Program Behavior Prediction Using a Statistical Metric Model

Ruhi Sarikaya
IBM Research
Yorktown Heights, NY
sarikaya@us.ibm.com

Canturk Isci
IBM Research
Hawthorne, NY
canturk@us.ibm.com

Alper Buyuktosunoglu
IBM Research
Yorktown Heights, NY
alperb@us.ibm.com

ABSTRACT

Adaptive computing systems rely on predictions of program behavior to understand and respond to the dynamically varying application characteristics. This study describes an accurate statistical workload metric modeling scheme for predicting program phases. Our evaluations demonstrate the superior performance of this predictor over existing predictors on a wide range of benchmarks. This prediction accuracy lends itself to improved power-performance trade-offs when applied to dynamic power management.

Categories and Subject Descriptors:
C.4 [Performance of Systems]: Modeling techniques

General Terms: Management, Measurement, Performance

1. INTRODUCTION

Today's adaptive computing systems heavily rely on accurate predictions of changes in application behavior to proactively manage system adaptations. Prior work proposed such predictors based on simple pattern history tables [3], recent program flow behavior [6], statistics over recent performance characteristics [1] and model based parametric predictors [5]. These predictors have shortcomings due to their deficiency in predicting global long range patterns, and their inability to model patterns of varying length. In this work, we propose a new prediction technique based on statistical metric modeling that overcomes these limitations. This predictor tracks the probabilities of variable-length metric sequences. These probabilities are used to predict the most likely future behavior given the history pattern. This predictor 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 this approach in comparison to the previously proposed predictors. We demonstrate the application of this predictor to dynamic power management, leading to better power-performance trade-offs compared to the existing approaches.

2. METRIC MODELING PREDICTOR

Our prediction approach is inspired from natural language processing, where we treat the workload metric samples as words and build a language model for each workload. Since the performance metrics are real numbers, we quantize them into a set of discrete sets, called "quantization bins". Our intuition is that, like in natural languages, we can treat modeling workload behavior as a language modeling problem. There is commonly an underlying structure in workload execution as also reflected in performance metrics, and our modeling approach can reveal this structure.

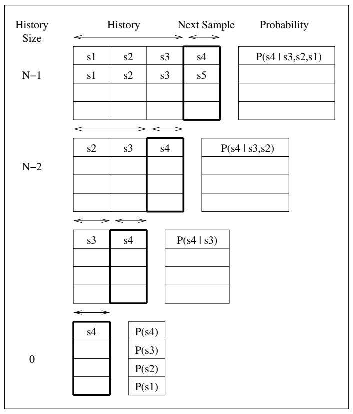

Figure 1: Model with back-off for $n = 4$.

The statistical metric model is a conditional distribution on the identity of the ith sample in a metric sequence, given the identities of all previous samples. We 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. 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. Our metric model is based on a class of Markov models, which is known as, the n-gram models [4]. Here n refers to the maximum length of the finite sequence of the metric samples. The probability of the nth sample is conditioned on the previous $n-1$ samples.

The metric model of order $n = 4$ is shown in Figure 1. The model has two sets 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 models of lower order m $(1 \leq m \leq n)$, increase the likelihood of finding a matching subsequence for a given finite sequence.

The 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 during workload execution.

$$P(s) = \prod_{i=1}^{l} P(s_i|s_{i-1}, s_{i-2}, ..., s_1) \qquad (1)$$

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,

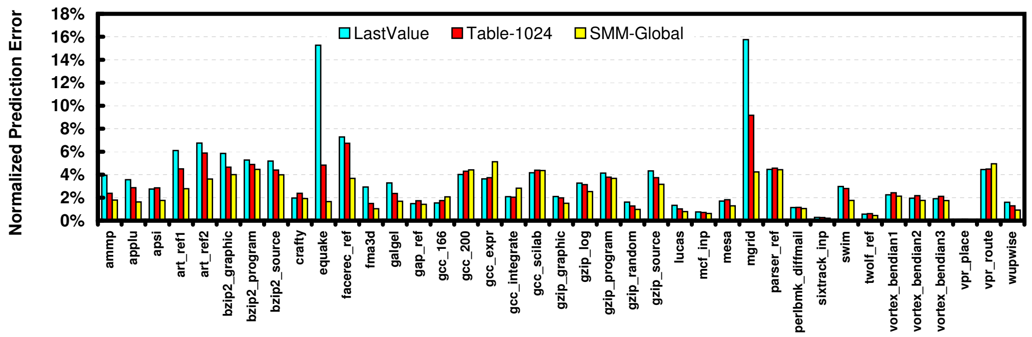

Figure 2: Prediction accuracy of our predictor, last-value and table-based predictors.

we can express the probability of a s, $P(s)$ as:

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

where s_i^j denotes samples $s_i, ..., s_j$. In order to simplify the description and formulation of the the metric model, 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:

$$P(s) = \prod_{i=1}^{l} P(s_i|s_{i-1}), \text{where } P(s_i|s_{i-1}) = \frac{C(s_{i-1}, s_i)}{C(s_{i-1})} \quad (3)$$

where $C(x)$ denotes the number of times the sequence x occurs in the metric. This is called the maximum likelihood (ML) estimate for $P(s_i|s_{i-1})$. In addition to ML probability estimates, we also apply additional "smoothing" to the relative frequencies to modify the conditional distributions away from pure relative frequency estimates in order to compensate for data sparsity.

3. EXPERIMENTAL EVALUATION

We evaluate our predictor with the SPEC CPU2000 suite running on an IBM POWER4 system. We use hardware performance counters to collect data for prediction. Our predictor performs predictions at 10 ms time scales. The overall computational overhead of our predictor is constrained to be less than a thousand multiply and divide operations. This translates to compute time overheads on the order of microseconds. Thus, the proposed scheme can be implemented within the operating system software in context switch time granularities with negligible performance impact.

We run an extensive set of experiments to evaluate our predictor in comparison to two existing prediction schemes, namely table-based [2] and last-value predictors. The table-based predictor tracks a fixed length of most recently observed characteristics. The predictions for future behavior is deterministically encoded into a table, based on prior observed patterns. Last-value predictor assumes that the next metric sample is the same as the last observed sample.

Figure 2 shows a comparison of last value, table based and our predictor for a model order $n = 8$ across all SPEC CPU2000 benchmarks. Our metric modeling based predictor improves prediction accuracy by 19% and 36% compared to the table based and last-value predictors. This improvement is even further emphasized for highly-varying applications, with 43% and 63% relative reductions in prediction errors over the existing approaches.

Last, we explore the application of our prediction approach to dynamic power management. We predict memory access rates of applications at runtime using metric modeling, and use the predicted memory access behavior to control dynamic voltage and frequency scaling (DVFS). We use memory access rates as the main differentiator for categorizing execution into different bins. In our evaluation system, we consider eight bins, corresponding to eight DVFS states. That is, when a predictor predicts the next application phase as bin_i the processor is proactively set to DVFS setting i. Similar to the prior experiment, we continuously predict application behavior at fixed sampling intervals. Based on the predicted execution behavior, i.e., memory access rate, we set the corresponding DVFS state for the following period. Then, at the end of this period we observe the achieved power savings and the associated performance degradation based on the actual execution behavior in this past interval. While our predictor achieves slightly lower power savings compared to the other predictors, it significantly reduces the performance degradation impact of such proactive power management, with 34% and 19% improvements compared to the last-value and the table-based predictors respectively.

4. CONCLUSIONS

We described a predictor based on a probabilistic model that learns application characteristics at runtime and captures long term, dominant application behavior. The proposed approach has four main strengths. First, it models long term global patterns in application behavior. Second, the predictor can accommodate variable-length patterns. Third, it is resilient to small fluctuations in the observed patterns. Last, the metric model has the ability to adapt itself; as it learns more it predicts better. The results show that the our 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. We also show the application of this prediction to dynamic power management, where higher prediction accuracies of our approach lends itself to superior power-performance trade-offs compared to the existing approaches.

5. REFERENCES

[1] E. Duesterwald, *et al.* Characterizing and Predicting Program Behavior and its Variability. *PACT*, 2003.

[2] C. Isci, *et al.* Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. *MICRO*, 2006.

[3] C. Isci, *et al.* Long-term Workload Phases: Duration Predictions and Applications to DVFS. *IEEE Micro*, 2005.

[4] F. Jelinek *et al.* Interpolated Estimation of Markov Source Parameters from Sparse Data *Pattern Recognition in Practice, E.S. Gelsema and L. N. Kanal* , 1980.

[5] R. Sarikaya, *et al.* A Unified Prediction Method for Predicting Program Behavior. *IEEE Trans. on Computers*, 2010.

[6] T. Sherwood, *et al.* Phase Tracking and Prediction *International Symposium on Computer Architecture*, 2003.