ELSEVIER

Performance Evaluation 27&28 (1996) 175–188

PERFORMANCE EVALUATION
An International Journal

Should caches be split or shared? Analysis using the superposition of bursty stack depth processes

Hanoch Levy [a,*], Robert J.T. Morris [b,1]

[a] *RUTCOR, Rutgers University, New Brunswick, NJ 08903, USA*
[b] *I.B.M. Almaden Research Center, San Jose, CA 95120, USA*

Abstract

We address the question as to whether caches containing items of different types should be split or shared. We characterize the access to those caches as stack depth processes and show that the process defining the mixing of the access streams to the different types is the critical determining factor, with the burstiness of the individual streams being a key parameter. We characterize the mixing process using a double-geometric model, and give an exact solution for the behavior of a LRU cache when it is subjected to such a mixed stream of stack depth processes. We show that this minimalist approach is quite successful in dealing with real data from a database application. We further extend the model to a Markov mixing model which is more accurate but has higher complexity.

Keywords: Cache; Memory hierarchy; Stack depth processes; Database performance

1. Introduction

Throughout the history of computer systems, the cache and memory hierarchy have retained a central role. In this paper we ask the question: if there are two types of items in a cache, should the available cache space be split between the two types or should the two types share a common cache? This question arises in a variety of contexts and is, for example, an actual and difficult database administration function [2,7]. The "cache" might be a section of main memory which serves as the page buffer for a database management system, and the different types of requests might be for index pages or data pages. Or the cache might be on a processor module, and the different requests might be for instruction or data lines.

Most caches are managed according to some variant of the Least Recently Used (LRU) policy, in which when a new item has to be brought into the cache the item to be replaced is chosen to be the

* Corresponding author. E-mail: hanoch@rutcor.rutgers.edu. On leave of absence from Tel-Aviv University, Department of Computer Science.
[1] E-mail: rjtm@almaden.ibm.com.

PII S0166-5316(96)00016-8

one which has been least recently used. The proportion of times that the requested item is found in the cache is called the *hit ratio*. The cache can be represented as a *stack* data structure: every time an item is accessed it is introduced at the top of the stack. If the accessed item was already present below in the stack, this is just a reordering of the stack, but if it is not present then a "victim" must be chosen to leave to make room for it (at steady state the cache is always full). The victim chosen for replacement is the one at the bottom of the stack, and this is indeed the least recently used item.

To get any insight into how caches behave in practice, we need to understand the streams of requests that represent their accesses. In general these streams are arbitrary, but a powerful model has been developed which helps. It is called the stack depth process [1]. To characterize the stream we consider an *infinite* stack just like the one described above. At steady state, the request process impinging on the (infinite) stack finds the requested data item (from a previous request) at depth $\mathcal{D}$ ($\mathcal{D}$ is a random variable, and its value is taken as infinite if the item was never in the cache before). The process is then represented by the *stack depth distribution* of $\mathcal{D}$. This process can effectively represent *temporal locality of access* by having the mass of the depth distribution concentrated near the origin. It can also represent cyclic access to a set of items (a distribution with a single atom) [2].

When should caches be split and when should they be shared? We will use some very simple examples to illustrate. Suppose process A repeatedly accesses data items $1, 2, 3, \ldots, 10$ and process B repeatedly accesses data items $11, 12, 13, \ldots, 20$. Now suppose processes A and B are running together and provide an interleaved stream of access, i.e., $1, 11, 2, 12, \ldots, 10, 20, 1, 11, 2, \ldots$. It is clear that if we had memory for 15 cache items, we would get the best result (with a LRU cache) by giving either process A or B space for 10 items – after that we do not have a good use for the remaining five spaces. Note that the process to which we give 10 will get a 100% hit ratio; to give any less than 10 would result in a 0% hit ratio and obviously giving it any more would not help. But giving all 15 to a shared cache for processes A and B would result in an overall 0% hit ratio. Thus the best overall result for such an interleaved process is a 50% hit ratio and a split cache is the right choice. Now suppose processes A and B run in long bursts, i.e., the combined stream consists of many repetitions of $(1, 2, 3, \ldots, 10)$ followed by many repetitions of $(11, 12, 13, \ldots, 20)$. In this case, it is clear that with only memory for 10 cache items, we can get an almost 100% hit ratio using a shared cache. The improved "time sharing" of the space results in a better result. In the case of "bursty" processes, we clearly want to reuse the memory resources, and the shared cache is the optimal choice. We have used cyclic streams in this example only for illustration. Indeed, if streams were purely cyclic, we would not use LRU. Of course in real systems, the processes are not so simple and the tradeoff between the respective advantages of splitting and sharing caches is not so clear. In fact, for real systems, LRU is a robust algorithm that is often chosen and is optimal under certain conditions [1]. The development of simple models to guide this decision is thus the subject of the paper.

We conclude from the above illustration that the way the streams are *mixed* or *superposed* is of key importance in deciding whether to split or share the cache. In fact it can easily be shown [4] that if the mixing is "random" (a Bernoulli mixing process where we randomly choose a stream to obtain each arrival), then *appropriately sized split caches are always optimal.* However, every system designer looks for ways to share scarce resources between competing demands and it is clear that a shared cache is the right choice when the streams are more bursty.

[2] Traces of item requests can be processed into stack depth distributions with a single pass of the data [5]. I/O device drivers have been constructed, for example [8], which automatically collect this information for individual streams.

In this paper, we provide a minimalist Markov model for a *bursty* mixing of the streams and provide an exact analysis under this model for the performance of a shared LRU cache subjected to such a mix of stack depth processes. The analysis for the case of separate (optimally split) caches being easy, we are then in a position to answer the question as to when caches should be split or shared.

The structure of this paper is as follows: In Section 2 we analyze the performance of a stream resulting from a double-geometric superposition. In Section 3 we present the analysis of a superposition following a more general model – one in which the superposition is governed by a general Markov chain. In Section 4 we provide an example in which the effect of burstiness on cache performance and on the decision whether to split or combine caches is demonstrated. In Section 5 we provide two examples taken from an actual database trace. Concluding remarks are given in Section 6.

The issue of modeling cache performance when it is subjected to the superposition of streams was addressed by several approximate techniques in the past (refer to [3,6] and [9] for complete references). A model of superposition which allows exact analysis, but which does not capture burstiness (since it is applied only to Bernoulli mixing of streams) was presented and analyzed in [3].

2. Double-geometric superposition of two stack reference streams

In this section we propose and study a model of superposition between stack depth processes which can capture not only the mix ratio between the streams but also their burstiness. The model proposed is called a *double-geometric superposition model.*

2.1. Notation and definitions

Consider two streams, P_1, P_2, having *stack depth distributions* $d^1(n), d^2(n)$. That is, when stream P_i is applied to a LRU-managed cache, the probability that the current reference finds the item that it references at depth n, $n = 1, 2, \ldots$ ($n = 1$ is the most recently used item in the cache) is $d^i(n)$, $i = 1, 2$. Note that $d^i(n)$ is a "defective" distribution in that it may not sum to unity: items 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 are *stack depth processes*, that is, they are stochastic processes which choose their next reference according to independent samplings of the stack depth distribution. 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.* Given a stack depth distribution $d()$, one may define the *cumulative depth distribution* $D(i) = \sum_{j=1}^{i} d(j)$, and so obtain $D_1()$ and $D_2()$.

A double-geometric superposition with parameters λ_1 and λ_2 of two stack depth processes, P_1 and P_2, can be described as follows: At every time epoch t, $t = 1, 2, \ldots$, we select the *next* item from either P_1 or P_2. If at time t the selected item is from P_1 (P_2) then at time $t + 1$ the selected item is taken from P_1 (P_2) with probability λ_1 (λ_2) and from P_2 (P_1) with probability $1 - \lambda_1$ ($1 - \lambda_2$). This process is called double-geometric superposition since the superposed process consists of alternating bursts of P_1 and P_2 and the lengths of the bursts are distributed according to the (shifted) geometric distribution with parameters λ_1 and λ_2.

The double-geometric superposition can be also described as driven by a simple discrete-time two-state Markov chain with states 1 and 2: if the chain state at time t, $S(t)$, is i ($i = 1, 2$) then at t we select the next item from P_i. Also, the transitions of the Markov chain are as follows: if $S(t) = 1$ then

$S(t+1) = 1$ with probability λ_1 and $S(t+1) = 2$ with probability $1 - \lambda_1$; similarly, if $S(t) = 2$ then $S(t+1) = 2$ with probability λ_2 and $S(t+1) = 1$ with probability $1 - \lambda_2$.

Remark 1. In the special case in which $\lambda_1 = 1 - \lambda_2$ the superposition is identical to the Bernoulli superposition studied in [3].

The two parameters λ_1, λ_2 allow one to capture both the *mix ratio* between the processes and their *burstiness*. The mix ratio is characterized by the relative fractions of type-1 (f_1) and type-2 (f_2) items within the superposed stream P; these are given by

$$f_1 = \frac{\bar{\lambda}_2}{\bar{\lambda}_1 + \bar{\lambda}_2}, \qquad f_2 = \frac{\bar{\lambda}_1}{\bar{\lambda}_1 + \bar{\lambda}_2}. \tag{1}$$

where $\bar{x} \stackrel{\text{def}}{=} 1 - x$.

The burstiness of the streams can be conveniently defined using the measure of *burst length*. To define this measure note that the combined stream can be viewed as alternating bursts of type-1 and type-2 items. A type-1 burst is a consecutive sequence of type-1 items in which the first item is preceded and the last item is succeeded by a type-2 item; a type-2 burst is defined similarly. Note, that the length of the burst is greater than or equal to one.

Let b_i denote the expected length of a type-i burst. The fractions of items which are of type-1 and type-2 are then expressed by

$$f_1 = \frac{b_1}{b_1 + b_2}, \qquad f_2 = \frac{b_2}{b_1 + b_2}. \tag{2}$$

Also, in the double-geometric stream, we have

$$b_1 = \frac{1}{1 - \lambda_1}, \qquad b_2 = \frac{1}{1 - \lambda_2}. \tag{3}$$

The *relative burst length*, or the *burstiness*, b, is defined to be the ratio between b_1 and the expected burst length of the type-1 items in the *corresponding Bernoulli process*; the corresponding Bernoulli process is a Bernoulli superposition process with relative population fractions f_1 and f_2. Note that b is also identical to the ratio between b_2 and the expected burst length of the type-2 items in the corresponding Bernoulli process. The value of b can be computed from the fact that in the corresponding Bernoulli process the expected burst length of type-1 items is given by $\frac{1}{1-f_1}$ and from $f_1 = \frac{b_1}{b_1+b_2}$

$$b = \frac{b_1 b_2}{b_1 + b_2}. \tag{4}$$

It is easy to see that

$$\frac{1}{2} \leq b \leq \infty. \tag{5}$$

namely, b is bounded below by $1/2$ and is not bounded from above. Recall that for a Bernoulli process $b = 1$. An implication of this property is that the burstiness in the Bernoulli process is relatively low. Thus, the Bernoulli process is expected to be a good model for processes with low burstiness and not so good for processes with high burstiness. This property will be later examined in Sections 4 and 5.

2.2. Analysis

Next we analyze the stack depth distribution of the combined stream when it is fed into a LRU stack. To analyze this system consider a type 1 tagged item (denoted $\mathcal{E}_1$) and trace its journey in the stack. Let the triple (n_1, n_2, S) represent the state of $\mathcal{E}_1$ where n_1 and n_2 ($n_1 \geq 0, n_2 \geq 0$) represent the number of type 1 and type 2 items above (more recently used) than $\mathcal{E}_1$ and $S \in \{1, 2\}$ represents the state of the switching process. Note that the future of $\mathcal{E}_1$ is fully determined by this state. We will trace the journey of $\mathcal{E}_1$ starting at the top of the stack (i.e. at state (0,0,1) or (0,0,2)) and continuing as long as $\mathcal{E}_1$ does not return to the top of the stack. Note that since we are interested in a single journey of $\mathcal{E}_1$, all states represent the location of $\mathcal{E}_1$ prior to its return to the top of the stack.

Let $t = 0$ be the time at which $\mathcal{E}_1$ starts its journey. Let $q^1(n_1, n_1, S; t)$ denote the probability that at time t $\mathcal{E}_1$ is at state (n_1, n_2, S) under the current journey. The values of $q^1(n_1, n_1, S; t)$ can be computed in a simple recursive way

$$q^1(0, 0, 1; 0) = \lambda_1 \tag{6}$$

$$q^1(0, 0, 2; 0) = \overline{\lambda_1} \tag{7}$$

$$q^1(n_1, n_2, S; 0) = 0, \qquad n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0), S \in \{1, 2\} \tag{8}$$

$$q^1(0, 0, 1; t) = 0, \qquad t > 0 \tag{9}$$

$$q^1(0, 0, 2; t) = 0, \qquad t > 0 \tag{10}$$

$$\begin{aligned} q^1(n_1, n_2, 1; t) = {} & q^1(n_1^-, n_2, 1; t-1)\lambda_1 \overline{D_1}(n_1) + q^1(n_1, n_2, 1; t-1)\lambda_1 D_1(n_1) \\ & + q^1(n_1, n_2^-, 2; t-1)\overline{\lambda_2}\,\overline{D_2}(n_2^-) + q^1(n_1, n_2, 2; t-1)\overline{\lambda_2} D_2(n_2) \\ & n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0), \ t > 0 \end{aligned} \tag{11}$$

$$\begin{aligned} q^1(n_1, n_2, 2; t) = {} & q^1(n_1^-, n_2, 1; t-1)\overline{\lambda_1}\,\overline{D_1}(n_1) + q^1(n_1, n_2, 1; t-1)\overline{\lambda_1} D_1(n_1) \\ & + q^1(n_1, n_2^-, 2; t-1)\lambda_2 \overline{D_2}(n_2^-) + q^1(n_1, n_2, 2; t-1)\lambda_2 D_2(n_2) \\ & n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0), \ t > 0, \end{aligned} \tag{12}$$

where n^- denotes $n - 1$.

Note that Eqs. (6) and (7) stem from the fact that when $\mathcal{E}_1$ starts its journey at (0, 0) it starts it either at the state $S = 1$ (with probability λ_1) or at the state $S = 2$ (with probability $1 - \lambda_1$). This is implied directly from the fact that at $t = -1$ the requested page is of type-1 and from the definition of the double-geometric arrival process.

Let us now define $s^1(n_1, n_1, S) = \sum_{t=0}^{\infty} q^1(n_1, n_2, S; t)$; note that $s^1(n_1, n_1, S)$ can be interpreted as the expected number of time units that $\mathcal{E}_1$ stays in the state (n_1, n_2, S) during the journey [3]. The recursion given in Eqs. (6)–(12) leads to a similar recursion between the s variables

$$s^1(0, 0, 1) = \lambda_1 \tag{13}$$

$$s^1(0, 0, 2) = \overline{\lambda_1} \tag{14}$$

[3] One may ask under what conditions the infinite sum converges. While this is an interesting question, for all practical purposes the depth distributions $d^1()$ and $d^2()$ can always be taken to have finite support. It follows that a finite set of data items then circulate through the cache, and thus they will all have a finite equal mean interaccess interval. Thus, in this case, the expected time of the journey is finite.

$$s^1(n_1, n_2, 1) = s^1(n_1^-, n_2, 1)\lambda_1\overline{D_1}(n_1) + s^1(n_1, n_2, 1)\lambda_1 D_1(n_1) + s^1(n_1, n_2^-, 2)\overline{\lambda}_2\overline{D_2}(n_2^-) + s^1(n_1, n_2, 2)\overline{\lambda_2} D_2(n_2), \qquad n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0) \tag{15}$$

$$s^1(n_1, n_2, 2) = s^1(n_1^-, n_2, 1)\overline{\lambda}_1\overline{D_1}(n_1) + s^1(n_1, n_2, 1)\overline{\lambda_1} D_1(n_1) + s^1(n_1, n_2^-, 2)\lambda_2\overline{D_2}(n_2^-) + s^1(n_1, n_2, 2)\lambda_2 D_2(n_2), \qquad n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0). \tag{16}$$

Note that (13) and (14) are consistent with the fact that $\mathcal{E}_1$ remains in position 1 of the stack exactly one time unit. This is the case since any access to another item in the stack will push $\mathcal{E}_1$ to position 2, while an access to $\mathcal{E}_1$ will terminate the current journey.

For these equations to properly hold on the boundaries ($n_1 = 0, n_2 = 0, (n_1, n_2) \neq (0, 0)$) we define

$$s^1(n_1, -1, 1) = s^1(n_1, -1, 2) = 0, \qquad n_1 > 0,$$
$$s^1(-1, n_2, 1) = s^1(-1, n_2, 2) = 0, \qquad n_2 > 0.$$

Note that the set of equations does not lend itself to a simple recursive solution due to the mutual dependency between the states. Nonetheless, this difficulty may be overcome in the following way: we may focus on a pair of variables $s^1(n_1, n_2, 1)$ and $s^1(n_1, n_2, 2)$ and consider the two equations associated with these variables. These are two equations (represented by Eq. (15) and Eq. (16) above) in two unknowns ($s^1(n_1, n_2, 1)$ and $s^1(n_1, n_2, 2)$). We may now solve this set to yield

$$s^1(0, 0, 1) = \lambda_1 \tag{17}$$

$$s^1(0, 0, 2) = 1 - \lambda_1 \tag{18}$$

$$s^1(n_1, n_2, 1) = \left\{[1 - \lambda_2 D_2(n_2)]\left[\lambda_1\overline{D_1}(n_1)s^1(n_1^-, n_2, 1) + \overline{\lambda}_2\overline{D_2}(n_2^-)s^1(n_1, n_2^-, 2)\right] + (1 - \lambda_2)D_2(n_2)\left[\overline{\lambda}_1\overline{D_1}(n_1)s^1(n_1^-, n_2, 1) + \lambda_2\overline{D_2}(n_2^-)s^1(n_1, n_2^-, 2)\right]\right\} \left\{[1 - \lambda_1 D_1(n_1)][1 - \lambda_2 D_2(n_2)] - \overline{\lambda}_2 D_2(n_2)\overline{\lambda}_1 D_1(n_1)\right\}^{-1}, \qquad n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0) \tag{19}$$

$$s^1(n_1, n_2, 2) = \left\{(1 - \lambda_1)D_1(n_1)\left[\lambda_1\overline{D_1}(n_1)s^1(n_1^-, n_2, 1) + \overline{\lambda}_2\overline{D_2}(n_2^-)s^1(n_1, n_2^-, 2)\right] + [1 - \lambda_1 D_1(n_1)]\left[\overline{\lambda}_1\overline{D_1}(n_1)s^1(n_1^-, n_2, 1) + \lambda_2\overline{D_2}(n_2^-)s^1(n_1, n_2^-, 2)\right]\right\} \left\{[1 - \lambda_1 D_1(n_1)][1 - \lambda_2 D_2(n_2)] - \overline{\lambda}_2 D_2(n_2)\overline{\lambda}_1 D_1(n_1)\right\}^{-1}, \qquad n_1, n_2 \geq 0, (n_1, n_2) \neq (0, 0). \tag{20}$$

Note that in Eqs. (19) and (20), $s^1(n_1, n_2, 1)$ and $s^1(n_1, n_2, 1)$ are expressed directly in terms of lower index variables. Thus, Eqs. (19), (20), (17) and (18) now allow us to use a simple recursive procedure for deriving the values of $s^1(n_1, n_2, 1)$ and $s^1(n_1, n_2, 2)$, $n_1, n_2 \geq 0$. The recursive procedure first determines $s^1(0, 0, 1)$ and $s^1(0, 0, 2)$ (from Eqs. (17) and (18)). It proceeds by sequentially increasing the values of n_1 and n_2 to recursively obtain $s^1(n_1, n_2, 1)$ and $s^1(n_1, n_2, 2)$ from Eqs. (19) and (20).

Next, define $d^1(n_1, n_2, S)$ ($S = 1, 2$) to be the probability that $\mathcal{E}_1$ finishes its journey at (n_1, n_2, S). This is simply given by

$$d^1(n_1, n_2, 1) = \sum_{t=0}^{\infty} q^1(n_1, n_2, 1; t)d_1(n_1 + 1) = s^1(n_1, n_2, 1)d_1(n_1 + 1)$$
$$d^1(n_1, n_2, 2) = 0, \qquad n_1, n_2 \geq 0. \tag{21}$$

Now, define $d^1_{\{1,2\}}(n)$ to be the probability that when $\mathcal{E}_1$ is requested it is in depth n. This is given by

$$d^1_{\{1,2\}}(n) = \sum_{n_1=0}^{n-1} d^1(n_1, n-1-n_1, 1), \qquad n \geq 1. \tag{22}$$

In a similar way we can trace the evolution of a type-2 tagged item, denoted $\mathcal{E}_2$, in the LRU stack. The analysis of $\mathcal{E}_2$ is identical to that of $\mathcal{E}_1$ and is done in a symmetric way (simply by renaming the item types). Using similar notation, $d^2(n_1, n_1, S)$ is defined to be the probability that $\mathcal{E}_2$ finishes its journey at (n_1, n_2, S) $(S = 1, 2)$, and $d^2_{\{1,2\}}(n)$ the probability that when $\mathcal{E}_2$ is requested it is in stack depth n.

Using this notation, the probability that when an arbitrary item (either type-1 or type-2) is requested it is in stack depth n, is given by

$$d(n) = \frac{\bar{\lambda}_2}{\bar{\lambda}_1 + \bar{\lambda}_2} d^1_{\{1,2\}}(n) + \frac{\bar{\lambda}_1}{\bar{\lambda}_1 + \bar{\lambda}_2} d^2_{\{1,2\}}(n). \tag{23}$$

2.3. Computational aspects

To compute $d(n)$ for $n = 1, \ldots, N$ one needs to compute all the values $s^i(n_1, n_2, j)$, $i = 1, 2$, $j = 1, 2, n_1 + n_2 < n$. Since each of these computation is done in constant time, the overall computation for $d(n), n = 1, \ldots, N$, requires $O(N^2)$ operations. Since the computation can be done "row by row" the amount of space required is $O(N)$.

3. Superposition of streams according to a general Markov chain

While the double-geometric model allows burstiness to be captured, it does not allow all mixing behaviors to be represented. This generalization can be achieved by modeling the mixing process as an M state Markov chain, with the type of data item being accessed being a function of the chain's state. A little thought shows that arbitrarily distributed burst lengths can be modeled in this way. The chain consists of M states, indexed $1, \ldots, M$. The transitions between the states are represented by a transition matrix $\Lambda = (\lambda_{i,j})$ where $\lambda_{i,j}$ denotes the probability that at time $t+1$ the chain will be at state j given that at time t it is at state i; note that $\sum_{j=1}^{M} \lambda_{i,j} = 1$, for $1 \leq i \leq M$. Let $q = (q_1, \ldots, q_M)$ be the steady state probability vector of this chain, namely q_i is the probability that in steady state the chain is at state i. This vector can be found by solving the linear equation set

$$q\Lambda = q \tag{24}$$

and the normalization equation

$$\sum_{i=1}^{M} q_i = 1. \tag{25}$$

Since the type of element that is accessed is a function of the state, it follows that the M states of the chain can be renamed and partitioned into two sets, $\{1, \ldots, M_1\}$ and $\{M_1 + 1, \ldots, M\}$ such that $1 < M_1 < M$. When the Markov chain is at state $1 \leq m \leq M_1$ the next item selected is taken from P_1, while when the Markov chain is at state $M_1 + 1 \leq m \leq M$ the next item selected is taken from P_2.

Remark 2. In some situations it is more convenient to consider a process in which at state m the next item is selected from Process 1 with probability π_m and from Process 2 with probability $1 - \pi_m$. However, every such process can be represented by the simpler one in which $\pi_m = 1$ (at state m always select P_1) or $\pi_m = 0$ (at state m always select from P_2).

The analysis approach provided in Section 2 can be generalized to accommodate this model. We consider a type-1 tagged item, denoted $\mathcal{E}_1$, and trace its journey in the LRU stack, from the moment it is in the top of the stack (most recently accessed) until it is accessed again. At every moment the system state relevant to the future behavior of $\mathcal{E}_1$ is given by the number of type-1 items and the number of type-2 items residing above (more recently accessed) it and the state ($1 \le m \le M$) of the Markov chain; the system state is therefore represented by the triple (n_1, n_2, m).

Similarly to the analysis in Section 2, let $s^1(n_1, n_2, m)$ be the expected number of time units that $\mathcal{E}_1$ stays in (n_1, n_2, m). Relations between the variables $s^1(n_1, n_2, m)$ can be written in a similar manner to Eqs. (13)–(16)

$$s^1(0, 0, k) = \sum_{m=1}^{M_1} q_m \lambda_{m,k} \tag{26}$$

$$\begin{aligned} s^1(n_1, n_2, k) = {} & \sum_{m=1}^{M_1} s^1(n_1^-, n_2, m)\lambda_{m,k}\overline{D_1}(n_1) + s^1(n_1, n_2, m)\lambda_{m,k} D_1(n_1) \\ & + \sum_{m=M_1+1}^{M} s^1(n_1, n_2^-, m)\lambda_{m,k}\overline{D_2}(n_2^-) + s^1(n_1, n_2, m)\lambda_{m,k} D_2(n_2), \\ & n_1, n_2 \ge 0, (n_1, n_2) \ne (0, 0), 1 \le k \le M. \end{aligned} \tag{27}$$

Note that Eq. (26) resembles Eqs. (17) and (18). Equation (26) results from two properties:

(1) $\mathcal{E}_1$ stays at $(0, 0, k)$ exactly one time unit at each journey. After arriving at $(0, 0, k)$, the next time unit must result either in $\mathcal{E}_1$ being pushed to $(0, 1, k')$ or $(1, 0, k')$ for some $1 \le k' \le m$ or in moving to state $(0, 0, k')$ $(1 \le k' \le m)$. The latter move however represents a request made to $\mathcal{E}_1$, in which the move belongs to a new journey.

(2) Given that $\mathcal{E}_1$ is in $(0, 0, k)$ at time t, it implies that at time $t - 1$ the Markov chain must have been in either of the states $1 \le m \le M_1$. The conditional probability that the Markov chain was at $t - 1$ at state m, given that it was at either of the states $1, \ldots, M_1$ is given by $\frac{q_m}{\sum_{i=1}^{M_1} q_i}$.

Similarly to Eqs. (13)–(16), Eqs. (27) and (26) do not lend themselves to a direct recursive computation. However, the structure of Eq. (27) suggests that provided that the values of the variables $s^1(n_1^-, n_2, m)$ and $s^1(n_1, n_2^-, m)$ are available for $1 \le m \le M$ then the equations for $s^1(n_1, n_2, k)$, $1 \le k \le M$ form an independent set of M equations in M unknowns. Thus, once the variables $s^1(n_1^-, n_2, m)$ and $s^1(n_1, n_2^-, m)$, $1 \le m \le M$, are computed, the solution for the values of $s^1(n_1, n_2, m)$, $1 \le m \le M$, can be achieved by solving M linear equations.

The equation for deriving the values of $s^2(n_1, n_2, k)$ are done in a similar way to those of Eqs. (27) and (26). The computation of $d^i(n_1, n_2, j)$, $i = 1, 2$; $j = 1, 2$, and of $d^i_{\{1,2\}}(n)$ are done as in Eqs. (21) and (22). The computation of $d(n)$ (equivalent to Eq. (23)) is done as follows:

$$d(n) = d^1_{\{1,2\}}(n) \sum_{m=1}^{M_1} q_m + d^2_{\{1,2\}}(n) \sum_{m=M_1+1}^{M} q_m. \tag{28}$$

This results from the fact that the fraction of accesses to items which are of type-1 is given by $f_1 = \sum_{m=1}^{M_1} q_m$.

3.1. Computational aspects

To compute $d(n)$ for $n = 1, \ldots, N$ one needs to compute all the values $s^i(n_1, n_2, j)$, $i = 1, 2$, $j = 1, 2, \ldots, M$, $n_1 + n_2 < n$. Thus the overall complexity is $O(N^2M^3)$ and the space complexity is $O(N) + O(M^2)$.

4. The burstiness and its effect on superposition modeling

As stated before, a simple alternative for modeling the mixing or superposition of reference streams is the Bernoulli model. That model allows both fast exact computation of the superposed depth distribution and simple approximations [3]. While both the Bernoulli model and the double-geometric model allow one to model any arbitrary mix ratio between the processes, the latter model is richer in that it allows the stream to have larger (or smaller) bursts than those of the Bernoulli model. The major question in selecting between the models is, therefore, the effect of the burstiness on the accuracy of the model.

To address this issue we now consider a synthetic example similar to the motivating example given in the Introduction. We use it to examine the effect of the burstiness on the depth distribution of superposed processes.

In the example we consider two stack depth processes P_1 and P_2. The depth distribution of P_1 is given by

$$\begin{aligned} d_1(k) &= 0; \qquad k \neq 1000, \\ d_1(1000) &= 1. \end{aligned} \tag{29}$$

The depth distribution of P_2 is identical to that of P_1, but the two processes address a disjoint set of items. Note that in each stream the items are referenced in a cyclic pattern that repeats itself in a period of 1000. We examine the superposition of P_1 and P_2 with an equal number of references coming from each stream.

Thus, we have $f_1 = f_2 = 0.5$. In this case we consider three situations:

(1) $b = 1$ (Bernoulli mixing); thus $b_1 = b_2 = 2$ (and $\lambda_1 = \lambda_2 = 0.5$).
(2) $b = 500$; thus $b_1 = b_2 = 1000$ (and $\lambda_1 = \lambda_2 = 0.999$).
(3) $b = 1000$; thus $b_1 = b_2 = 2000$ (and $\lambda_1 = \lambda_2 = 0.9995$).

We also compare the performance using an optimally split cache. It is easy to see that the optimal split policy (for an available total cache size between 1000 and 2000 items) will grant all of its first 1000 spaces to P_1 and the rest to P_2).

We observe from Fig. 1 that the performance of the joint stream under a joint cache is very sensitive to the burstiness of the streams. Furthermore, we observe that the decision whether to use a shared cache or an optimally split cache is sensitive both to the cache size and to the burstiness level. For example, note that when the burstiness is 1 (Bernoulli) the optimally split cache is highly superior to the joint cache for any cache size (as proved in [4]). When the joint stream is very bursty ($b = 1000$) a shared cache is highly superior over the whole range. When the burstiness level is between these levels ($b = 500$) the superiority varies as function of the cache size.

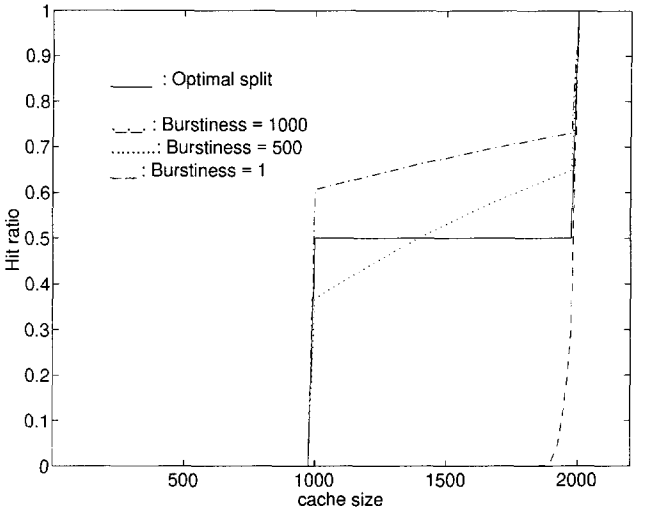

Fig. 1. A synthetic example – shared vs. optimal split cache.

5. Experimental results from a real database trace

In this section we consider a real database trace and evaluate the accuracy of prediction of the various superposition models. A trace was obtained from a large DB2 database installation. The page requests are directed to 47 different database tables, and thus the stream of requests can be considered as a superposition of 47 streams, each directed to a different table.

Out of these 47 streams we first selected a pair of streams and examined it. The two streams consist of 3204 and 10411 requests respectively. Initially, we consider two *artificial* interleaving scenarios for these streams: the first interleaves them in 13 equal sized (deterministic) bursts (thus leading to block sizes of roughly 246 and 800 respectively); the second interleaves them in 62 equal sized segments (thus leading to block sizes of roughly 51 and 167 respectively). Note that these interleaving scenarios are chosen to test and stress the various modeling techniques.

We now examine how this interleaving affects the performance of the shared cache and the quality of our model in predicting the cache behavior.

To carry out this examination we do the following:

(1) Each of the two streams is individually applied to an LRU cache and the depth distribution of the streams are recorded in $D_1()$ and $D_2()$.
(2) We measure the mix ratio of the streams (and record them as f_1 and f_2) and the expected burst sizes in the combined interleaved stream (and record them as b_1 and b_2).
(3) We apply the combined stream to a single LRU cache and record the depth distribution of the merged stream; we denote this "exact".

Using the measurements derived in items 1 and 2 above, we now use the double-geometric and the Bernoulli models to predict the depth distribution of the joint stream. In addition we use an approximate approach (presented in [4]), which is based on assuming that the burst lengths are deterministic, to provide one more prediction (called "deterministic prediction").

In Figs. 2 and 3 we consider these two cases (13 interruptions and 62 interruptions, respectively) by

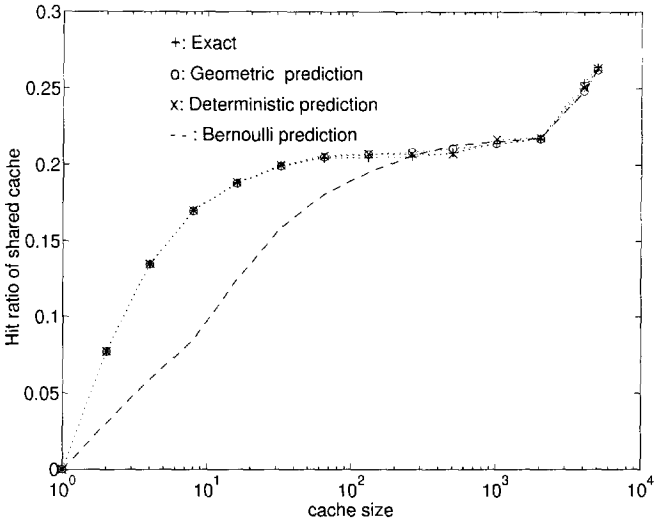

Fig. 2. A real example – 13 interruptions between the streams.

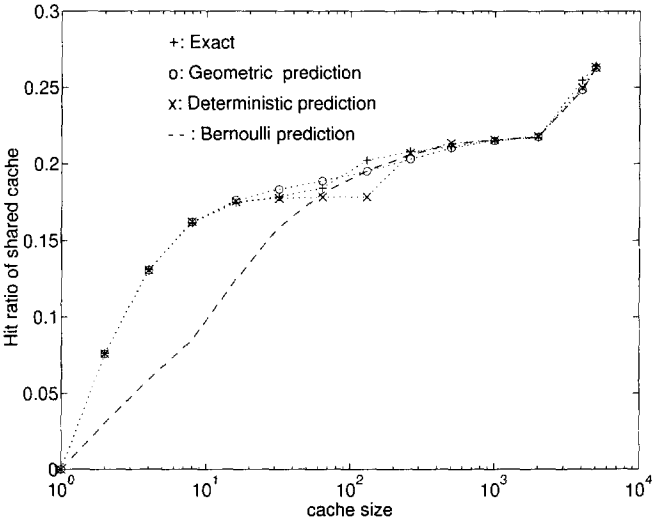

Fig. 3. A real example – 62 interruptions between the streams.

depicting the hit ratio (cumulative depth distribution) as function of the cache size. We observe that the actual performance of the system (as observed on the "exact" curve) is sensitive to the burstiness, and varies significantly between the figures. We also observe that the double-geometric model provides a very accurate prediction of the exact results, and responds adequately to the change in burstiness. In contrast, the Bernoulli prediction is not responsive to the changes in burstiness (as expected) – and its prediction is much worse than that of the double-geometric model. We also note that the deterministic model [4] responds adequately to the burstiness, but performs worse than the double-geometric model; this is despite the fact that the burst sizes are deterministic in this example (and not geometric), and

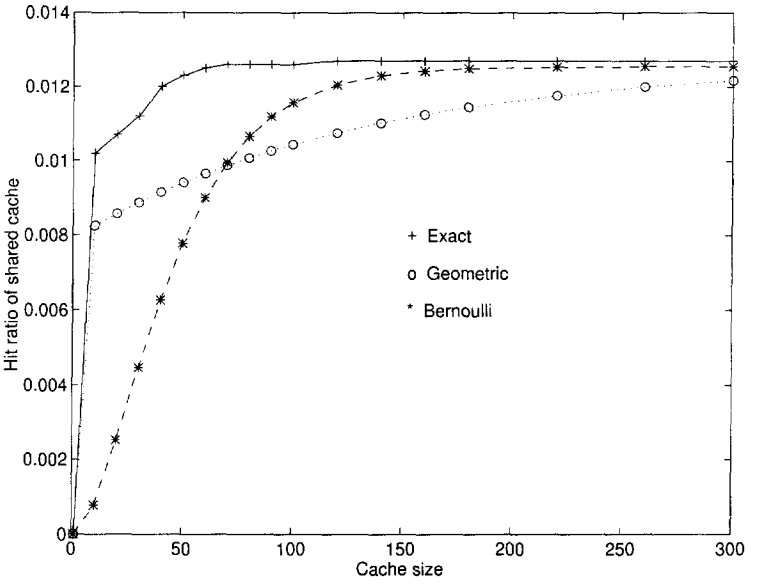

Fig. 4. Real streams (second pair) – low end.

is due to the fact that the deterministic model contains other approximations designed to reduce the computational effort.

We next consider a second pair of streams. This time we use the *actual interleaving or superposition that was found in the trace.* The total number of requests from both these streams is 104 028. The expected block sizes are $b_1 = 4.944$ and $b_2 = 113$. The relative burst length is therefore quite short: $b = 4.73$. Figures 4 and 5 show the behavior of these two streams; we plot the hit ratio of the shared cache as function of the cache size for: 1) Exact analysis, 2) Bernoulli model prediction, and 3) Double-geometric model prediction. Note that the exact analysis is obtained by taking the original combined trace from the database (which contained the actual interleaving) and applying that to a LRU simulation. The figures show the low end (Fig. 4) and the high end (Fig. 5) of the cache size range. Although the interleaving is in fact far more complex than represented by either the Bernoulli or double-geometric models, both models show quite good accuracy. The double-geometric model more correctly predicts the structure at the low end of the curve where the cache size is small, as would be expected. Overall the double-geometric model shows smaller "maximum error". The Bernoulli model shows relatively good performance (and close to the double-geometric) because the relative burst length is still quite low. However, as explained above and shown in Fig. 1, the Bernoulli approximation will tend to result in misleading cache allocations. A more complete study of cache allocation and assignment (assignment is the optimization of which streams should be placed in which caches) is the object of paper [4].

6. Conclusions

The task of analyzing a complex workload to see whether a cache should be split or shared can be addressed by a simple characterization of the workload comprising the stack depth distribution and mean burst length for each constituent stream. After obtaining that characterization, the decision as to whether to split or share the cache can be obtained by computing the effect of the superposition of these streams

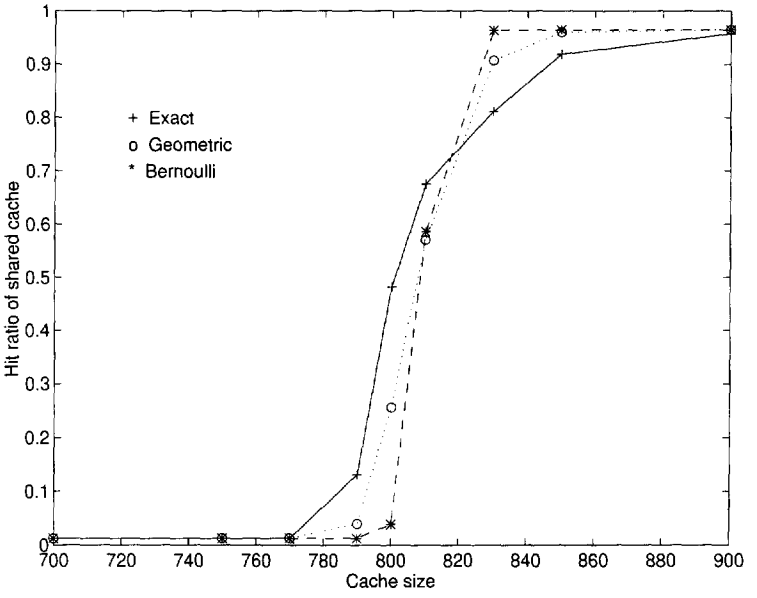

Fig. 5. Real streams (second pair) – high end.

into the cache. We have found that the methodology of this paper adequately models this superposition. Having developed these techniques, long traces can be reduced to compact representations that can then be used to efficiently analyze many options, for example those resulting when several streams and caches exist.

Just as importantly, the theoretical results concerning the importance of burstiness give good qualitative insight as to what to do. Indeed, if one takes a case of index and data access in a database management system, these are pretty well mixed, and we have found through extensive experimentation and field work (with our colleagues Ted Messinger and Dick Mattson) that only modest improvements in overall hit rates are obtainable from the separation of index and data into separate "buffer pools". However when different types of applications are running on a machine ("transaction" versus "complex query") the right combination of sharing and splitting can result is significant improvement. The complete solution of the problem of assignment of streams to caches is a more complex combinatorial problem which must be solved using more heuristic techniques [4]. Those techniques rely upon superposition as the basic building block. While approximate superposition methods were used in [4], the present paper provides an exact solution under the double-geometric assumption.

References

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

[2] C. Hoover, Understanding the DB2 Version 3 Buffer Manager, Compuware Corporation, Farmington Hills, Michigan (1994).

[3] H. Levy and R.J.T. Morris, Exact Analysis of Bernoulli Superposition of Streams into a Least Recently Used Cache, *IEEE Transactions on Software Engineering* **21** (8) (Aug. 1995) 682–688.

[4] H. Levy, T.G. Messinger and R.J.T. Morris, Improved Performance of Database Systems by Buffer Pool Cache Assignment, submitted.

[5] R.L. Mattson, J. Gecsei, D.R. Slutz and I.L. Traiger, Evaluation Techniques for Storage Hierarchies, *IBM Systems Journal* **9** (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, California.

[7] R.J.T. Morris, A General Method for Optimizing DB2 Buffer Pool Allocation, *Pacific Region DB2 User Conference*, 11/23-25/1994, Melbourne, Australia.

[8] B. Samadi, TUNEX: A Knowledge-based System for Performance Tuning of the UNIX Operating System, *IEEE Transactions on Software Engineering* **15** (7) (July 1989) 861–874.

[9] H.S. Stone, J. Turek and J.L. Wolf, Optimal Partitioning of Cache Memory, *IEEE Transaction on Computers* **41** (9) (Sep. 1992) 1054–1068.