Analysis of the Generalized Clock Buffer Replacement Scheme for Database Transaction Processing

Victor F. Nicola, Asit Dan and Daniel M. Dias
IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

The CLOCK algorithm is a popular buffer replacement algorithm because of its simplicity and its ability to approximate the performance of the Least Recently Used (LRU) replacement policy. The Generalized Clock (GCLOCK) buffer replacement policy uses a circular buffer and a weight associated with each page brought in buffer to decide on which page to replace. We develop an approximate analysis for the GCLOCK policy under the Independent Reference Model (IRM) that applies to many database transaction processing workloads. We validate the analysis for various workloads with data access skew. Comparison with simulations shows that in all cases examined the error is extremely small (less than 1%). To show the usefulness of the model we apply it to a Transaction Processing Council benchmark A (TPC-A) like workload. If knowledge of the different data partitions in this workload is assumed, the analysis shows that, with appropriate choice of weights, the performance of the GCLOCK algorithm can be better than the LRU policy. Performance very close to that for optimal (static) buffer allocation can be achieved by assigning sufficiently high weights, and can be implemented with a reasonably low overhead. Finally, we outline how the model can be extended to capture the effect of page invalidation in a multi-node system.

1 Introduction

The performance of database transaction processing systems is very sensitive to the database buffer hit probability, and therefore to the buffer replacement scheme. Improving the buffer hit probability reduces the average disk I/O per transaction, and thus improves the transaction response time as well as the throughput. There are a large number of buffer replacement policies in the literature (see [11] for a good survey). Two popular policies that have good performance over a wide range of workloads are the Least Recently Used (LRU) and the CLOCK algorithms [5, 17]. In the Generalized CLOCK (GCLOCK) algorithm [17], a counter is associated with each page, whose initial value (weight) is assigned when the page is brought into the buffer. In GCLOCK, the weights may be different for different data types. On a page miss, a circulating clock pointer sweeps through the buffer pages decrementing their counters until a page with a count of zero is found. This page is selected for replacement. On a buffer page hit, the counter associated with that page is reset (not necessarily to the initial count) or incremented.

A general conclusion in the literature is that while the CLOCK algorithm is simple and efficient, its performance is comparable to that of the LRU replacement policy [17, 11, 2, 4]. In this paper we develop approximate analytical models for the GCLOCK buffer replacement scheme under the Independent Reference Model (IRM [13]). We compare the performance of the GCLOCK algorithm with the Least Recently Used (LRU) schemes and with the optimal (static) buffer allocation policy [1], and examine when the performance of the GCLOCK algorithm differs from that of the LRU policy. Using a workload similar to the Transaction Processing Council benchmark A (TPC-A) [18] as an example, we show how the GCLOCK algorithm performs for this type of workload. We also quantify how the knowledge of access patterns, as in the case of TPC-A, can be used to advantage in the GCLOCK policy. We examine various cases of skewed data access and show both the accuracy of the approximate analytical model and the comparative performance of the GCLOCK policy with the LRU and the optimal buffer allocation policies.

There are few existing analytical models for the database buffer. An approximate analysis for the LRU policy is presented in [6] under IRM for skewed (non-uniform) data access, and extended to a multi-systems environment in [7, 8, 9]. However, no such analysis ex-

1992 ACM SIGMETRICS & PERFORMANCE '92-6/92/R.I., USA
© 1992 ACM 0-89791-508-9/92/0005/0035...$1.50

ists for the CLOCK or GCLOCK policies, though numerous simulation studies are reported in the literature (see [11] for a survey). Under IRM, each buffer page access is independent of all previous references. Such a model holds for many database transaction processing applications where each transaction accesses a relatively small set of pages. For instance, in the TPC-A workload, each transaction performs a debit/credit operation to an account chosen independently and equiprobably from the account relation. Therefore, the account references follow the IRM model. In general both sequential and random access patterns are found in typical database workloads. Further support for random access patterns is found in [12] where a large fraction (close to 90%) of the accesses were non-sequential. Furthermore, the random access patterns were successfully modeled as IRM in [10].

While the IRM model implies independent database accesses, these accesses need not be uniform over all pages. In the above TPC-A example, index pages to the accounts are accessed more frequently than account pages. The importance of considering such skewed data access for database applications is also discussed in [3], where a replacement strategy is modeled that fixes some buffers for the most frequently used blocks, and uses the remaining buffers to read in other blocks. Other buffer allocation methods, such as domain separation [15] or the hot set model [16], are static and inflexible [4]. We explicitly model the GCLOCK algorithm for a skewed access pattern. The model also captures the effect of different weights for different data types. For instance, in the TPC-A example, account pages can be assigned a lower weight than that of index pages. We show that the performance of the GCLOCK algorithm is sensistive to this weight assignment, and its performance can be either better or worse than that of the LRU replacement policy. The analysis is used to select the weights in the GCLOCK algorithm such that the buffer hit probability is arbitrarily close to that of optimal buffer allocation.

The analysis in this paper can be augmented to model various buffer coherence policies in a multi-node system. As an example, we outline the extension to analyze the broadcast invalidation policy [7, 9]. We also validate the extended model and show that it agrees closely with simulation.

The paper is organized as follows. Section 2 describes the analysis of the GCLOCK scheme. First, a simple approximate model is proposed. A more refined model is subsequently developed, which uses the solution of the approximate model as a starting point in an iterative solution. Validation, comparison with the LRU replacement policy and the optimal buffer allocation policy, weight assignment for the GCLOCK policy, and an extension to the multi-node case are presented in Section 3. A summary and concluding remarks appear in Section 4.

2 The Model

In this section we will first describe the workload environment and the details of the GCLOCK buffer replacement scheme. We consider a database consisting of P partitions such that access to database pages within a partition is uniform [6]. Skewed data access can be modeled by assigning different sizes and access frequencies to different partitions [7]. Let S_p be the size of partition p, $1 \leq p \leq P$. (A list of symbols used in the following analyses are given in Table 1). We assume that access to database pages follows the IRM model (i.e., each page request is independent of all previous requests) and each access is to a page of partition p with probability r_p. If a request is to a page that is not in the buffer, a buffer page must be selected for replacement. As outlined in Section 1, when a miss occurs, the clock points to the buffer page immediately following the page that was brought in for the previous miss. If the count associated with the page pointed at is zero, then the page is selected for replacement, otherwise the count is decremented and the next buffer page in sequence is examined, until a page with zero count is found and selected for replacement. The page that caused the miss is then fetched into the buffer location (after writing out the replaced buffer page, if necessary), and the count associated with it is set to the initial weight I_p (if the new access is to partition p). If a buffer request is for a page that is in the buffer (this can be determined using a hash table for pages in the buffer) then the count associated with that buffer page is reset to the hit weight (weight assigned on a hit), say, H_p. Note that, on a hit, there is no advancement of the clock pointer. We denote the maximum weight of partition p (the larger of the initial and hit weights) by L_p, i.e., $L_p = max(I_p, H_p)$. In the following analyses, we will assume that, on a hit, the counter of the buffer page is reset to the initial weight, i.e., $L_p = H_p = I_p$. The analysis is extended in [14] to include the cases where the initial and the hit weights may not be the same.

An exact Markov model for the GCLOCK policy, requires a very large state space, which is $O(B^{\sum_{p=1}^{P}(L_p+1)})$, where B is the buffer size in pages. For example, for a buffer size of 1000 pages, 3 partitions, and each partition with a maximum weight of 3, an exact model has 10^{36} states. Therefore, we must resort to approximate models. We first develop a simple Markovian model. This approximate model has the number of states equal to $\sum_{p=1}^{P}(L_p + 1)$. Balance equations are written for this model and simplified to P coupled equations. These equations are solved by an iterative procedure. This simple model ignores the distance of a buffer

Table 1: List of Symbols Used

Workload and system parameters

P	*Number of data partitions*
S_p	*Size of partition p (pages)*
r_p	*Prob. of accessing the p^{th} partition*
I_p	*Initial weight assigned to partition p*
H_p	*Hit weight assigned to partition p*
L_p	*$max(I_p, H_p)$*
B	*Buffer size (pages)*

Simple approximate model

h_p	*hit prob. of partition p*
m_p	*miss prob. of partition p*
h	*overall buffer hit probability*
m	*overall buffer miss probability*
n_p	*Average number of buffer pages of partition p*
$n_{p,i}$	*Average number of buffer pages of partition p with count i*
n_0	*Average number of buffer pages with count 0*

Additional symbols for the refined model

$n_{p,i,j}$	*Average number of buffer pages of partition p with count i and at a distance of j misses from the clock pointer*
$\tilde{n_0}$	*Average number of misses per clock cycle*

page from the clock pointer, and underestimates the buffer hit ratio of the hot partitions (as detailed later). We then make a refinement to the model, by which we roughly take into account the distance of a buffer page from the clock pointer. The resulting Markovian model has the number of states as the product of $\sum_{p=1}^{P}(L_p+1)$ and the average number of misses in one rotation of the clock pointer around the buffer (the reason for this will be detailed later). The solution from the simple model is used as a starting point for solving the refined model. Though the refined model is still approximate, validation in the next section shows that it is very accurate across a wide range of parameter values.

The objective of the model is to determine the buffer hit probability h_p for each partition. Let n_p be the steady-state average number of pages of partition p in the buffer. Then the buffer hit probability for this partition $h_p = \frac{n_p}{S_p}$ and its miss probability $m_p = 1 - h_p$. The overall hit probability is $h = \sum_{p=1}^{P} r_p h_p$, and the overall miss probability is $m = 1 - h$. In Section 2.1 we develop and analyze a simple approximate model, which is refined in Section 2.2. From these models, we will estimate n_p for each of the partitions.

2.1 Approximate Model

The Markov chain for the simple approximate model represents the state of an arbitrary buffer page at the instant of a random page request. A portion of that Markov chain, corresponding to partition p, is shown in Figure 1. In this figure, a state (p, i), $0 \leq i \leq L_p$, corresponds to a buffer page of partition p, with a count i. In other words, the steady-state probability of being in state (p, i) is the probability that an arbitrary buffer page is of partition p and has a count i. The complete Markov chain includes all portions (similar to that in Figure 1), corresponding to all partitions. Transitions among the parts of the Markov chain (corresponding to different partitions) occur only through states $(p, 0)$ and (p, L_p), $0 \leq p \leq P$, as shown.

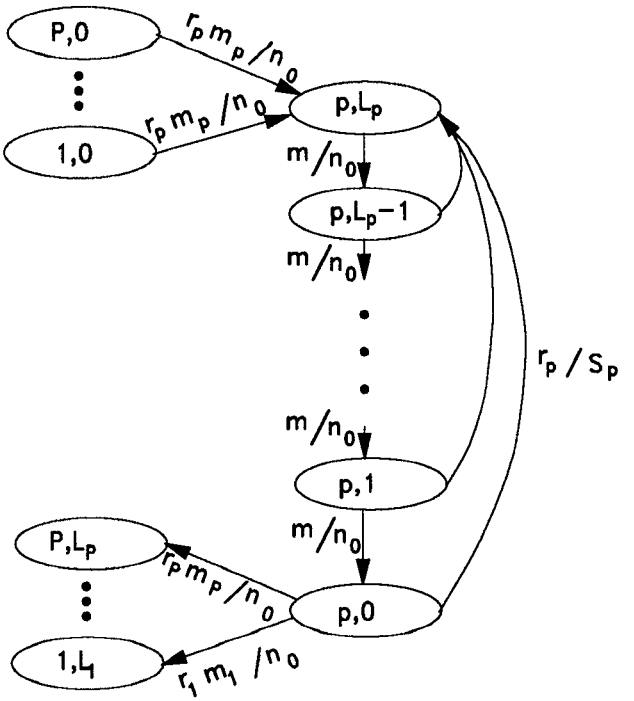

Figure 1: Approximate Model

Since all pages in buffer must sum to the buffer size B, we have

$$\sum_{p=0}^{P} n_p = B. \tag{1}$$

Let $n_{p,i}$ be the steady-state average number of pages of partition p having a count i, $0 \leq i \leq L_p$. Then

$$n_p = \sum_{i=0}^{L_p} n_{p,i}. \tag{2}$$

We denote by $n_0 = \sum_{p=0}^{P} n_{p,0}$ the steady-state average number of buffer pages, from all partitions, having a zero count. Define a clock cycle as a complete rotation

of the clock pointer through all pages in the buffer. Note that at each buffer miss, the clock traverses buffer pages until it finds a buffer page with a zero count. Therefore, in a clock cycle, the number of buffer misses, say $\tilde{n}_0$, equals the number of buffer pages with a zero count encountered during that cycle. The basic approximation in the simple model is assuming that the average number of misses in a complete cycle of the clock is equal to n_0, i.e., $\tilde{n}_0 = n_0$. From this approximation, it follows that, given a miss, the probability of decrementing the count of an arbitrary buffer page equals $\frac{1}{n_0}$. Therefore, at a page request, the transition probability from state (p, i) to state $(p, i-1)$, $0 < i \leq L_p$, is $\frac{m}{n_0}$. Note that the probability of a transition out of state $(p, 0)$ due to a miss is also $\frac{m}{n_0}$, and corresponds to the probability of replacing an arbitrary buffer page having a zero count. An arbitrary page of partition p is requested with probability $\frac{r_p}{S_p}$, and this is the transition probability from any state (p, i) , $0 \leq i \leq (L_p - 1)$, to state (p, L_p). On a request, the probability that a page of partition p is brought into the buffer is $r_p m_p$. Therefore, the transition probability from each of the states $(p, 0)$, $1 \leq p \leq P$ to state (p, L_p) is $\frac{r_p m_p}{n_0}$.

Before we proceed to derive n_p, the average number of partition p pages in buffer in the steady-state, we first note that the steady-state probability of a page being in state (p, i) can be expressed as $\frac{n_{p,i}}{B}$. In the following, we write the balance equations of the Markov chain in terms of, only, $n_{p,i}$, $0 \leq i \leq L_p$, since B cancels out in all balance equations.

Balance at the aggregate states for each of the partitions, yields

$$n_{p,0} = \frac{n_0}{m} r_p m_p, \quad 1 \leq p \leq P. \tag{3}$$

Balance at the state i, $0 \leq i \leq L_p - 1$, yields

$$n_{p,i} = n_{p,i-1}(1 + \frac{n_0}{m}\frac{r_p}{S_p}), \quad 1 \leq i \leq L_p.$$

It follows that

$$n_{p,i} = (1 + \frac{n_0}{m}\frac{r_p}{S_p})^i \; n_{p,0}, \quad 0 \leq i \leq L_p. \tag{4}$$

Substituting from Equations (3) and (4) in Equation (2), with $m_p = 1 - \frac{n_p}{S_p}$, we finally get

$$n_p = S_p(1 - \frac{1}{(1 + \frac{n_0}{m}\frac{r_p}{S_p})^{L_p+1}}), \quad 1 \leq p \leq P. \tag{5}$$

In the above expression for n_p, notice that the quantities m and n_0 always appear together as the ratio $\frac{n_0}{m}$, which can be determined by substituting from Equation (5) in (1). The resulting nonlinear equation can be solved iteratively for the ratio $\frac{n_0}{m}$. One possible way to solve this equation is the bisection method, which, according to our experience, converges very quickly. Once the ratio $\frac{n_0}{m}$ is determined we can evaluate n_p, $1 \leq p \leq P$, from Equation (5), and, hence, also the hit probabilities for each partition.

The approximate model gives reasonably accurate results, but, as will be shown in Section 3, tends to underestimate high hit probabilities for the relatively hot partitions (mostly within 5% of the values obtained from simulation). However, this translates to large relative errors for the low miss probabilities, which may not be acceptable. Next we develop a more accurate model to overcome this problem.

2.2 Refined Model

As mentioned earlier, the main approximation in this simple model is the assumption that the average number of misses per clock cycle, which we denote by $\tilde{n}_0$, is equal to n_0, the steady-state average number of pages with a zero count in buffer. This assumption is not accurate, since pages with a count zero may have been hit and converted to pages with a non-zero count, as the clock pointer traverses one round through the buffer. This effect causes n_0 to overly estimate $\tilde{n}_0$, particularly for high overall hit probabilities. To capture this effect in our analysis, we must include some kind of measure for the distance from the clock pointer in the state representation of an arbitrary buffer page. By definition, on the average, it takes $\tilde{n}_0$ misses for the pointer clock to return to the same buffer page. Therefore, one way to model the distance from the clock pointer is to add another phase (state) variable in the state representation of a buffer page. Upon replacement or decrement of its count, a page is in phase $\tilde{n}_0$ (i.e., it is $\tilde{n}_0$ misses away from the clock pointer). With each miss, the phase of a buffer page is decreased by one, since the page is getting one miss closer to the clock pointer. Only at the $\tilde{n}_0$-th miss after decrement (or replacement), the count of a buffer page is again decremented (or replaced, if it has a zero count). Note that, in this refined model, a buffer page may be hit and its count reset to the hit weight before the clock pointer returns to it. This captures the missing effect in the approximate model. The refined model is not exact, since we have modeled the distance from the clock pointer by a deterministic variable $\tilde{n}_0$ rather than a stochastic variable. However, because $\tilde{n}_0$ is usually a large number, this approximation is very robust over the entire range of model parameters (as shown from validations in Section 3).

A portion of the state space representation of the refined Markov chain model, corresponding to partition p, is shown in Figure 2. In this figure, the index p, indicating the partition, has been omitted, therefore, a state (i, j) in the figure corresponds to the actual state (p, i, j). A state (p, i, j), $0 \leq i \leq L_p$, $0 \leq j \leq \tilde{n}_0 - 1$,

represents an arbitrary buffer page of partition p having a count i and at a distance of j misses from the clock pointer. At any state (p,i,j), $j \neq 0$, upon a miss, there is a transition to the right, i.e., to state $(p,i,j-1)$. At any state (p,i,j), $i \neq 0$, $j = 0$, upon a miss, there is a transition down to the left, i.e., to state $(p,i-1,\tilde{n}_0-1)$. At state $(p,0,0)$, upon a miss, there is transition to state $(p,L_p,\tilde{n}_0-1)$, $1 \leq p \leq P$, with probability $r_p m_p$. At any state (p,i,j), $i \neq L_p$, upon a page request, an arbitrary buffer page of partition p is hit with a probability $\frac{r_p}{S_p}$, thus causing a transition to state (p,L_p,j). Notice that a page hit does not bring it closer to the clock pointer, only resets its count to the hit weight.

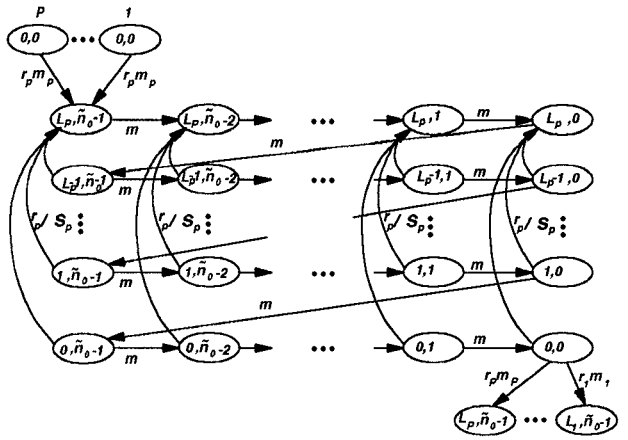

Figure 2: Refined Model

Let $n_{p,i,j}$ be the steady-state average number of pages of partition p having a count i $(0 \leq i \leq L_p)$ and at a distance of j $(0 \leq j \leq \tilde{n}_0 - 1)$ misses from the clock pointer. Then

$$n_p = \sum_{i=0}^{L_p} \sum_{j=0}^{\tilde{n}_0-1} n_{p,i,j}. \tag{6}$$

We denote by $n_{00} = \sum_{p=0}^{P} n_{p,0,0}$ the steady-state average number of buffer pages, from all partitions, having a zero count and at a zero distance from the clock pointer. By definition of the GCLOCK algorithm, $n_{00} = 1$, since only one page with a count zero is replaced upon a miss.

The steady-state probability of being in state (p,i,j) can be expressed as $\frac{n_{p,i,j}}{B}$. As before, we write the balance equations of the Markov chain in terms of, only, $n_{p,i,j}$, $0 \leq i \leq L_p$, $0 \leq j \leq \tilde{n}_0 - 1$, since B cancels out in all balance equations.

Balance at the aggregate states for each of the partitions, yields

$$n_{p,0,0} = \frac{r_p m_p}{m}, \quad 1 \leq p \leq P. \tag{7}$$

Balance at the state (p,i,j), $i \neq L_p$, $j \neq \tilde{n}_0 - 1$, yields

$$n_{p,i,j} = (1+\frac{r_p}{mS_p}) n_{p,i,j-1}, \quad 0 \leq i \leq L_p - 1,\ 1 \leq j \leq \tilde{n}_0 - 1.$$

Balance at the state $(p,i,\tilde{n}_0-1)$, $i \neq L_p$, yields

$$n_{p,i,0} = (1 + \frac{r_p}{mS_p}) n_{p,i-1,\tilde{n}_0-1}, \quad 1 \leq i \leq L_p.$$

From the above balance equations, we get

$$n_{p,i,0} = (1 + \frac{r_p}{mS_p})^{i\tilde{n}_0} n_{p,0,0}, \quad 0 \leq i \leq L_p. \tag{8}$$

let $n_{p,.,j} = \sum_{i=1}^{L_p} n_{p,i,j}$, $0 \leq j \leq \tilde{n}_0 - 1$. Then from the balance equations at the set of states $(p,.,j)$, $0 \leq j \leq \tilde{n}_0 - 2$, we have

$$n_{p,.,j} = n_{p,.,j-1},\ 1 \leq j \leq \tilde{n}_0 - 1.$$

It follows that

$$n_p = \sum_{j=0}^{\tilde{n}_0-1} n_{p,.,j} = \tilde{n}_0\ n_{p,.,0}, \tag{9}$$

where, using Equation (8),

$$n_{p,.,0} = \sum_{i=0}^{L_p} n_{p,i,0} = \frac{(1+\frac{r_p}{mS_p})^{(L_p+1)\tilde{n}_0} - 1}{(1+\frac{r_p}{mS_p})^{\tilde{n}_0} - 1}\ n_{p,0,0}.$$

From Equations (7) and (9) and $m_p = 1 - \frac{n_p}{S_p}$, we finally get the following expression for n_p

$$n_p = \frac{S_p}{1+f_p}, \quad 1 \leq p \leq P, \tag{10}$$

where

$$f_p = \left[\frac{\tilde{n}_0}{m}\ \frac{r_p}{S_p}\ \frac{(1+\frac{r_p}{mS_p})^{(L_p+1)\tilde{n}_0} - 1}{(1+\frac{r_p}{mS_p})^{\tilde{n}_0} - 1}\right]^{-1}.$$

In the above expression for n_p, the quantities m and $\tilde{n}_0$ are not yet known. These can be determined by substituting from Equation (10) in Equation (1). The resulting set of nonlinear equations can be solved iteratively as follows:

1. Start with an initial guess for m (a very good approximation can be obtained from the simple model in Section 2.1)
2. Substituting in Equation (1), we get a nonlinear equation which can be solved for $\tilde{n}_0$ using the bisection method
3. New values for n_p, $1 \leq p \leq P$, can now be computed, and, hence, a new value for $m = \sum_{p=1}^{P} r_p(1-\frac{n_p}{S_p})$. Terminate if the new and old values of m are sufficiently close. Otherwise, repeat from the first step with the new value for m.

The above iterative procedure converges very quickly in practice, particularly, with a good initial guess for m, which can be obtained by solving the first approximate model. The good convergence properties were also obtained for a larger number of partitions and a wide range of weights. The reason is that, independent of the number of partitions and the weights assigned to them, the equations have been analytically reduced to two equations in m and $\tilde{n}_0$ that are solved iteratively.

In the next section we perform experiments to validate the refined model over a wide range of parameters. As will be shown, the results are very accurate (less than 1% of simulation results) in all cases examined. This was also found to be true for a larger number of partitions.

The results in this section hold for equal initial and hit weights of the same partition. The analysis of the refined model is extended in [14] to include the cases where the initial and hit weights may be different. Further, the refined model can be extended to capture multi-node invalidation effect [7, 9]. For example, with broadcast invalidation policies, a page update at one node invalidates copies of that page (if it exists) in the buffers of other nodes. In each node, invalidated buffer pages are collected in a free-page list, which, on a miss, is checked for page replacement. Only, if the free-page list is empty, then the clock pointer advances in search for a (valid) buffer page replacement. The refined model can be easily modified to accommodate this scheme by adding an extra state (say, "free" state) representing the buffer page in the free-page list. A transition may occur from any "valid" state to the "free" state if that page is invalidated by another node. Due to space limitations we do not present details of the analysis for the extended model. However, in Section 4 we will present some validation for the accuracy of this extension. Details of this and other buffer coherency policies will be reported in a forthcoming report.

3 Validation and Performance

In this section we will first validate our analyses by comparison with estimates from a simulation model. A discrete event simulation model is developed for the GCLOCK buffer replacement scheme under IRM. In the simulation there is buffer warm-up period after which statistics are gathered. Each simulation is run long enough so that the 95% confidence interval is within 2% of the mean. We use workload parameters similar to those derived from the TPC-A benchmark [18]. The relative access rate to various data types are fixed under the TPC workload, which results in a particular skewed access pattern. To provide further validation, we will therefore choose workloads with various skewed access patterns [7]. We will then compare the performance of the GCLOCK replacement policy with that of the LRU replacement policy and the optimal buffer allocation policy [1]. Finally, we consider the selection of weights for different partitions in the GCLOCK policy so that its performance approaches that of the optimal allocation policy.

TPC-A is a database benchmark workload for a debit-credit banking application. The database consists of three types of data; namely, BRANCH, TELLER, and ACCOUNT. In addition, there are INDEX data sets that are used to locate any tuples within the database. Each transaction fetches and updates an account record, and the corresponding branch and teller balances. TPC-A specifies the absolute and relative sizes of the data types. While the relative sizes remain constant, the absolute size of each data type increases linearly with the transaction rate. For instance, at 10 transaction/second (TPS), there are 1 million accounts, 1000 tellers, 100 branches and a 90 day history. A hundred byte record is associated with each branch, teller and account. For clarity of presentation, we will assume that only the three kinds of data (INDEX,TELLER and ACCOUNT) are managed using the GCLOCK algorithm. Given the relatively small number of branches, the BRANCH data will rarely be replaced from the buffer even for a small buffer size, and therefore are ignored. Assuming 4K pages, and typical index sizes, the account, index and teller partitions are 25,000, 2500 and 2.5 pages, respectively. In order to have more significant impact of the smallest partition, we will consider a teller partition of size 250 pages (rather than 2.5 pages in the above TPC-A example). Due to the latter assumption, we refer to this as a TPC-A like workload. Note that the access rate to pages in the three partitions is the same for the TPC-A workload.

3.1 Validation

We first use the TPC-A like workload with the parameters given above to validate the models. In the following experiments, it is assumed that the initial and hit counts (weights) are the same. In Figure 3 we show the buffer hit probabilities for the three partitions as a function of the buffer size, for the case where the weight for each of the partitions is zero. Note that this special case is equivalent to the First In First Out (FIFO) buffer replacement scheme, since then each page is replaced after one complete round of the clock pointer regardless of whether the page was re-referenced in the interim. Since the weights are the same, each of the partitions is treated equally. The figure shows the estimates from the approximate, refined and simulation models. For this case, the estimates from both the approximate and the refined models match well with the simulation results. As explained earlier in Section 2,

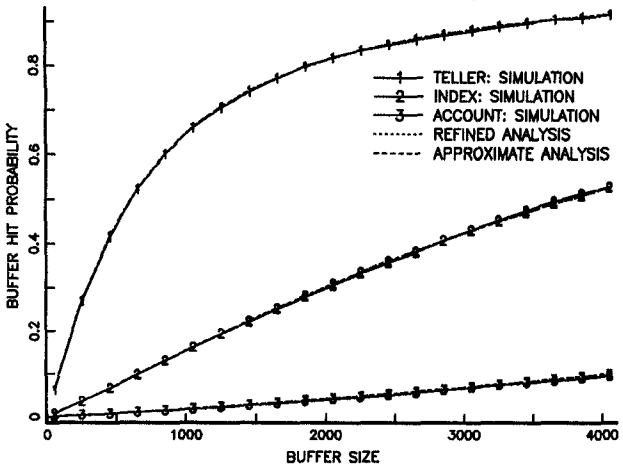

Figure 3: Buffer hit probability with WEIGHTS(0,0,0)

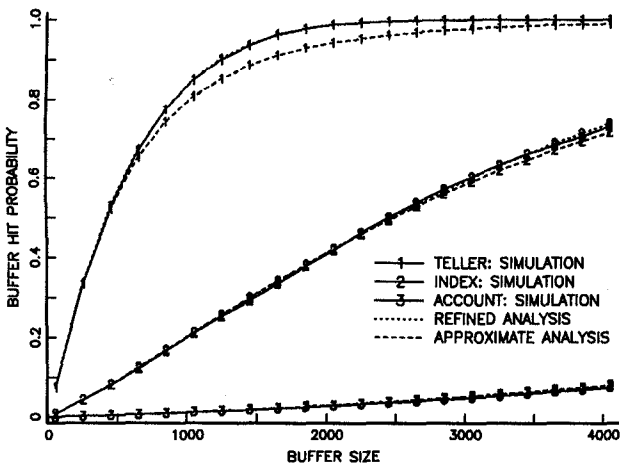

Figure 4: Buffer hit probability with WEIGHTS(1,1,0)

the approximate model does not capture the effect of resetting a page count to a non-zero value if it is a hit in buffer. Note however, that with a weight of zero, in the case of a buffer hit, the count is unchanged at zero. Therefore, the approximate model is also rather accurate for this case. Note from the figure that, with these weights, even the hit probability for the relatively small teller partition with only 250 pages does not saturate (i.e., approach unity) until the buffer is several thousand pages. The index partition does not evidence any saturation in the figure.

Figure 4 is analogous to Figure 3 except that the weights for the teller, index and account partitions are chosen as 1, 1 and 0, respectively. That is, the cold account partition is effectively given a lower priority than the other partitions. The estimates from the simple approximation are pessimistic for the teller partition, while it is reasonably accurate for the larger partitions. The reason for this error, as pointed out in the previous section, is that the approximate model assumes that

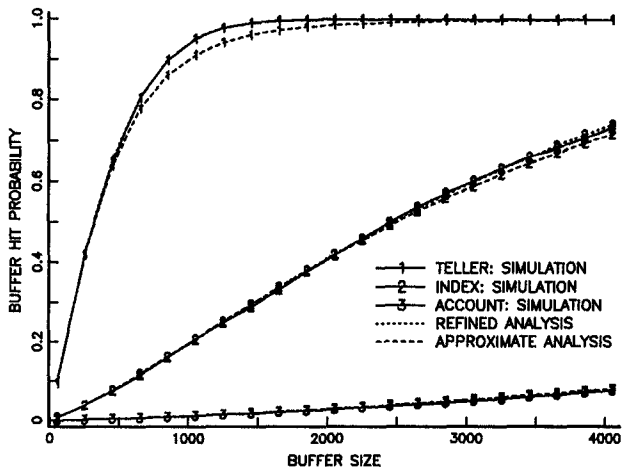

Figure 5: Buffer hit probability with WEIGHTS(2,1,0)

the average number of buffer pages with zero count of a given partition is equal to the average number of replaced buffer pages of the same partition that the clock encounters in one rotation around the buffer. For the small partition, the likelihood that a zero count page is re-referenced, before the clock revisits it, is significant, and therefore the number of zero pages of that partition seen by the clock is significantly smaller than the steady-state average number in the buffer. Therefore, the approximate model estimates that more pages of the small partition are replaced in a clock rotation than what actually occurs, thus leading to a pessimistic estimate of the buffer hit probability. The refined model, by accounting for the above described phenomenon, is more accurate. In fact, it is barely distinguishable from the simulation for each of the partitions. Comparing Figures 3 and 4, we note that, with the higher weights, the buffer hit probabilities of both the teller and index partitions increase significantly, with a small degradation in the account hit probability.

Figure 5 is similar to the previous two figures with the weights for the teller, index and account now set to 2,1,0, respectively. That is, the weights are now set to discriminate between each pair of the partitions. Comparing Figures 5 and 4, the buffer hit of the teller partition improves further, with an imperceptible change in the other two partitions. This is the effect of the increased discrimination between the partitions. Later we will examine whether further discrimination by these weight results in a near optimal performance. From a validation point of view, the agreement between the refined analysis and the simulation is striking.

We now consider other cases of skewed access, where the access frequencies of the partitions are different. In the following experiments we will only show the estimates from the refined model. In Figure 6 we focus on a case with two partitions with 80% of the page

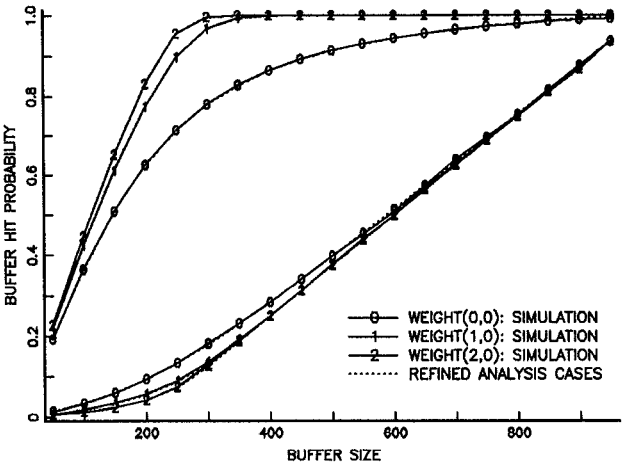

Figure 6: Buffer hit probability for the 80-20 skewed access pattern

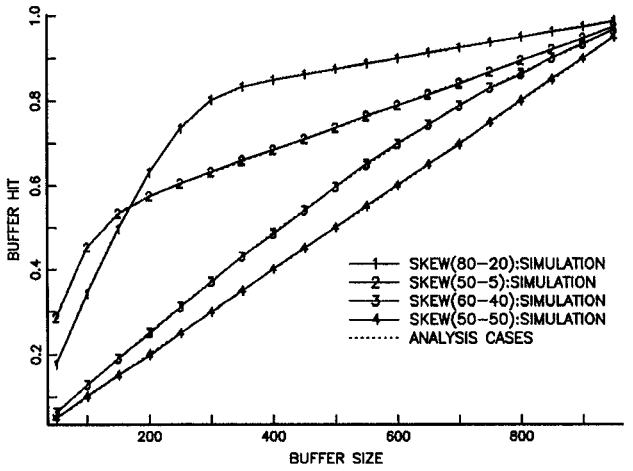

Figure 7: Buffer hit probability for different skewed access patterns

accesses going to 20% of the pages. The number of pages in the database is fixed at 1000 pages. We vary the weights assigned to the two partitions. The legend WEIGHT(X,Y) represents a weight of X to the hot partition and Y to the cold partition. For instance, in the figure, WEIGHT(2,1) represents a weight of 2 to the hot partition and a weight of 1 to the cold partition. The figure shows that an increased weight to the hot partition results in better discrimination between the partitions. Consequently, we see an increasing hit probability for the hot partition, and a corresponding but smaller decrease in the hit probability to the cold partition. Again, there is excellent agreement between the estimates from the refined analysis and the simulation.

In Figure 7, the overall buffer hit probability versus buffer size is shown for four cases of skewed access. The number of pages in the database is again fixed at 1000 pages. The legend of this figure uses the notation of

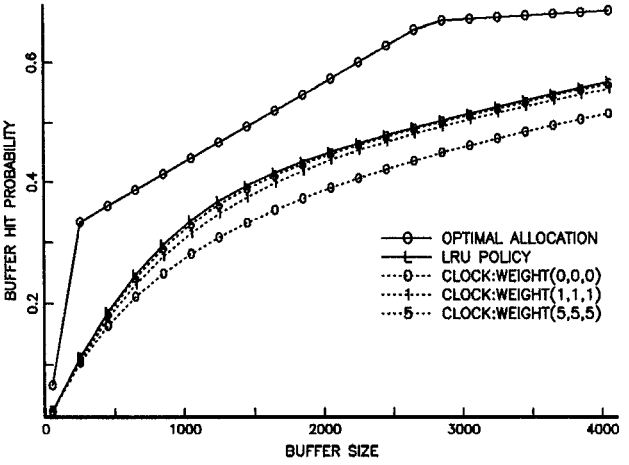

Figure 8: Comparison with LRU and Optimal Allocation with Equal Weights to Partitions

SKEW(X-Y) to represent that X% of the accesses go to Y% of the pages. For instance, in the case labelled SKEW(80-20), 80% of the page requests are to 20% (i.e., 200 of the 1000 database pages). In all the curves, the hot partition is given a weight of 1, and the cold partition is given a weight of 0. For all of these cases, the estimates from the refined analysis are almost indistinguishable from the simulation. The higher skew cases (i.e., SKEW(80-20) and SKEW(50-5)) clearly exhibit two regions with different slopes, corresponding to saturation of the hottest partition.

3.2 Comparison with LRU and Optimal Static Allocation

We now compare the buffer hit probability using the GCLOCK algorithm with that of the LRU buffer replacement and optimal allocation policies. Figure 8 is for TPC-A like parameters used in Figures 3 through 5. The figure shows the overall buffer hit probability for the GCLOCK algorithm with equal weights for all partitions (set to 0, 1 and 5). Also shown on the figure is the buffer hit probability for the LRU policy and optimal buffer allocation. For optimal buffer allocation, the entire buffer is allocated to the hottest partition (the teller partition in this case) until the buffer size equals the size of the hottest partition. After this point, which we will refer to as the first break-point, additional buffer space is allocated to the next hottest partition (the index partition in this case), until this partition fits in buffer, and so on. This gives rise to a series of break-points at which the slope of the buffer hit probability versus buffer size characteristic changes. The optimal allocation characteristic represents a bound on the performance of the more general buffer allocation policies.

In Figure 8, we note that, with equal weights of zero assigned to the different partitions, the overall buffer hit

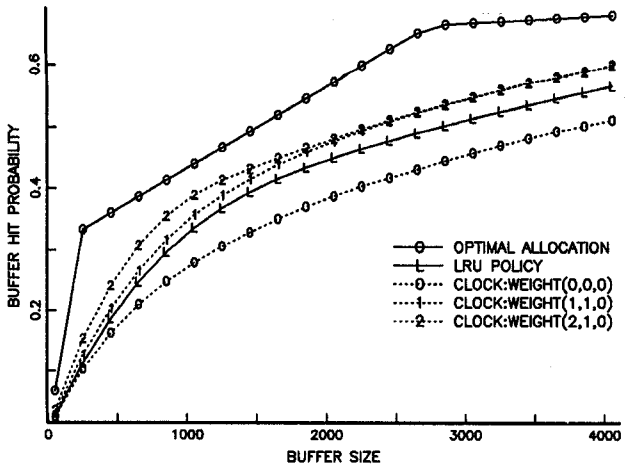

Figure 9: Comparison with LRU and Optimal Allocation with Different Weights to Partitions

probability of the GCLOCK algorithm is significantly worse than that of the LRU policy. With the weights at 1 for each partition, the buffer hit probability improves considerably and is slightly worse than LRU. The reason is that, although the weights are equal, when the hotter partition pages are re-referenced, their count is reset to being greater than zero and therefore they are not replaced in the next clock sweep. For weights of zero, a larger hit probability does not give the hot partitions any advantage. As the weight for all partitions is increased further to five, the buffer hit probability gets closer to LRU, as shown in the figure. For larger weights, the curves for the GCLOCK policy approach that of the LRU policy.

We now turn to the case with different weights to different partitions. Figure 9 shows the overall buffer hit probability for the GCLOCK algorithm with the weights used in figures 3 through 5. The overall buffer hit probability is better than that of the LRU policy, except for the case with zero weights. The reason is clearly that the higher weights discriminate between the partitions to some degree, and tend to keep a larger fraction of the hot partitions in the buffer than with equal weights. However, comparing these cases with the performance of optimal buffer allocation shows that, while the different weights improve performance, the overall buffer hit probability even with the weights set at (2,1,0) is far from optimal. This is particularly so at the breakpoints.

3.3 Weight Assignment

The deviation from optimal allocation for the GCLOCK algorithm raises the question as to whether the weights of the different partitions can be chosen for close to optimal performance. We use the refined analysis to examine this question. The objective is to determine

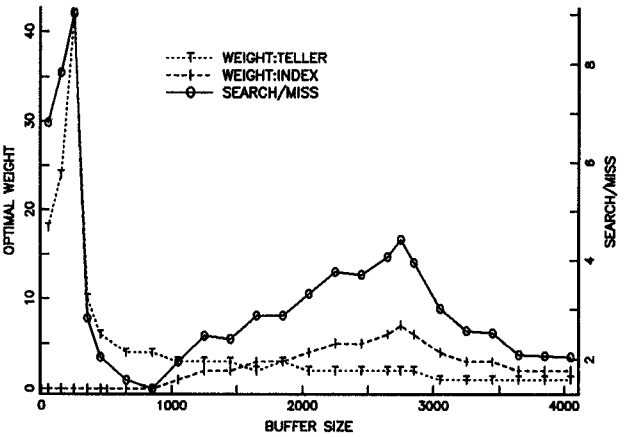

Figure 10: Weight assignments for performance within 90% of Optimal Allocation

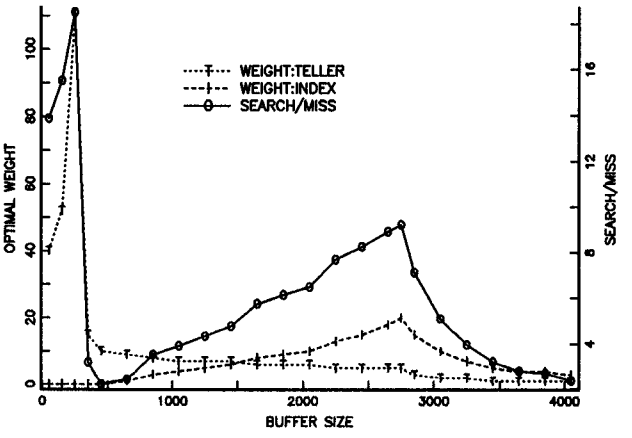

Figure 11: Weight assignments for performance within 95% of Optimal Allocation

partition weights that result in a buffer hit probability that is within some tolerance of that for optimal buffer allocation. Let $(W_1, ..., W_P)$ denote the weights assigned to the different partitions. For any buffer size, we start with an equal weight for each partition. We compute the buffer hit probability for the weight assignments $(W_1, ..., W_i + 1, ..., W_P)$, $1 \leq i \leq P$, using the refined analysis. We then pick the partition i that leads to the largest buffer hit probability and increment its weight. If the estimated average buffer hit probability is within the desired tolerance from that for optimal allocation, the procedure is terminated. Otherwise, the procedure is repeated.

Figure 10 shows the weights determined using this method for the TPC-A like case (considered in Figures 3 through 5), where the overall buffer hit probability is required to be greater than 90% of that using optimal buffer allocation. Referring back to Figure 9, the deviation of the buffer hit probability for the GCLOCK algorithm from that of optimal allocation is large for small buffer sizes and for the weights chosen. Therefore, for small buffer sizes, a larger weight is needed for the hottest partition. Figure 10 indeed shows this to be the case. For buffer sizes less than the first break-point a large weight must be assigned to the hottest partition (i.e., the teller partition in this case). At the first break-point, the weight that must be assigned to the hottest partition is surprisingly large (about 40 in this case). The explanation is that for optimal allocation, the entire buffer must be allocated to the hottest partition, and in order to approach this, the GCLOCK policy needs an inordinately large weight for this partition. Beyond the first break-point, the weight that needs to be assigned to the hottest partition falls sharply. For instance, for a buffer size of 500 pages (i.e., twice the size of the hottest partition), a weight of 4 needs to be assigned to this partition to meet the objective of a buffer hit probability greater than 90% of that for optimal allocation. As the buffer size increases further, the weight assigned to the hottest partition falls slowly.

Turning now to the weight assigned to the second hottest (i.e., the index partition), Figure 10 shows that for smaller buffer sizes, this partition is assigned a low weight. This is because the hottest partition must be well discriminated from the next hottest partition until a little beyond the first break-point. As the buffer size increases, a larger weight must be assigned to this partition, and the assigned weight peaks at the second break-point (buffer size of 2750). An interesting aspect is that the weights assigned to the hottest and the next hottest partition cross beyond a certain buffer size, as the figure shows. The reason is that, beyond a certain buffer size, the hottest partition easily fits in the buffer, and due to re-references continues to reside in the buffer. For example, with 250 teller pages and a buffer size of 2000, the teller page is very likely to be re-referenced before the clock gets back to it and it is no longer necessary to give it a high weight. By comparison, for this buffer size, the 2500 page index partition does not fit in the buffer, and a high weight must be assigned to the index partition in order to favor it over the account partition. Note also from Figure 10 that the weights assigned at the second break-point are much lower than that of the first break-point. The explanation is that for the same tolerance (relative to the optimal) the clock rotates at a slower rate for larger buffer sizes, thus allowing for the same discrimination with a lower weight.

One potential problem with the clock algorithm, when high weights are assigned, is that on a buffer miss the clock must search through many buffer pages before finding a page with a zero count. This could lead to large overheads. Therefore, in Figure 10 we have also shown the average number of buffer pages that must be searched for each buffer miss. Notice that the number of pages that must be searched per miss increases significantly at the break-points where a large weight is assigned to some partitions. However, the average number of pages searched per miss is less than 10 for all cases. Since a search merely involves accessing and updating a data structure in memory, and a comparison, the overhead per search is a few instructions, and is probably acceptable.

Figure 11 is similar to Figure 10 except that the objective is that the buffer hit probability should be greater than or equal to 95% (rather than 90%) of that for optimal allocation. The general trends are the same as in the previous figure, except that the weights required are much larger (about twice) the previous weights, particularly at the break-points. The reason is that tighter tolerance requires more discrimination between the partitions and consequently larger weights. However, the figure also shows that the overhead in terms of the number of buffer pages searched per miss approximately doubles, as compared to the previous case.

The overall buffer hit probability resulting from the weight assignments of the previous two figures, along with that for optimal allocation and the LRU policy, is shown in Figure 12. The figure shows the tolerance objectives, relative to optimal allocation, being met by the weights selected for the GCLOCK algorithm. Changes in weight assignment in Figures 10 and 11 cause jumps in the buffer hit probabilities in Figure 12, however, the tolerance objectives are still met.

These experiments lead to the question of whether weights can be assigned to the GCLOCK scheme so as to bring its performance arbitrarily close to that of the optimal allocation policy. The problem with attempting this is that the weights assigned become very large at the breakpoints, and that the overhead in terms of the searches per miss also shoots up at the break-points.

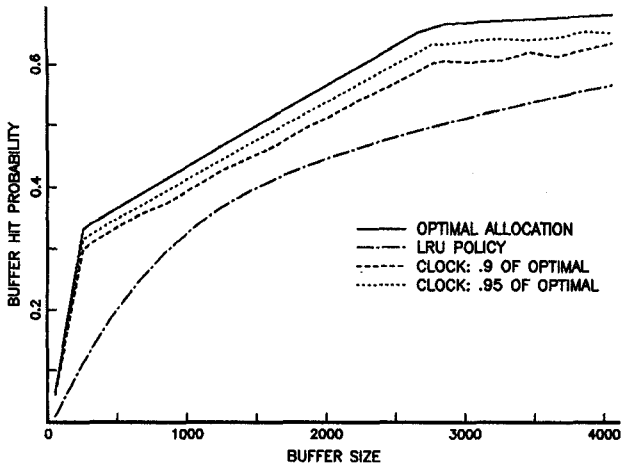

Figure 12: Buffer Hit with Optimal Weight Assignments

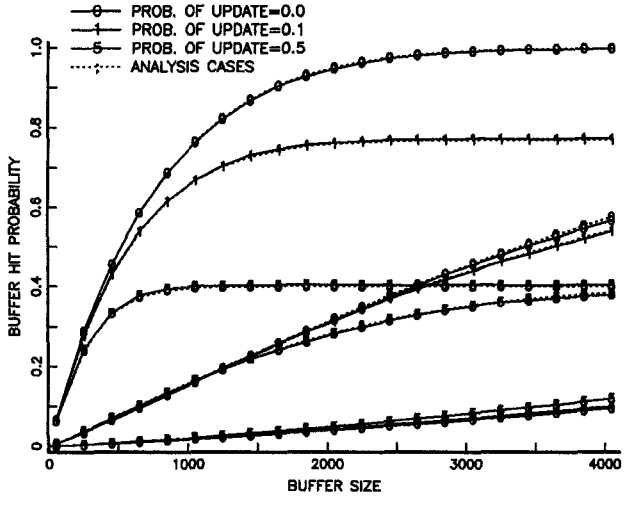

Figure 13: Validation of Broadcast Invalidation Policy

3.4 Multi-Node Extension

Finally, we validate the extension of the refined model to capture the broadcast invalidation policy in a multi-node system. Figure 13 shows the buffer hit probability for the three partitions in the TPC-A like workload, with weights set at (1,1,1). The probability of update per access is assumed to be the same for all partitions. The figure corresponds to a system with four nodes and different probabilities of update; namely, 0.0, 0.1, and 0.5. Note that the case of zero update probability implies no buffer invalidation and, therefore, is the same as the single node case. As the update probability increases, the buffer hit probability of the hot partition decreases and saturates at a considerably lower buffer hit probability. The buffer hit probability of the other partitions also decreases, but to a lesser extent.

4 Summary and Conclusions

In this paper we have developed approximate analytical models for the GCLOCK buffer replacement policy, under IRM. An exact model for a reasonable size buffer becomes intractable because a very large number of states is required. We first developed a simple approximate model that has a number of states equal to the sum of the (maximum weight + 1) of all partitions. This model ignores the distance of a buffer page from the clock pointer. If there is a large difference between the sizes and access frequencies of the partitions, then this simple model underestimates the buffer hit probability of the hottest partitions. A refined model is developed that approximately takes into account the distance of a buffer page from the clock pointer, and has the number of states as the product of the sum of the (maximum weight + 1) of all partitions and the average number of misses in one rotation of the clock pointer around the buffer. In the analysis of this model, the initial and hit weights assigned to a partition may be different. By comparison with simulation results, this refined model is shown to be very accurate across a wide range of parameters. In fact, for all cases examined the estimates from the refined model are virtually indistinguishable from simulation results.

We apply the analysis to a case similar to the TPC-A benchmark, and compare the performance of the GCLOCK policy with that of the LRU policy and optimal (static) buffer allocation. With zero weights assigned to all partitions, the performance of the GCLOCK algorithm is significantly worse than that of the LRU policy. With equal weights assigned to all partitions, the performance of the GCLOCK policy approaches that of the LRU policy as the weight is increased. If knowledge of the partitions in the workload is assumed, weights can be assigned to the partitions in the GCLOCK policy so that its performance is better than that of the LRU policy. We use the refined analysis to determine partition weights that result in a buffer hit probability within some tolerance of that for optimal buffer allocation. For sufficiently small tolerance objectives, large weights need to be assigned to the hot partitions. This is particularly true at the break points of the optimal buffer allocation policy. At these points, a weight of more than 100 may be required in some cases. These large weights can lead to a larger overhead for the GCLOCK policy, in terms of the number of pages that must be searched on a buffer miss. This overhead was also estimated, and shown to be modest for a tolerance objective of 90% of that for optimal buffer allocation. Tighter tolerances can result in unacceptable overheads. In practice, this limits how close the buffer hit probability for the GCLOCK policy can be made to approach that of optimal buffer allocation.

The models considered in this paper can be extended to capture the multi-node invalidation effect. We outlined the extension to model the broadcast invalidation policy, and illustrated its accuracy. We are currently investigating the performance of other buffer coherency policies using further extensions of the refined model.

Acknowledgement

We would like to thank Scott Leutenegger for useful discussions.

References

[1] Belady, L. A., "A Study of Replacement Algorithms for a Virtual Storage Computer", *IBM Systems Journal*, Vol.5, No.2, 1966, pp. 78–101.

[2] Carr, R. W., and J. L. Hennessy, "WSClock - A Simple and Effective Algorithm for Virtual Memory Management", *ACM SIGMETRICS*, 1981, pp. 87–95.

[3] Casas, R. I., and K. C. Sevcik, "A Buffer Management Model for Use in Predicting Overall Database System Performance", *5th International Conference on Data Engineering*, Los Angeles, CA, Feb. 1989, pp. 463–469.

[4] Chou, H. T., and D. J. Dewitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems", *11th International Conference on Very Large Databases*, Stockholm, Sweden, 1985, pp. 127–141.

[5] Corbato, F. J., "A Paging Experiment with the Multics System", *MIT Project MAC Report MAC-M-384*, May 1968.

[6] Dan, A., and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes," *ACM SIGMETRICS*, Denver, CO, May 1990, pp. 143–152.

[7] Dan, A., D. M. Dias, and P. S. Yu, "The Effect of Skewed Data Access on Buffer Hits and Data Contention in a Data Sharing Environment," *16th International Conference on Very Large Databases*, Brisbane, Australia, Aug. 1990.

[8] Dan, A., D. M. Dias, and P. S. Yu, "Modelling a Hierarchical Buffer for the Data Sharing Environment", *ACM SIGMETRICS*, San Diego, CA, May 1991.

[9] Dan, A., and P. S. Yu, "Performance Comparisons of Buffer Coherency Policies," *11th International Conference on Distributed Computing Systems*, Arlington, TX, May 1991.

[10] Dan, A., P. S. Yu and J. Y. Chung, "Characterization of Database Access Skew of a Transaction Processing Environment", IBM Research Report RC 17436, Yorktown Heights, NY, Sept. 1991.

[11] Effelsberg, W. and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. Database Systems*, Vol. 9, No. 4, Dec. 1984, pp. 560-595.

[12] Kearns, J. P., and S. Defazio, "Diversity in Database Reference Behavior", *Performance Evaluation Review*, Vol. 17, No. 1, pp. 11-19.

[13] King, W. F., "Analysis of Paging Algorithms", *Proc. IFIP Congress*, Ljublanjana, Yugoslavia, Aug. 1971, pp. 485-490.

[14] Nicola, V. F., A. Dan, and D. M. Dias, "Analysis of the Generalized Clock Buffer Replacement Scheme for Database Transaction Processing", IBM Research Report RC 17225, Yorktown Heights, NY, Sept. 1991.

[15] Reiter, A., "A Study of Buffer Management Policies for Data Management Systems", *Technical Summary Report 1619*, Mathematics Research Center, Univ. of Wisconsin, Madison, March 1976.

[16] Sacco, G. M., and M. Sckolnick, "Buffer Management in Relational Database Systems", *ACM Trans. Database Systems*, Vol. 11, No. 4, Dec. 1986, pp. 473-498.

[17] Smith, A. J., "Sequentiality and Prefetching in Database Systems", *ACM Trans. Database Systems*, Vol. 3, No. 3, Sept. 1978, pp. 223-247.

[18] "TPC Benchmark A Standard Specification", *Transaction Processing Council*, 1989.