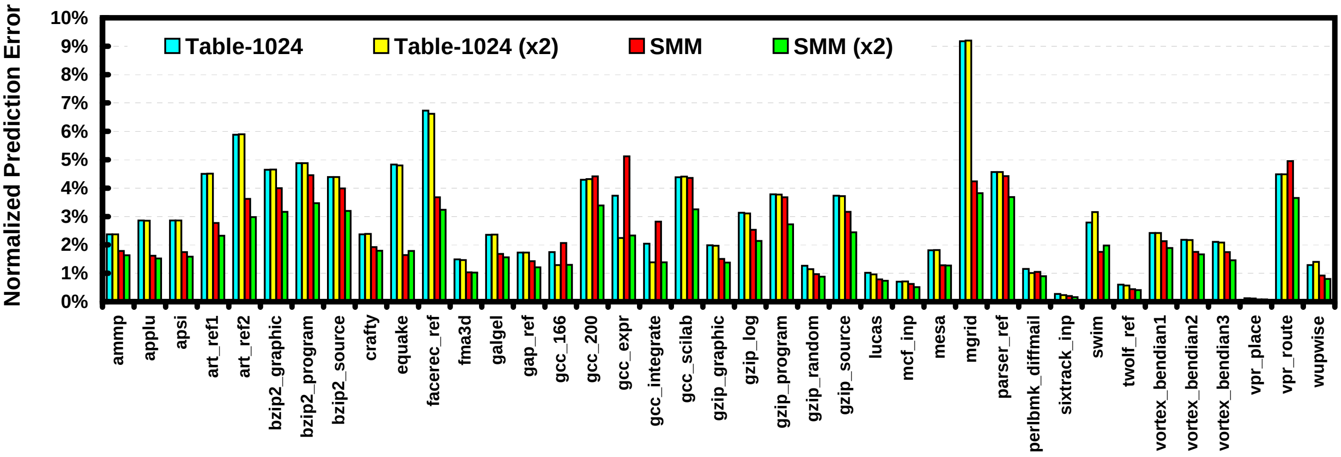

Figure 10: Improvement in SMM predictor accuracy with longer benchmark execution times. In "(x2)" experiments, each application is ran twice. SMM predictor improves its prediction accuracy significantly as it gathers more information on application behavior. Table-based approaches show practically no improvement over time.

This leads to a much higher accuracy in predicting future behavior.

## 6.3 Impact of Model Order on SMM Predictor Performance

For SMM, model order ($n$) and the history size are related via $HistorySize = n - 1$. We mentioned that the performance of SMM predictor largely depends on the model order 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—for days or even months—during system uptime, with a very large set of running metric samples. In that case using larger model order should only benefit the prediction performance.

In this section, we try different model orders and observe their effect on the prediction performance. Each benchmark can be described better with a specific model order and may achieve the smallest prediction accuracy. In Figure 11, we provide prediction errors for all benchmarks with model orders: {4,6,8}. In general, as the model order increases, overall prediction accuracy also increases (i.e. lower prediction error). Typically, models with larger model order are good in describing the fine structure in the segment of the data that is observed so far. 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 (n=4) for parser_ref achieves a slightly lower prediction error than model order 8 (n=8). We also observe that the improvements with the larger model order appears to be leveling off for $n \geq 6$ for most of the benchmarks with the exception of `mgrid`,

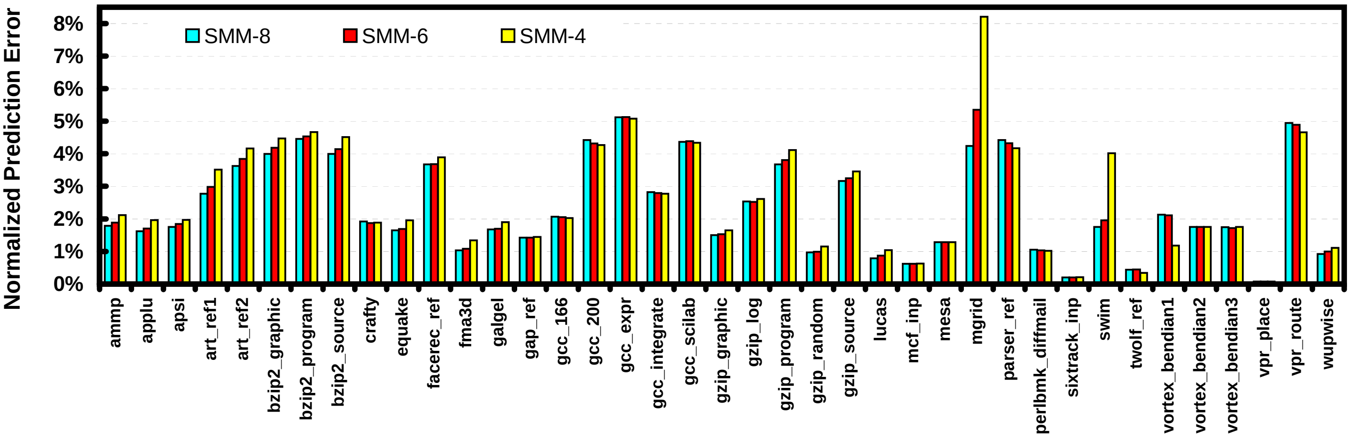

Figure 11: SMM prediction performance for different model orders of 4, 6, and 8.

where there is further possible reduction in prediction error with larger model sizes.

# 7 Conclusions

We describe a new method called Statistical Metric Model (SMM) for predicting dynamically-varying program behavior. SMM is a probabilistic model that learns application characteristics at runtime and captures long term, dominant application behavior. There are four main strengths of SMM compared to existing predictors. First, it models long term global patterns in application behavior. Second, the predictor can respond to variable-length patterns. Third, 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 of the SMM predictor, 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 respectively, with an average improvement of more than 60% and 40% for highly varying benchmarks.

# References

[1] 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. In *Computational Linguistics*, 18(1):31–40, March 1992.

[2] J. Goodman. Language Model Size Reduction by Pruning and Clustering. In *Inter. Conference on Spoken Language Processing*, Beijing China, 2000.

[3] J.M. Tendler, J.S. Dodson, S. Fields, H. Le and B. Sinharoy. POWER4 system microarchitecture. In *IBM Journal of Res. and Dev.*, 46(1), 2002.

[4] F. Jelinek and R.L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Pattern Recognition in Practice, E. S. Gelsema and L. N. Kanal, Eds. Amsterdam: North-Holland,* 1980.

[5] 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. *IEEE Computer*, 36(12):43–51, 2003.

[6] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proc. of the 32nd Inter. Symposium on Computer Architecture (ISCA-32)*, 2005.

[7] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Inter. Symposium on Microarchitecture*, pages 245–257, 2000.

[8] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-Driven Energy Accounting for Dynamic Thermal Management. In *Proc. of the Workshop on Compilers and Operating Systems for Low Power (COLP'03), New Orleans*, Sept. 2003.

[9] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *Proc. of the 2005 Inter. Symposium on Low Power Electronics and Design (ISLPED)*, 2005.

[10] K. Choi, R. Soma, and M. Pedram. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proc. of Inter. Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2004.

[11] A. Dhodapkar and J. Smith. Managing multi-configurable hardware via dynamic working set analysis. In 29th Annual Inter. Symposium on Computer Architecture, 2002.

[12] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and Predicting Program Behavior and its Variability. In *IEEE PACT*, pages 220–231, 2003.

[13] M. Huang, J. Renau, and J. Torrellas. Positional Adaptation of Processors: Application to Energy Reduction. In *Proc. of the Inter. Symp. on Computer Architecture*, 2003.

[14] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *Proc. of the 39th ACM/IEEE International Symposium on Microarchitecture (MICRO-39)*, 2006.

[15] C. Isci and M. Martonosi. Identifying Program Power Phase Behavior using Power Vectors. In *Proc. of the IEEE Inter. Workshop on Workload Characterization (WWC-6)*, 2003.

[16] C. Isci and M. Martonosi. Detecting Recurrent Phase Behavior under Real-System Variability. In *Proc. of the IEEE Inter. Symposium on Workload Characterization*, Oct. 2005.

[17] C. Isci, M. Martonosi, and A. Buyuktosunoglu. Long-term Workload Phases: Duration Predictions and Applications to DVFS. *IEEE Micro: Special Issue on Energy Efficient Design*, 25(5):39–51, Sep/Oct 2005.

[18] J. Lau, S. Schoenmackers, and B. Calder. Transition Phase Classification and Prediction. In *11th Inter. Symposium on High Performance Computer Architecture*, 2005.

[19] R. Sarikaya and A. Buyuktosunoglu. Predicting program behavior based on objective function minimization. In *Proc. of the IEEE Inter. Symposium on Workload Characterization (IISWC)*, Sept. 2007.

[20] X. Shen, Y. Zhong, and C. Ding. Locality Phase Prediction. In *11th Inter. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI)*, Oct. 2004.

[21] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In 10th Inter. Conference on Architectural Support for Programming Languages and Operating Systems, Oct 2002.

[22] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proc. of the 28th International Symposium on Computer Architecture (ISCA-30)*, June 2003.

[23] A. Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proc. of the Inter. Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002), Grenoble, France,*, Aug. 2002.

[24] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar. Dynamic Tracking of Page Miss Ratio Curve for Memory Management. In *Proc. of the 11th Inter. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI)*, 2004.

[25] S. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Annual Meeting of Association for Computational Linguistics (ACL-96)*, Santa Cruz, CA, 1996.