Exact Analysis of Bernoulli Superposition of Streams into a Least Recently Used Cache *

Hanoch Levy
RUTCOR, Rutgers University
New Brunswick, NJ 08903

hanoch@rutcor.rutgers.edu

Robert J. T. Morris
I.B.M. Almaden Research Center,
San Jose, CA 95120

rjtm@almaden.ibm.com

Revised, December 1994

Abstract

We present an exact analysis of the superposition of address streams into a cache buffer which is managed according to a Least Recently Used (LRU) replacement policy. Each of the streams is characterized by a stack depth distribution, and we seek the cache hit ratio for each stream, when the combined, or superposed, stream is applied to a shared LRU cache. In this paper the combining process is taken to be a Bernoulli switching process. This problem arises in a number of branches of computer science, particularly in database systems and processor architecture.

Previously, a number of approximation techniques of various complexities have been proposed for the solution of this problem. The main contribution of this paper is the description of an exact technique. We evaluate the performance of the exact and an approximate technique on realistic data, both in a lab environment and a large database installation. The results allow comparisons of the techniques, and provide insight into the validity of the Bernoulli switching assumption.

1 Introduction

We consider models which can be used to predict the behavior of a cache buffer when it is subjected to the combining, or superposition, of I streams of requests, each of which has a known stack depth distribution. This cache buffer is assumed to be managed according a Least Recently Used (LRU) replacement policy. While a combined trace of the I streams, or timestamped traces of each of the streams, contains the complete information needed

*The present paper is an extension to the conference paper [6]. In that paper a simple approximation method and an improved method were described to solve the superposition problem. This paper extends the earlier work by providing the exact solution. However since the simple approximation has lower computational complexity than the exact method, we also repeat its description here, and show its accuracy using trace data.

for an exact analysis, stack depth distributions provide a much more compact, manageable, and understandable representation. Traces can be processed into these stack depth distributions with a single pass of the data [5]. I/O device drivers have been constructed, for example [7], which automatically collect this information for individual streams. If it is desired to collect such information for a number of streams in a manner which is more compact than a trace, but to retain the identity of streams, then these stack depth distributions must be collected on an individual basis. If one then wishes to design buffering strategies which combine more than one stream into a buffer, then the problem of characterizing the superposed stream must be solved. It should be clear that further assumptions must be made about how these two streams will be combined in order to obtain the effect of the superposed stream, however as we will see in the experimental results section a simple Bernoulli switching assumption can be made which results in effective predictions, at least in certain database environments.

Previously, a number of approximate techniques of various complexities and accuracies were proposed to solve this problem [2] [6] [8]. In this paper we derive a practical technique to obtain the exact solution. Thus the modeler can now concentrate on the validity of other assumptions (e.g., how the streams are combined) rather than any errors or artifices that may be introduced by the analysis.

We now illustrate by example the practical application of solving this superposition problem. The first example arises when separate (non-timestamped) traces or depth distributions are collected, on different machines or at different times, and the effect of a combined workload or stream needs to be analyzed for purposes of system re-configuration or analysis. The second example concerns the allocation and sizing of buffer pools. It is relevant to the tuning of database management systems (e.g., [9]) in which different objects (e.g., tables, indexes) may be assigned to separate buffer pools to improve performance. Given a set of streams, it is relatively easy to solve an optimization problem which determines the size of each member of a set of individual buffer pools, one for each stream, which maximizes the overall hit ratio, subject to an overall buffer size constraint [1]. However, this approach cannot simply be applied to a set of streams consisting of a stream for each object, since this would imply a number of buffer pools which is larger than can be practically implemented on a system, or would result in a poor degree of resource sharing and therefore poor performance. The question arises as to how these streams can be combined to share a smaller set of (perhaps 2 or 3) buffer pools. By solving the superposition problem efficiently, we can provide procedures which determine which streams should be combined to optimize performance. This problem is addressed in [4].

Consider I streams, $S^1, \cdots, S^I$, having *stack depth distributions* $d^1(n), \cdots, d^I(n)$. That is, when stream i is applied to a LRU-managed buffer, the probability that the current reference finds the element that it references at depth n ($n = 1$ is the most recently used element in the buffer) is $d^i(n), i = 1, \cdots, I$. Note that $d^i(n)$ is a "defective" distribution in

[1]See [8] for a method for miss ratios which are a convex function of buffer size, [4] for a method in the general case.

that it may not sum to unity: elements that have never previously been accessed will be assumed to be found at an infinite stack depth. For modeling purposes, we will assume that the streams actually obey the *stack reference model*, that is, they are stochastic processes which choose their next reference according to independent samplings of the stack depth distribution. Such processes are called *stack depth processes*, and their hit ratio curves as a function of cache size coincide with their cumulative stack depth distributions. This is one of several simple models for reference streams, and it has been reported, for example [1], that this model tends to be quite successful in capturing temporal locality of references within a trace, and is superior to the so called *independent reference model* or IRM.

Several other papers have considered the problem of superposition of streams into a cache. The paper [2] provides a comprehensive treatment of the problem of characterizing the complexity of buffer pool access in a realistic database management system. It presents a method for characterizing a composite workload as consisting of three types of access patterns: locality within a transaction, random access by transactions, and sequential access by long requests. This paper describes several effects that must be taken into account when modeling realistic database management systems. It is based on the use of a skewed independent reference model. But because the IRM is limited to modeling streams that have concave hit ratios, and these occur for example when looping behavior is present, the authors use a method to filter out non-random behavior and model it separately. This was a satisfactory approach because some database management systems separately account for such non-random behavior in the way they manage buffers. An approximate analysis of processor caches containing two types of elements (instruction and data) is described in [8], and simple approximate models are given to determine equilibrium conditions and optimal allocation.

In order to carry out the analysis of this paper it is necessary to make some assumptions about how the streams combine, or mix, into a single stream. We will assume that a I-way Bernoulli process chooses the stream for each access. Thus, according to an independent random trial, with probability λ_i, the element accessed is drawn from stream i ($\sum_{i=1}^{I} \lambda_i = 1$). Throughout the paper we will also adopt the steady state assumption that none of the buffer positions are empty.

2 A Simple Approximate Model

We now describe a simple approximate model based on flow conservation which will be shown to capture the essence of the effect of combining streams; this model was first introduced in [6]. We will define the data elements referenced by stream i as type i elements and, for simplicity, we assume that elements referenced by the individual streams are disjoint. The rate of flow into the buffer of type i elements is the type i miss rate, which can be approximated by $\lambda_i(1 - D^i(X_i))$, where $D^i(n) = d^i(1) + \cdots + d^i(n)$ and X_i is the mean steady state contents in the buffer of type i elements. This is an approximation since the population of type i elements in the buffer is a random variable, and the mean may not be integral, but as reported below this is not a significant source of error. The rate of flow of type i elements out of the buffer is taken to be $\sum_{j=1}^{I} \lambda_j(1 - D^j(X_j))X_i/B$, where B is the size of the finite buffer. This is an approximation of the miss rate of all types, multiplied by an estimate of the probability of a type i element being replaced. This estimate assumes a "well-mixed buffer" property, namely that the type of element of the buffer that is replaced is chosen in proportion to the buffer population of each particular type. Under a LRU replacement strategy, this is equivalent to the assumption that the population distribution *by type* in the last buffer (least recently used) position coincides with the entire buffer distribution. This is a mean-value version of a similar assumption made in [8]. We will have more to say about this approximation in the next section. Now, over the long run, the rate of inflow to, and outflow from, the buffer of type i elements must balance so we have that

$$\lambda_i(1 - D^i(X_i)) = \sum_{j=1}^{I} \lambda_j(1 - D^j(X_j))X_i/B, \quad i = 1, \cdots, I,$$

or,

$$\lambda_i(1 - D^i(X_i))B/X_i = \sum_{j=1}^{I} \lambda_j(1 - D^j(X_j)), \quad i = 1, \cdots, I.$$

These I nonlinear equations in I unknowns are easily solved numerically, since the right hand side is independent of i. It will also become clear that this solution is unique. To see how, define $h_i(x) = \lambda_i(1 - D^i(x))B/x$. For simplicity, extend the definition of $D^i()$ to allow any positive argument using any nondecreasing, continuous interpolation. Now we need to find C and $X_1, \cdots, X_I$ to solve

$$h_i(X_i) = C, \quad i = 1, \cdots, I,$$

and

$$\sum_{i=1}^{I} X_i = B.$$

Now noting that $h_i()$ is strictly decreasing (while positive), it has an inverse function $h_i^{-1}()$ which is also strictly decreasing and so the single equation $\sum_{i=1}^{I} h_i^{-1}(C) = B$ is easily solved numerically for a unique C. Once C is obtained, the X_i are obtained from $X_i = h_i^{-1}(C)$. The individual miss *rate* of stream i is then CX_i/B, and its miss *ratio* is $m_i = 1 - D^i(X_i) = CX_i/(B\lambda_i)$. Note that C has the interpretation of being the total miss rate for the combined stream.

A transparent special case occurs for $I = 2$. In this case, the balance equations become

$$\lambda_1(1 - D^1(X_1))(B - X_1) = \lambda_2(1 - D^2(B - X_1))X_1.$$

The left hand side decreases, and the right hand side increases, in X_1 thus resulting in an easily found unique solution. It can also be shown directly that the solution of this equation for X_1 in increasing in B. The computational complexity of this method for a fixed B is in its simplest implementation linear in B, although this can be reduced through a discrete root finding technique. If miss ratio curves are desired, they need only be plotted for the values of B desired; due to the monotonicity of X_1 in B, these can be computed using one pass through the values of X_1. Thus the overall complexity is linear.

2.1 Discussion

While this model will be shown to be quite accurate for our database buffer management applications, it is worth discussing its sources of error. Besides the assumption of the stack reference model, errors in the model can arise from three sources:

1. The Bernoulli assumption of combining the streams.
2. The use of means X_i as constant buffer contents in the miss rate expressions.
3. The "well-mixed buffer" assumption that the last element in the LRU buffer is populated *by type* according to the entire buffer population.

The main purpose of the first assumption is to provide that arriving references "see" the time average contents of the buffer, and so that flow balances on a per-arrival basis. A minor violation of this assumption can be tolerated without damaging the accuracy of the model, but if a stream of one type of references is active for so long that it significantly upsets the equilibrium contents of the buffer poor accuracy will result. For the database buffer problems we consider below consisting of, for example, index and data accesses to a buffer manager, this does not turn out to be a significant source of error, even though we do observe noticeable bursts of index- or data-only access.

The second assumption seems, on the surface, capable of introducing errors. The model can be enhanced to constitute a multidimensional Markov Chain, and correct for fluctuation in the buffer population by type. In doing this, the Bernoulli mixing assumption is explicitly needed for the process to be Markovian. We found negligible effect on any of our answers by using this Markov Chain model, and therefore we will not take up space in this paper for its description. Apparently, for all our database workloads, there is not a great deal of fluctuation in the buffer population (by type). Furthermore the apparent validity of the mixing process mentioned in the last paragraph further contributes to non-volatility. It is interesting to note that in [8], in the different context of mixing of instruction and data streams into a processor cache, relatively small variance in the occupancy of the cache by one of these entities was observed.

The third assumption provided the greatest source of errors in our studies. As a simple idealized stress-test case, we constructed a uniform independent reference model test trace with 250 distinct data elements chosen according to a uniform i.i.d. process for each of two types, and mixing probabilities $\lambda_1 = .9, \lambda_2 = .1$. Note that this is a special case of the stack reference model. For a buffer of size 300, this resulted in an overall occupancy of the first type of 78%, whereas the occupancy in the last place in the buffer was 44%.

The way this model is set up, the type 1 elements tend to occupy the top (most recently used) 250 elements in the cache, and the type 2 elements tend to drop to the bottom 50. This is an extreme case since it creates a widely differing population distribution *by type* in the cache. This behavior causes the approximate model to break down. A successful approximate correction for this effect was reported in [6]. However, all sources of error are eliminated by the exact method for Bernoulli superposition given in the next section.

3 Exact Analysis

In this section we provide an exact analysis of the Bernoulli superposition of stack depth streams and derive an efficient algorithm to derive the depth distribution of the combined stream. In Section 3.1 we carry out the analysis for the superposition of two streams. In Section 3.2 we discuss the computational aspects of this approach. In Section 3.3 we describe the application of the approach for I streams.

3.1 Bernoulli Superposition of Two Streams

Recall that S^1 and S^2 are two stack reference model page request streams and let S be the stream resulting from a Bernoulli superposition (with parameters λ_1, λ_2) of S^1 and S^2. Recall also that $d^1(n)$ and $d^2(n)$ are the stack depth distributions of the individual streams while $D^1(n)$ and $D^2(n)$ are the corresponding cumulative distributions. Let $d^i_{\{1,2\}}(n)$, $n = 1, 2, ...$, be the probability that when a type i page is requested from S, the page is found at depth n. Thus $d^1_{\{1,2\}}(n)$ is a depth distribution, just like $d^1(n)$, the subscript $_{\{1,2\}}$ emphasizes that the statistics of a type 1 request are obtained in the presence of both type 1 and type 2 requests. In what follows we focus on deriving $d^1_{\{1,2\}}(n)$ while the derivation of $d^2_{\{1,2\}}(n)$ is obtained by reversing the definitions of type 1 and type 2.

A straightforward approach would attempt to take advantage of Markovian properties, and carry out an analysis of the steady state probability distribution describing the entire shared cache. This approach is not practical, since it leads to a state space which is exponential in the cache size; the state space explosion results from the fact that in order to predict the future of the cache the state must capture which type occupies each position in the cache. The following alternative approach results in an exact analysis.

Let us consider a tagged type 1 page $\mathcal{P}$. At every point of time consider the number of type 1 pages (n_1) and the number of type 2 pages (n_2) residing above $\mathcal{P}$ (i.e., more recently accessed than $\mathcal{P}$) in the cache. Due to the memoryless properties of S^1 and S^2, and the memoryless property of the the Bernoulli combining process, the state (n_1, n_2) captures all that is needed to predict the future of $\mathcal{P}$.

In this analysis we trace $\mathcal{P}$ along its journey which starts at the top of the cache (location 1, the most recently used position) and ends when $\mathcal{P}$ is either requested (and thus returns to the top of the cache) or is pushed out of the cache (in the event that the stack is of finite size).

Let $q_1(n_1, n_2)$ be the probability that on this journey $\mathcal{P}$ *ever arrives to state* (n_1, n_2). Since $\mathcal{P}$ starts its journey at (0,0) (no page is above $\mathcal{P}$), it is obvious that

$$q_1(0,0) = 1 \tag{1}$$

Now, assume that $\mathcal{P}$ is in state (n_1, n_2) at time t. $\mathcal{P}$ will encounter one of the following events at the time of the next request at $t+1$:

1. $\mathcal{P}$ remains at (n_1, n_2). This occurs if either a request is made for a type 1 page of depth smaller than or equal to n_1, or if a request is made for a type 2 page of depth smaller than or equal to n_2. Thus the probability of this event is given by

$$\lambda_1 D^1(n_1) + \lambda_2 D^2(n_2)$$

2. $\mathcal{P}$ moves to (n_1+1, n_2). This occurs if a request is made for a type 1 page of depth greater than n_1+1; the probability of this event is

$$\lambda_1 \overline{D^1(n_1+1)}.$$

   where $\overline{X}$ denotes $1-X$.

3. $\mathcal{P}$ moves to (n_1, n_2+1). This occurs if a request is made for a type 2 page of depth greater than n_2; the probability of this event is

$$\lambda_2 \overline{D^2(n_2)}.$$

4. $\mathcal{P}$ finishes its journey. This occurs if a request is made for a type 1 page of depth n_1+1; the probability of this event is

$$\lambda_1 d^1(n_1+1).$$

It follows from the definitions of the distributions $d()$ and $D()$ that these four probabilities sum up to 1.

Note that this list determines the state of $\mathcal{P}$ at time $t+1$ as function of its state at time t; this includes the self-transition step resulting in $\mathcal{P}$ remaining at (n_1, n_2). For our analysis we are interested in the state to which $\mathcal{P}$ moves *once it leaves* (n_1, n_2). The list of these states and the corresponding transition probabilities are obtained from the above list using the standard conditioning technique:

1. $\mathcal{P}$ moves to (n_1+1, n_2). The probability of this event is

$$\frac{\lambda_1 \overline{D^1(n_1+1)}}{\lambda_1 \overline{D^1(n_1)} + \lambda_2 \overline{D^2(n_2)}}. \tag{2}$$

2. $\mathcal{P}$ moves to (n_1, n_2+1). The probability of this event is

$$\frac{\lambda_2 \overline{D^2(n_2)}}{\lambda_1 \overline{D^1(n_1)} + \lambda_2 \overline{D^2(n_2)}}. \tag{3}$$

3. $\mathcal{P}$ finishes its journey. The probability of this event is

$$\frac{\lambda_1 d^1(n_1+1)}{\lambda_1 \overline{D^1(n_1)} + \lambda_2 \overline{D^2(n_2)}}. \tag{4}$$

Note that the sum of the above probabilities still yields 1.

Using Equations 2 - 4, we can now use recursion to compute $q_1(n_1, n_2)$, the probability that $\mathcal{P}$ will eventually reach the state (n_1, n_2). This can be done by recognizing that $\mathcal{P}$ can reach (n_1, n_2) only from $(n_1 - 1, n_2)$, if $n_1 > 0$, or from $(n_1, n_2 - 1)$, if $n_2 > 0$.

This leads to a recursive expression for computing $q_1(n_1, n_2)$. For simplicity of presentation (of the cases in which either $n_1 = 0$ or $n_2 = 0$) define:

$$\begin{aligned} q_1(n_1, -1) &= 0 \qquad n_1 \geq 1 \\ q_1(-1, n_2) &= 0 \qquad n_2 \geq 1 \end{aligned} \tag{5}$$

and the recursive expression is:

$$\begin{aligned} q_1(n_1, n_2) &= q_1(n_1 - 1, n_2) \frac{\lambda_1 \overline{D^1(n_1)}}{\lambda_1 \overline{D^1(n_1 - 1)} + \lambda_2 \overline{D^2(n_2)}} \\ &+ q_1(n_1, n_2 - 1) \frac{\lambda_2 \overline{D^2(n_2 - 1)}}{\lambda_1 \overline{D^1(n_1)} + \lambda_2 \overline{D^2(n_2 - 1)}}, \\ &\qquad n_1 \geq 0, n_2 \geq 0, (n_1, n_2) \neq (0, 0). \end{aligned} \tag{6}$$

Now, let $d^1(n_1, n_2)$ be the probability that $\mathcal{P}$ ends its journey at (n_1, n_2), namely that when $\mathcal{P}$ is retrieved it has n_1 type 1 and n_2 type 2 pages above it. This event occurs if and only if $\mathcal{P}$ eventually reaches (n_1, n_2), and then it finishes its journey. Thus

$$d^1(n_1, n_2) = q_1(n_1, n_2) \frac{\lambda_1 d^1(n_1+1)}{\lambda_1 \overline{D^1(n_1)} + \lambda_2 \overline{D^2(n_2)}}. \tag{7}$$

Finally we can now use $d^1(n_1, n_2)$ to compute $d^1_{\{1,2\}}(n)$, the probability that when $\mathcal{P}$ leaves the stack it leaves from depth n. This is given by

$$d^1_{\{1,2\}}(n) = \sum_{n_1=0}^{n-1} d^1(n_1, n-1-n_1), \qquad n \geq 1. \tag{8}$$

As stated before, the distribution $d^2_{\{1,2\}}(n)$ is obtained by reversing the definitions of type 1 and type 2. Finally, the depth distribution of an arbitrary page in the merged stream, denoted by $d(n)$, is given by:

$$d(n) = \lambda_1 d^1_{\{1,2\}}(n) + \lambda_2 d^2_{\{1,2\}}(n) \tag{9}$$

Remark 1: It can be shown that the variables $d^1(n_1, n_2)$ have the property that their sum over all values of n_2 (for a given value of n_1) yields $d^1(n_1+1)$, namely: $\sum_{n_2=0}^{\infty} d^1(n_1, n_2) = d^1(n_1+1)$. To show this, it is permissible to lump all the states for which n_1 is the same into a single state (yielding a one dimensional state space) and examine the corresponding probabilities. Note that each of the states (n_1, n_2) for $n_2 = 0, 1, ...$ and n_1 fixed, has a transition probability to a state (n_1+1, n_2) of $\lambda_1 \overline{D^1(n_1+1)}$ and a termination probability of $\lambda_1 d^1(n_1+1)$. Thus, this set of states satisfies the conditions of *lumpability* (see, e.g., [3] Theorem 6.3.2), and the lumped chain has the desired termination probability of $d^1(n_1+1)$.

Remark 2: Note that if one is interested in computing $d(n)$ for a cache of size $B' < B$, then it is sufficient to use a partial representation of the functions $d^1()$ and $d^2()$ up to depth B' while the tails of their distributions are represented by redefining $d^i(B'+1) \stackrel{\text{def}}{=} \overline{D^i(B')}, i = 1, 2$. Thus we sweep up all the mass of the distribution $d^i()$ beyond B' and place it all at $B'+1$. This input is sufficient to provide exact values of $d(n)$ for $1 \le n \le B'$ and of the miss ratio in the combined stream which is $1 - \sum_{n=1}^{B'} d(n)$.

3.2 Computational Aspects

Equations 1, 5 and 6 provide a simple recursive algorithm to compute $q_1(n_1, n_2)$ on the triangle $n_1 + n_2 \le B - 1$. Equations 7 and 8 then allow computation of $d^1_{\{1,2\}}(n), n = 1, 2, ..., B$. The time complexity of the algorithm is $O(B^2)$ where B is the maximal depth for which the merged distribution is to be computed. Since the computation of $q_1(n_1, n_2)$ can be done in a "row by row " fashion, the algorithm can be carried out without having to store more than one row at a time, and thus its space complexity is linear in B.

3.3 Superposition of I Streams

Consider the general problem of I streams $S^1, ..., S^I$ which are chosen according to an I-way Bernoulli process, with parameters $\lambda_1, ...\lambda_I$, $\sum_{i=1}^{I} \lambda_i = 1$. We are interested in the stack depth distribution of the combined stream. The direct extension of the analysis described in Section 3.1 is to consider an I-dimensional state representation (replace $q_1(n_1, n_2)$ by $q_1(n_1, ..., n_I)$). However this has a computational complexity of $O(B^I)$ and a simpler approach is as follows.

The superposition of the I streams can be carried out approximately via a sequence of $I-1$ applications of the process in the previous section. Note that the superposition of a stack depth process is not a stack depth process. In the first stage one merges streams S^1 and S^2, with the Bernoulli probabilities $\lambda_1/(\lambda_1+\lambda_2)$ and $\lambda_2/(\lambda_1+\lambda_2)$. In the second stage one merges the stream resulting from the first stage with S^3 using Bernoulli probabilities $(\lambda_1+\lambda_2)/(\sum_{i=1}^{3}\lambda_i)$ and $(\lambda_3)/(\sum_{i=1}^{3}\lambda_i)$, respectively. This process continues for $I-1$ stages until all streams are combined into one stream. Thus, with a complexity of $O((I-1)B^2)$, we obtain an approximation to the overall depth distribution of the combined stream, similar to Equation (9). If it is desired to have the depth distribution as seen by each individual type in the combined stream, this complexity is increased by factor of $log(I)$. To accomplish this, merge all the depth distributions together two at at a time according to a tree, and save the intermediate results. Then, in order to obtain the depth distribution as seen by each individual type in the combined stream, repeat this process, but with one stream omitted, and merge it in at the end: this entails an additional $log(I)$ mergings. Carrying this out for all I streams has an overall complexity of $O(log(I)(I-1)B^2)$. The space complexity is $O(IB)$ regardless of whether the overall depth distribution or the individual depth distributions are required.

4 Experimental Results

In evaluating the performance of these two methods, a number of database traces were used. These traces consisted of all the logical I/O requests placed by a database management system with its buffer pool manager. Traces were collected from two environments: a large commercial installation, and also from a lab system under the control of the experimenters. Two other commercial installations were also traced and studied, with similar results. The lab environment was purposely configured to be simultaneously running a large query application (which included repeated sequential scans of tables) and a large transaction application (which made random accesses to large tables). Each of the traces consisted of several million I/O requests, and among other data, each trace record contained the identity of the requested database table/index and the page within that table/index. Further details of the precise setup for the lab environment are contained in [4]. In these experiments, we will consider a simple partition of the requests into two types (streams), namely, requests for data (i.e., table) pages and requests for index pages. These will be referred to as Types 1 and 2, respectively. A trace was processed by inputting it first to a simulation of a single LRU stack, and the precise depth distribution for each request was recorded in a histogram. This enabled a sample depth distribution to be obtained for the combined or superposed trace. Next the trace was split into a data and an index trace, and each was the applied to a separate LRU stack, thus obtaining sample depth distributions for the individual streams. As an indication of the advantage of working with stack depth distributions rather than traces, the depth distributions were thousands of bytes in length versus hundreds of millions of bytes for the traces.

Both sets of results are presented in the attached figures. The figures show the miss ratio curve for the actual combined trace (which did *not* satisfy the Bernoulli switching assumption). We then show the results, derived from the separate depth distributions and the Bernoulli switching assumption, of the approximate (Section 2) and exact (Section 3) models. The first overall observation is that the Bernoulli assumption is quite reasonable for this class of traces. It is interesting to observe that in the commercial installation, the miss ratio curve is decidedly non-convex,[2] so methods which are based on assumptions which imply convex hit ratio (concave miss ratio) curves would not be applicable. It should also be noted that the accuracy of these methods is lower at very low buffer pool sizes (note that buffer pools of less than 1000 buffers are rarely used in DBMS). This is because the Bernoulli mixing assumption is more of a problem for these small buffer sizes. In fact, eyeballing the traces, runs of several hundreds of one type of access could be observed. This is sufficient to start to upset the equilibrium conditions assumed of the buffers in the model.

In the earlier paper [6], it was observed that the simple approximate model could break down, and so in that paper an improved model was described that modeled the individual type population at each depth in the buffer. The latter model was shown to have considerably improved performance, particularly for the "stress test" described in Section 2.1. When we developed the exact solution, we checked it against the improved approximate model, and found that the latter model was very accurate for all cases that we examined. However, since the improved approximate model has the same computational complexity as the exact model, it is rendered uninteresting by the present method and it is not described further here.

5 Conclusions

Bernoulli switching is found to be a suitable assumption in predicting the effect of combining cache buffer streams at least for the data and index streams in a database management system. Under this assumption an approximate model with linear complexity, or an exact model with quadratic complexity can be used and both show good agreement with experimentally observed results. Bernoulli mixing appears to be an appropriate assumption, provided that the buffer is not so small that its size is comparable with the lengths of bursts of one type of data.

If the streams being combined have pronounced "burst characteristics" that lead to errors, then extensions to these models may be needed. For example, if the streams consist of requests to individual database objects, then these streams would be expected to show considerable "burstiness" as different queries or transactions execute and access those objects. However, in the above experiments we restricted attention to a relatively

[2] Since the abscissa of the graph is on a log scale, this might not be immediately obvious, but is borne out by closer examination.

coarse typing of the requests (into index and data) and so it is not surprising that the Bernoulli assumption (that is, the absence of significant burst or transitional activity) suffices. Extensions to these superposition techniques for situations where it is important to capture burstiness are the subject of further work by the authors [4]. Nevertheless, in situations such as those reported here where the Bernoulli assumption is adequate, the present technique provides an exact solution to this component of the analysis.

6 Acknowledgments

The authors would like to thank T. G. Messinger and R. L. Mattson for collecting and post-processing the traces on which the experimental work was based.

References

[1] E. G. Coffman and P. J. Denning, "Operating System Theory," Prentice-Hall, 1973.

[2] A. Dan, P. Yu, and J. Y. Chung, "Database Access Characterization for Buffer Hit Prediction," *9'th International Conference on Data Engineering*, Vienna, Austria, 1993, 134-152.

[3] J. G. Kemeny and J. L. Snell "Finite Markov Chains," D. Van Nostrand Co., 1963.

[4] H. Levy, T. G. Messinger and R. J. T. Morris, "The Cache Assignment Problem and its Application to Database Buffer Management," preprint available from authors.

[5] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, Vol. 9, No. 2, 1970, 78-117.

[6] R. J. T. Morris, "Analysis of Superposition of Streams into a Cache Buffer", 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1993, Santa Clara, CA.

[7] B. Samadi, "TUNEX: A Knowledge-based System for Performance Tuning of the UNIX Operating System," *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, July, 1989, 861-874.

[8] H. S. Stone, J. Turek and J. L. Wolf, "Optimal Partitioning of Cache Memory," *IEEE Transaction on Computers*," Vol. 41, No. 9, Sep. 1992, 1054-1068.

[9] J. Z. Teng and R. A. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance," *IBM Systems Journal,* Vol. 23, No. 2, 1984.

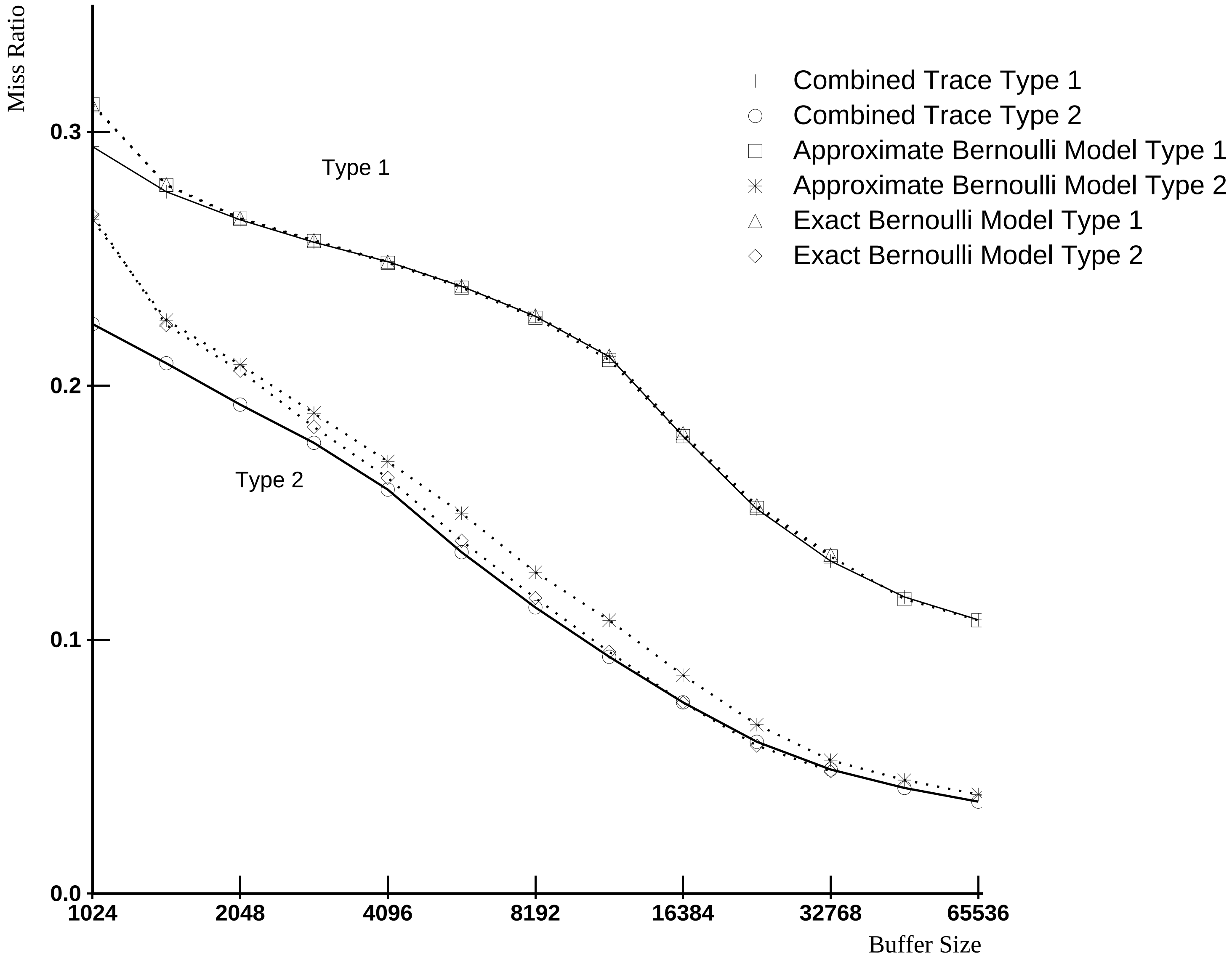

Figure 1: Miss ratio versus buffer size for a large commercial workload.

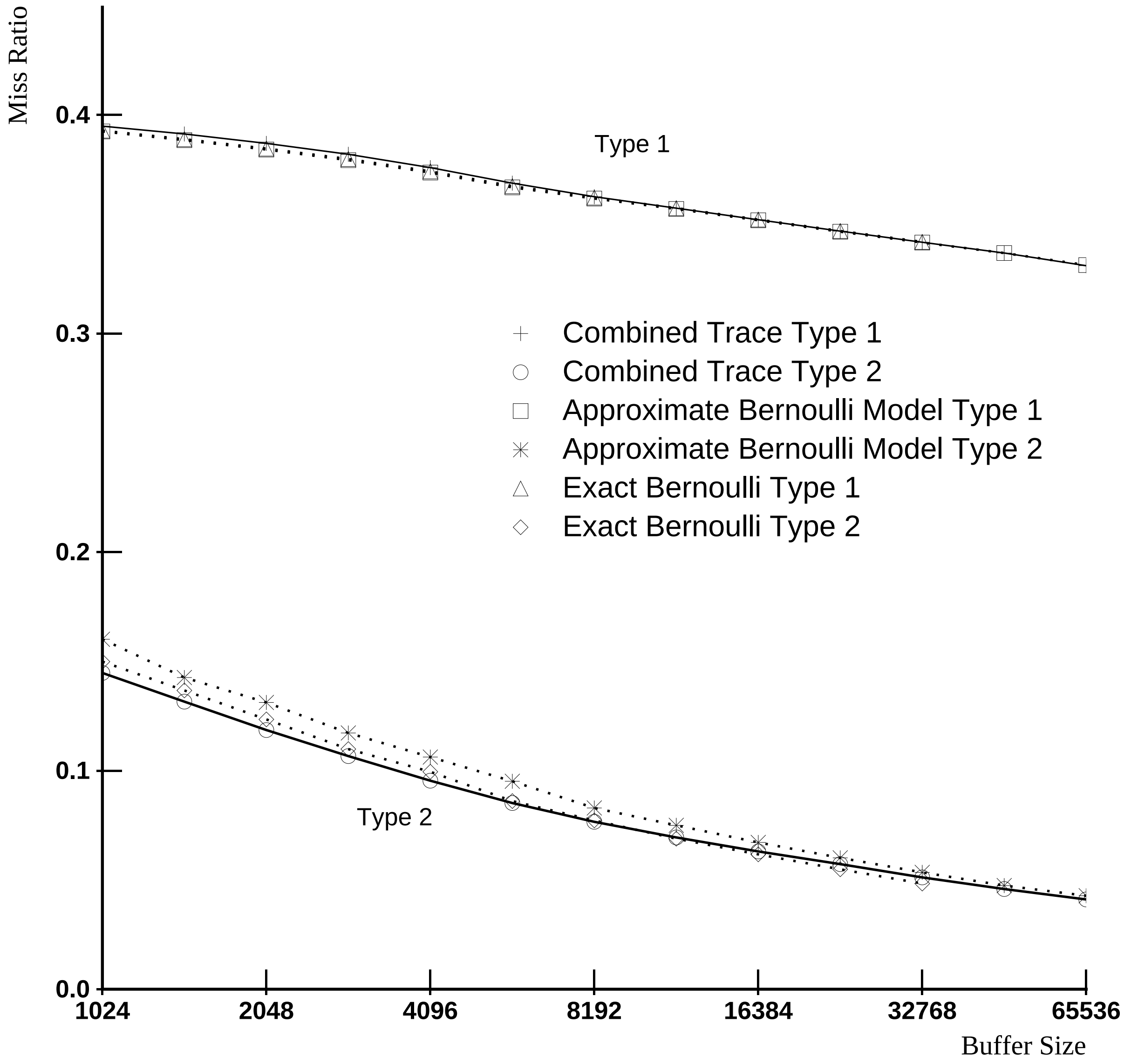

Figure 2: Miss ratio versus buffer size for a lab workload consisting of a mixture of transaction and query work.