Adaptive Sampling of Performance Counters

David Carrillo-Cisneros
University of California, Irvine
dcarrill@ics.uci.edu
Isaac D. Scherson
University of California, Irvine
isaac@ics.uci.edu

Abstract—Many applications of profiling based on sampling of Performance Counters (PC), such as feedback-directed optimization and software reliability, are often constrained by the amount of information that can be obtained without perturbing significantly the behavior of the profiled task. Current implementation of event and time based sampling software utilize fixed or random sampling periods which are unresponsive to changes on the frequency of occurrence of architectural events. In this paper we introduce an adaptive sampling schema for event-based sampling of PCs that adaptively varies the sampling period in order to minimize the reconstruction error and reduce the number of sample points taken. By restricting the number of samples taken, AES prevents oversampling and overload to the profiled task. We conducted experiments within the perf_event kernel module and have observed a decreased reconstruction error (MSE) with respect with the ubicuous fixed-period sampling. An additional objective of this work is to motivate the formal study of the properties of sampling Hardware Performance Counters, an optimization opportunity that has rarely been explored up to now within this context.

I. INTRODUCTION

Most modern microprocessors include Performance Monitoring Units (PMUs)[4, 6] that allow event-based sampling [9] of its performance counters. Sampling of PMUs has significantly lower overhead than instrumentation or software based sampling[10] and it does not require software modifications or binary recompilation.

Let \( s_i = (t_i, c_{t_i}) \) be the \( i \)-th sample-point taken from a performance counter at time \( t_i \) and with an observed value \( c_{t_i} \). Then:

\[
\begin{align*}
  k_i &= c_{t_i} - c_{t_{i-1}}, \quad n_i = t_i - t_{i-1} \quad \text{and} \quad m_i = k_i/n_i \quad (1)
\end{align*}
\]

as the increments between samples \( s_{i-1} \) and \( s_i \) for event’s count and time, respectively. Since all HPC store non-negative integers, \( 0 \leq k_i, 0 \leq n_i, 0 \leq m_i \) for all \( i \). Each sample point can be seen as a point in an integer lattice, as pictured in Figure 1.

A. Inherent problems of profiling sampling

1) Perturbation: When a sample is taken, the execution sampling routine pollutes the cache and memory subsystem and incurs in significant overhead[11]. Some lightweight implementations[5] have been proposed that alleviate but not solve the problem. The overhead increases as the sampling frequency increases and if its high enough, the behavior of the event counted is greatly modified by the sampling mechanism.

2) Reconstruction Uncertainty: The process of estimating the values taken by the HPC between the sampled time points is called reconstruction. Previous work have experimentally compared different reconstruction approaches[8]. Software implementations such as Linux’s perf_event[3] and PAPI [2] perform linear interpolation between the sampled values as reconstruction technique.

Given two sample points, \( s_{i-1} \) and \( s_i \), the linearly interpolated value for the count at time \( t \) is given by:

\[
R(t | s_{i-1}, s_i) = \left[ m_i(t - t_{i-1}) + c_{t_{i-1}} \right] \quad (2)
\]

For any profiled task, each possible time evolution of the event counts follows a staircase walk[Weinstein2002] in an integer lattice between the sample points \( s_{i-1} \) and \( s_i \). The \( j \)-th staircase walk is denoted by the tuple of non-negative integers:

\[
W_j(s_{i-1}, s_i) = (\delta_{t_{i-1}}^j, \ldots, \delta_{t_i}^j) \quad (3)
\]

The maximum absolute difference between any of the possible staircase walk and the reconstructed value is the maximum reconstruction error and is given by:

\[
U(s_{i-1}, s_i) = \max_j \left| \sum_{t=t_{i-1}}^{t_i} \delta_t^j - \left( R(t | s_{i-1}, s_i) - c_{t_{i-1}} \right) \right|
\]

\[
= \sum_{t=0}^{n_i} k_t - m_t \quad t
\]

Ignoring rounding errors:

\[
U(s_{i-1}, s_i) \approx \int_0^{n_i} (k_t - m_t x) \, dx = \frac{k_t n_t}{2} \quad (4)
\]

B. Disadvantages of Fixed-period sampling

All the software libraries that the authors have surveyed, including Linux’s perf_event[3], Perfmon2[7], Limit[5] and PAPI[2], implement a fixed-period (and/or fixed frequency) sampling schema. This can lead to the following scenarios:

- Oversampling: High frequency of occurrence of event implies excessively high sampling frequency.
- Undersampling: When few events occur in a large time, there are large periods of the application that go unobserved.

Oversampling is specially problematic since the overflow handler may be called so often that a significant amount of the available CPU time is spent in the sampling routine and perturb the task as discussed in section I-A1. Restricting the maximum sampling period, as done in perf_event, may cause undersampling instead. Low overhead overflow handlers [5] reduce but do not solve the problem.

II. ADAPTIVE EVENT-BASED SAMPLING

Our goal is to find \( q \) sampling periods, \( k_1, \ldots, k_q \), while taking samples at an average sampling period of \( p \) such that: (1) Reduces reconstruction uncertainty. (2) Restricts sampling perturbation.

A. Loss Function

The maximum reconstruction error is given by equation \( 4 \). The loss due to reconstruction is given by:

\[
L_R = \sum_{t=1}^{q} k_t n_t / q
\]
A regularization term that penalizes large differences of $k_1$ from the desired sampling period $p$, in order to prevent the optimization procedure to find extreme values, is given by:

$$L_U = (k_1 - p)^2$$

The total loss function is: $L = L_R + L_U$. The loss function can be simplified by realizing that an average sampling period of $p$ implies $p = \sum_{i=1}^{\infty} k_i / q$ and restricting to the case $q = 2$ (without loss of generality). The resulting loss function is:

$$L = (k_1 - p)^2 + \frac{k_1 n_1 + k_2 n_2}{2} \quad (5)$$

Information of the behavior of the sampled function is required for an adaptive sampling schema to work. We assume the existence of:

- $\tilde{n}_1$: Predictor of slope for the interval of the next sample.
- $\hat{n}_2$: Predictor of the overall slope for all $q$ samples.

In practice, recent history can be used as predictor in regions with high locality.

Substituting $\tilde{n}_1 = \frac{k_1}{n_1}$, $\hat{n} = (k_1 + k_2)/(n_1 + n_2)$ and minimizing we obtain the optimal sampling period:

$$k_1^* = \arg\min_{k_1} L = \frac{(\hat{n} + \tilde{n}_1 + 2 \hat{n} \tilde{n}_1)p}{2 \hat{n}(1 + \tilde{n}_1)} \quad (6)$$

Figure 2 illustrates the proposed adaptive sample schema for a function with high variability in its event occurrence rate.

III. EVALUATION AND RESULTS

We implemented the proposed sampling schema within perf events[3]. We executed each benchmark in the suite PARSEC[1] repeatedly using different periods for fixed-period and adaptive sampling. Figure 3 show the difference in average reconstruction error between fixed and adaptive sampling for the ferret and blackscholes benchmarks. We observed an improvement in most benchmark, with the notable exception of bodytrack benchmark. Further investigation of this benchmark lead us to attribute the poor performance of our algorithm to low locality, which is required for the slope predictor to work adequately. A weakness of the proposed schema is that it depends in the quality of the slope predictors. The performance of the algorithm is highly affected the state of the system and frequency scaling and current work is aimed to control for this effects.

A. Future Work

This work is an initial step towards a formal study of the sampling of performance counters. An immediate extension would aim to reduce uncertainty when sampling multiple performance counters within the same sampling event (group sampling). Another future step is to incorporate information from the profiled task in order to improve the quality of the slope predictors. Information extracted from the source code, or history of previous sampling schemes can be utilized for this purpose.

REFERENCES