Trace Sampling for Desktop Applications on Windows NT

Patrick Crowley and Jean-Loup Baer
Department of Computer Science & Engineering
Box 352350
University of Washington
Seattle, WA 98195-2350
{pcrowley, baer}@cs.washington.edu

Abstract

This paper examines trace sampling for a suite of desktop application traces on Windows NT. This paper makes two contributions: we compare the accuracy of several sampling techniques to determine cache miss rates for these workloads, and we present a victim cache architecture study that demonstrates that sampling can be used to drive such studies. Of the sampling techniques used for the cache miss ratio determinations, stitch*, which assumes that the state of the cache at the beginning of a sample is the same as the state at the end of the previous sample, is the most effective for these workloads. This technique is more accurate than the others and is reliable for caches up to 64KB in size.*

1 Introduction

Trace-driven simulation is a common approach for evaluating memory systems. Trace-driven simulations demand large amounts of space and time, particularly for large caches and long running applications. These demands can be greatly reduced by employing sampling techniques at the expense of providing only a statistical estimate of the properties of a full trace. Previous studies contain results for various workloads and caches [Martonosi et al. 93, Laha et al. 88, Kessler et al. 94] and discuss the conditions under which sampling may, or may not, be used.

Our interest is in the behavior of commonly used desktop applications. When compared to benchmarks such as SPEC95, these applications have larger working sets, are feature rich, and, of course, can run for billions and billions of instructions. Hence, traces based on exhaustive or elaborate executions of such applications will be extremely large. In this work, we consider the usefulness and limitations of trace sampling for a suite of five publicly available desktop application traces for Windows NT on the Intel X86 platform.

In order to determine the feasibility of sampling for these application traces, we present cache miss rate results for a large set of cache sizes and sampling techniques. Beyond the statistical assumptions necessary for sampling, other assumptions are made that permit the assignment of confidence, in the statistical sense, to the results. Our results are derived from traces of limited size (from 0.1 billion to 1.5 billion references) so that we can compare sampling results to "true" results. We have also performed sampling experiments on much longer traces and, in general, they show the same trends as those based on a smaller number of samples.

In order to demonstrate the utility of these sampling techniques for architectural studies, we simulate the use of victim caches. This experiment shows that trends may be observed with sampling, thus allowing us to test a wide range of architectural parameters in a relatively short amount of time.

The remainder of the paper is organized as follows. Section 2 briefly reviews the details of trace sampling and previously published results. Section 3 introduces the benchmarks used in this study, presents our methodology, and concludes with a selection of cache miss rate results [1] and discussion. The sample architectural study is given in Section 4. The paper concludes with a discussion in section 5.

This work was supported in part by NSF Grant MIP-9700970 and by a gift from Intel Corporation.

[1] Complete results will be available in a forthcoming Technical Report.

2 Trace sampling

In trace sampling an observation, or *sample*, is obtained by recording a fixed number, the *sample size*, of consecutive references from a reference stream[2]. Another fixed number of references are ignored before the next observation is made. The *sampling ratio* is the percentage of total references used in all the observations.

Sampling theory states that sets of random, unbiased observations from a population may be used to make inferences about that population. As described above, observations in trace sampling are not random; they are systematic since they are evenly spaced throughout the trace. This non-random pattern is not a problem, though, since systematic observations can be used to make even more precise inferences than random observations under certain circumstances [Cochran 77] (that is, when the variance of systematic observations is greater than the variance of the population). Unfortunately, however, trace sampling neither involves unbiased observations nor a sufficient alternative. The problem is that the state of the cache is unknown at the start of each observation. In other words, since portions of the trace are unexamined between observations, it is unknown whether the first reference to each cache block will be a hit or a miss. Such references are referred to as *unknown* [Wood et al. 91] or *cold-start* [Laha et al. 88] references.

A number of techniques have been employed to mitigate the bias due to unknown references. One approach is to make assumptions about, or construct, the state of the cache at the start of each sample. These assumptions may include: assuming an empty cache(i.e., assume that a complete context switch occurred between samples; hereafter denoted *cold*), assuming the state at the end of the previous sample [Agarwal et al. 88] (*stitch*), and using some number of references to prime the cache [Fu & Patel 94] (e.g., 20% of the sample, denoted *prime-20*, and 50%, denoted *half*). The efficacy of these assumptions depends on workload, cache organization, and choice of sampling parameters(i.e., sample size and sampling ratio.) If complete context switches occur in a cache between samples from a given trace, then assuming an empty cache at the start of each sample, as is the case with *cold*, will be an accurate assumption. If most misses are due to conflicts in a small number of cache lines, then *stitch* may work well since only a small portion of the working set is likely to change between samples. Priming the cache will be effective if unknown references are few relative to the sample size and are mostly included in the priming set.

Another approach is to directly determine or approximate the miss ratio of unknown references, which we denote here as μ. For example, *cold* can also be thought of as an estimator that assumes all unknown references miss. In [Laha et al. 88], unknown references are not included in the estimate of overall miss rate. That is, unknown references are used to prime the cache but are not counted as hits or misses. As noted in [Wood et al. 91], this implicitly assumes that the miss ratio for unknown references is equivalent to the miss ratio for all other references. By employing a renewal-theoretic model that depends on the percentage of time a given cache block frame is alive or dead, [Wood et al. 91] show why μ should be, and is observed to be, higher than the overall miss rate.

This model is used to estimate μ by observing the probability that a reference to a cache line occurs within a dead time (where time is measured in total references, and dead time implies that the next reference to that cache line will miss). This suggests that if a random time t has probability P of occuring within a dead time for a given cache line, then P is also the probability that an unknown reference will miss in that cache line. This probability can be measured in a full trace by observing the average live and dead time lengths for each cache line in a cache. In a sampled trace, this probability must be estimated with observations within each sample. This sampled probability is the basis for *INITMR*, the miss rate estimator described in [Kessler et al. 94] and [Wood et al. 91].

Accurately coping with unknown references is particularly important when sampling for large caches, where the number of unknown references can easily dominate the number of known misses. Very large caches typically correspond to a very small number of misses, and, hence, are inherently at odds with sampling [Kessler et al. 94]. As we will see, however, when known misses dominate unknown references, several approaches will be effective.

Application	Executable Size (MB)	Size with DLLs (MB)	# DLLs used (shared)
acrord32	2.26	9.73	34 (24)
netscape	3.17	9.95	28 (24)
photoshp	3.65	13.5	44 (25)
powerpnt	4.36	12.5	26 (21)
winword	3.78	11.2	26 (21)

Table 2. *Application object file characteristics.*

3 Benchmarks, Methodology, and Sampling Results

Benchmarks: Table 1 describes the 5 personal desktop applications that we used as benchmarks and their corresponding workloads. Table 2 presents the size of the original binaries as well as DLL usage. A comparison between the execution characteristics of these applications with those of

[2]In statistics, the term *sample* is used to denote an entire collection of observations. Like most other studies, we eschew this usage.

Application	Description	Instructions Executed (millions)
acrord32	Adobe Acrobat Reader 3.0: Reader for portable document format (PDF) files. The benchmark loads acrobat.pdf (a 277 KB file) from the standard acrobat reader distribution, and navigates through the document three different ways: through the hyperlinks in the document itself, through the forward and back button provided by acrobat reader, and through a view of the document outline provided by acrobat reader. Finally, the benchmark searches for the word "buy" in the document before closing the program.	408
netscape	Netscape Navigator 3.1 web browser. The benchmark opens four web pages: `www.cs.washington.edu`, `www.cnn.com`, `www.mtv.com`, and `www.washington.edu`. These pages were viewed on March 18, 1998. The java module for netscape was turned off because Etch (our instrumentation tool) does not handle the dynamically generated code generated by the java just-in-time compiler.	92
photoshp	Adobe Photoshop 4.0 image editing package. The benchmark loads fruit.jpg (a 591 KB still-life photograph of fruit) from the standard distribution and applies the *color pencil*, *accented edges*, *diffuse glow*, and *add noise* photo filters to the image.	1,511
powerpnt	Microsoft PowerPoint 7.0b slide preparation package. The benchmark loads in a 311 KB 18-page presentation (the presentation included five pages of graphs and six pages of figures in addition to text) in slide mode, scrolls through 3 pages, edits a figure, and continues scrolling through until the end of the document. The benchmark then goes into the outline mode and creates a new page and goes back into the slide mode to move text around. Finally, the benchmark goes into slide sorter mode and moves some slides around.	209
winword	Microsoft Word 7.0 word processor. The benchmark simulates a user typing in seven paragraphs in an eight page document (document size is 29K). The benchmark then performs four search and replace commands on the document before saving a text version of the file. The interactive spell checker was turned on.	351

Table 1. *Benchmarks used for this study. The traces of these applications were produced on a dual Pentium Pro 200 system running Windows NT Workstation 4.0 service pack 3.*

Technique	Description
true-sample	starts each observation with correct cache state
cold	assumes that the cache is empty at the beginning of each observation
prime-20	uses the first 20% of each observation to prime the cache
half	uses the first 50% of each observation to prime the cache
stitch	uses the end state of the previous observation as the initial cache state for the current observation
non-uniform	same as cold, except the observations are not evenly spaced (jittered by 20% of the sample size

Table 3. *Sampling techniques for coping with unknown references.*

the integer SPEC95 suite can be found in [Lee et al. 98].

Methodology: After experimenting with various samples sizes and sample ratios, we settled on a sample size of 500,000 references and a sampling ratio of 0.1. The process of tuning these parameters for a given workload is important [Martonosi et al. 93, Kessler et al. 94]. The rationale for our choice for these Windows NT desktop application traces can be found in the technical report.

Table 3 describes the sampling techniques considered in this study. As noted in the previous section, they differ by the state of the cache at the beginning of a sample. Two techniques not mentioned earlier are *true-sample* and *non-uniform*. *true-sample* simulates the caches over the full trace and reports the miss ratio observed over the regions that are sampled with the other techniques. It is therefore an unbiased estimator of the miss ratio for the entire trace. Its accuracy depends on how "fine-tuned" our sampling parameters, *sample size* and *sampling ratio*, are to a given cache and workload. While *true-sample* is not a practical method, it is however the basis for comparisons with the other techniques which, in addition to the same sampling errors, will have unknown references biases. *non-uniform* is similar to *cold* but uses non-uniform sampling intervals.

We consider direct mapped and 4-way set-associative instruction and data caches with sizes ranging from 8KB to 128KB and direct-mapped and 4-way set-associative unified caches with sizes ranging from 256KB to 4MB. Due to space considerations, select examples from these configurations will be presented here, but complete results will be available in the technical report. In the figures to follow, each data point is the arithmetic mean of the miss ratios observed with one sampling method over each of the samples taken from the trace. The error bars for each data point correspond to the 90% interval of confidence based on the distribution of miss ratios of the systematic samples [Cochran 77]. The actual miss ratio for the entire trace is indicated by the solid bar.

Sampling Results: Figure 1 and Figure 2, respectively, display the miss rates corresponding to the simulations of

direct-mapped instruction caches for the *acrord32* application and of 4-way set-associative data caches for *powerpnt*. These figures are representative of the complete set of simulations. The following observations can be made:

1. *true-sample* sometimes underestimates the true miss rate (cf. Figure 1) and sometimes overestimates it (cf. Figure 2). Recall that *true-sample*'s accuracy is linked to the choices of sample size and sampling ratio. By choosing a single (sample size, sampling ratio) pair for all applications, we cannot tune these parameters for each application. Note however that when *true-sample* underestimates (resp. overestimates), it does it consistently for all cache sizes for a given application. Also, in all cases, the real miss rate is within the 90% interval of confidence of *true-sample*.

2. All techniques work well, i.e., give results within the 90% interval of confidence for caches up to 32 KB. Among these techniques, *stitch* works best and gives good results for caches up to 128 KB. All techniques, except those priming the cache, tend to overestimate the *true-sample* miss ratio, i.e., they underestimate the miss ratio of the unknown references as explained in [Wood et al. 91]. With larger caches, the bias gets larger since the number of unknown references is also larger. The priming techniques have a slightly different behavior since the statistics are gathered on a smaller number of references. Nonetheless, their accuracy for caches of 32 KB and more is always inferior to that of *stitch*.

3. A general trend is that confidence intervals decrease with cache size. It is not the case though that we are more confident with the results for larger caches, rather, the miss rates are simply smaller and, hence, so are the confidence intervals. To make comparisons between confidence intervals of different cache sizes, it is necessary to consider the confidence interval *as a percentage of the miss rate.* We see here that this percentage remains roughly constant.

4. *cold* and *non-uniform* yield the same results, suggesting that completely systematic samples are sufficient and there is no need to inject randomness in intervals between samples.

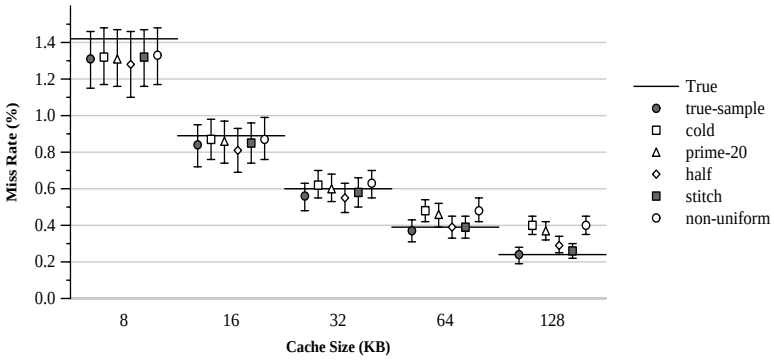
Figure 3 depicts the results based on the *winword* trace for large, direct-mapped combined data and instruction caches. In this case, we see that the errors due to the choice of the sampling parameters are very small: *true-sample* is highly accurate. However, the bias due to unknown references is extremely high for all techniques except *stitch*. What we had seen for the larger caches in Figure 1 becomes even more pronounced. If one were satisfied with a 95% rather than a 90% confidence interval, *stitch* might be sufficient.

Long Trace Sampling Results: For the previous results we have used traces of limited size (from 0.1 billion to 1.5 billion references) in order to compare sampling results to "true" results. We have constructed a set of longer traces (based on billions of references) of the same suite of applications to test a greater range of features more exhaustively and to determine whether it is necessary or instructive to use larger workloads to drive these applications. Figure 4 gives results for the *stitch* sampling technique based on a "long" trace of *netscape*. This trace contains samples amounting to 10% of the original reference stream which contained approximately 2.5 billion instruction and data references. Figure 4 compares the true, *true-sample*, and *stitch* miss rates for the original short trace to the sampled *stitch* miss rates of the larger trace. For *netscape* and these data caches, the differences are not dramatic. This suggests that while the longer trace contains more instructions corresponding to more features, they are not substantially different, with respect to cache miss rate, from the features represented in the shorter trace. The full set of results for long traces and cache configurations contains cases where differences are either slightly more or slightly less significant that those depicted here. In all cases, however, the short and long trace sampling results are similar enough to indicate the same results (i.e., trends) within the context of architectural studies as presented in the next section.

4 An Architectural Study: Trends in Victim Caches

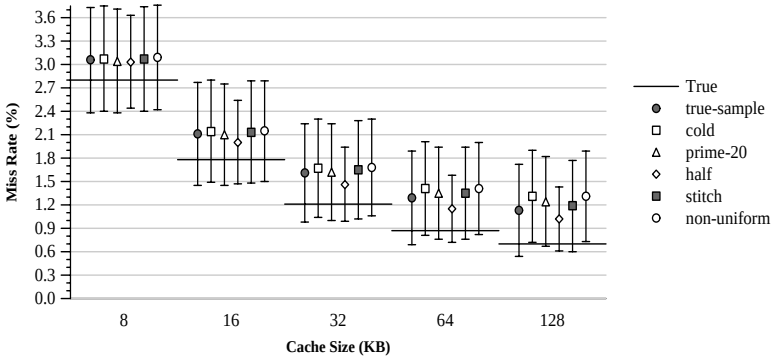
Victim caches are a relatively recent innovation in memory systems [Jouppi 90]. The scenario we consider here is that of an architect who wishes to gather an efficient estimate of the expected decrease in miss ratio for data caches when those caches are augmented with victim caches with between one and five entries. Our purpose here is not to study victim cache trends, as that has been done elsewhere [Jouppi 90], but to demonstrate how sampling techniques may be efficiently used in this regard.

The results are presented in Figure 5 for the *netscape* trace. Figure 5 compares the true results for the victim cache simulation to the *stitch* sampled results. The sampled results were obtained in one tenth of the time required to generate the true miss rates since only one tenth of the original trace was used. In addition to the actual miss rates being very similar, the true miss rate is always either within the 90% confidence interval or slightly outside, the trends are precisely the same. Even if the actual miss rates didn't agree as well, the predicted trends would be correct. In particular, the trends would be accurately represented even for large caches.



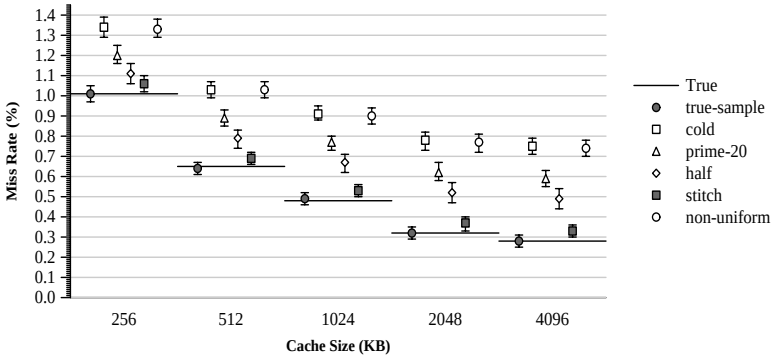
acrord32 (direct, icache) sampling results

Figure 1. *Simulation results for* acrord32.



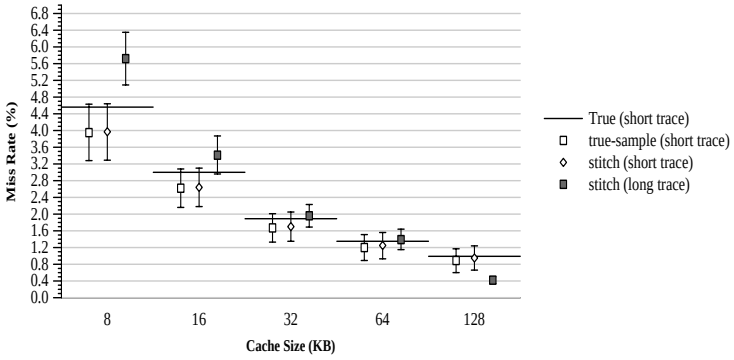
powerpnt (4-way, dcache) sampling results

Figure 2. *Simulation results for* powerpnt.



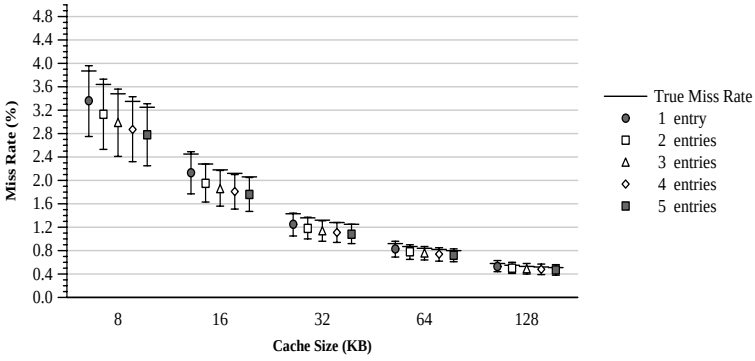
winword (direct, ccache) sampling results

Figure 3. *Simulation results for* winword.



netscape (direct, dcache) long trace sampling results

Figure 4. *Simulation results for the long* netscape *trace.* true-sample *was not available for the long trace since we did not record all references.*



netscape-stitch (direct, dcache) victim cache sampling results

Figure 5. *Victim Cache Simulation results for* netscape.

5 Conclusion and further study

This study has considered when trace sampling may be accurately used in architectural studies. Results were given that demonstrate, for a suite of Windows NT desktop application traces, that sampling techniques may be used to very accurately estimate cache miss ratios for caches of size 64KB and smaller. Of the techniques surveyed, *stitch* was found to be the most effective for this workload, followed by *half*. Furthermore, an example was given in the analysis of victim cache trends that demonstrates how architectural studies may be conducted with sampling, thus permitting great savings in simulation time.

There are a number of additions to this work that are currently under way. Two are major additions, and one is minor. The minor addition will see the *INITMR* estimator presented in [Wood et al. 91] added to the list of sampling techniques evaluated here (this is the most accurate technique and should be included for completeness). A more major addition will evaluate the effectiveness of trace sampling in the estimation of parameters for analytical cache models like those of Agarwal [Agarwal et al. 89]. Finally, the limits of sampling will be evaluated by considering sampling measures beyond the cache memory hierarchy. In particular, we will consider the accuracy of sampling branch prediction behavior for the same set of Windows NT desktop application traces.

References

[Agarwal et al. 88] Agarwal, A., Hennessy, J., and Horowitz, M. Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, 6(4):393–431, November 1988.

[Agarwal et al. 89] Agarwal, A., Horowitz, M., and Hennessy, J. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2):184–215, May 1989.

[Cochran 77] Cochran, W. G. *Sampling Techniques*. John & Wiley Sons, 1977.

[Fu & Patel 94] Fu, J. W. C. and Patel, J. H. Trace driven simulation using sampled traces. In *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences Vol. I: Architecture*, pages 211–220, January 1994.

[Jouppi 90] Jouppi, N. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of 17th Int. Symp. on Computer Architecture*, pages 364–373, 1990.

[Kessler et al. 94] Kessler, R., Hill, M. D., and Wood, D. A. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, June 1994.

[Laha et al. 88] Laha, S., Patel, J. H., and Iyer, R. K. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Transactions on Computers*, 37(11):1325–1335, November 1988.

[Lee et al. 98] Lee, D. C., Crowley, P. J., Baer, J.-L., Anderson, T. E., and Bershad, B. N. Execution characteristics of desktop applications on windows nt. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[Martonosi et al. 93] Martonosi, M., Gupta, A., and Anderson, T. Effectiveness of trace sampling for performance debugging tools. In *Proceedings of ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pages 248–259, 1993.

[Wood et al. 91] Wood, D. A., Hill, M. D., and Kessler, R. E. A model for estimating trace-sample miss ratios. In *Proceedings of the ACM SIGMETRICS Conference for the Measurement and Modeling of Computer Systems*, pages 79–89, June 1991.