VLDB Journal, 4, 127-154 (1995), Wesley Chu, Editor

Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability

Asit Dan, Philip S. Yu, and Jen-Yao Chung

Received August, 1994; accepted February, 1994.

Abstract. The analytic prediction of buffer hit probability, based on the characterization of database accesses from real reference traces, is extremely useful for workload management and system capacity planning. The knowledge can be helpful for proper allocation of buffer space to various database relations, as well as for the management of buffer space for a mixed transaction and query environment. Access characterization can also be used to predict the buffer invalidation effect in a multi-node environment which, in turn, can influence transaction routing strategies. However, it is a challenge to characterize the database access pattern of a real workload reference trace in a simple manner that can easily be used to compute buffer hit probability. In this article, we use a characterization method that distinguishes three types of access patterns from a trace: (1) locality within a transaction, (2) random accesses by transactions, and (3) sequential accesses by long queries. We then propose a concise way to characterize the access skew across randomly accessed pages by logically grouping the large number of data pages into a small number of partitions such that the frequency of accessing each page within a partition can be treated as equal. Based on this approach, we present a recursive binary partitioning algorithm that can infer the access skew characterization from the buffer hit probabilities for a subset of the buffer sizes. We validate the buffer hit predictions for single and multiple node systems using production database traces. We further show that the proposed approach can predict the buffer hit probability of a composite workload from those of its component files.

Key Words. Database access characterization, access skew, sequential access, reference trace, workload management, analytic prediction.

1. Introduction

In a relational database environment, accesses to the database come from various application sources. There are many short transactions that read and/or write a

Asit Dan, Ph.D., Philip S. Yu, Ph.D., and Jen-Yao Chung, Ph.D. are Research Staff Members, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, *asit@watson.ibm.com*.

small number of pages. There are also long queries that may sequentially access a large number of pages from one or more database relations. In addition, within both short transactions and long queries some of the pages are rereferenced, and are called *locality sets* (Chou and Dewitt, 1985). Viewed as a whole, the combination of transaction and query accesses generates all possible access patterns (Rodriguez–Rosell, 1976; Smith, 1978; Hawthorn and Stonebraker, 1979; Effelsberg and Loomis, 1984; Chou and Dewitt, 1985; Verkamo, 1985; Sacco and Schkolnick, 1986; Kearns and Defazio, 1989). Traditional buffer management policies (e.g., strict LRU policy that does not exploit the knowledge of access components such as sequential vs random accesses) may not provide good buffer hit probability (Smith, 1978; Effelsberg and Haerder, 1984; Teng and Gumaer, 1984; Chou and Dewitt, 1985; Sacco and Schkolnick, 1986; IBM, 1993). Alternatively, if the various access components can be identified and categorized, the information can be used not only to design better buffer management policies but also to predict the buffer hit probabilities. The knowledge of access patterns can also be useful to a buffer management policy that exerts control over the buffer space to provide different buffer hit probabilities to transactions and queries. It is the goal of this article to provide a characterization method that can be used for the above purpose.

In an earlier study on database reference traces (Kearns and Defazio, 1989), it was shown that the database access pattern of each transaction type and file changes very little over time. Kearns and Defazio (1989) collected traces over five working days, and found the degree of stability notable, even for the least stable transaction types. Therefore, the characterization of access pattern in the traces from representative workload can be used for analytic prediction of buffer hit probability in various system configurations. For example, existing analytic models for the LRU (Dan and Towsley, 1990) and Clock (Nicola et al., 1992) replacement policies can be used to predict the buffer hit probabilities of multiple relations sharing the same buffer pool, given the access characterizations of individual relations. Such buffer hit prediction capability will be extremely useful for both workload management as well as system capacity planning in various ways. The knowledge can be helpful for proper allocation of buffer space to various database relations as well as the management of buffer space for a mixed transaction and query environment (Faloutsos et al., 1991; Ng et al., 1990; Yu and Cornell, 1991). In a multi-node environment, the access characterization of individual relations can also be used to predict the effect of cross-node buffer invalidation which can influence transaction routing strategies. However, it is a challenging problem to characterize the database access pattern based on a real workload reference trace in a simple manner that can easily be used to compute the buffer hit probability.

There have been many earlier analytical buffer modeling works that assumed skewed access pattern based on independent reference model (IRM) for a transaction processing workload (Dan and Towsley, 1990; Dan et al., 1994*a*, 1994*b*; Nicola et al., 1992). Ability to characterize the model parameters from workload traces will make these models applicable to real environment. In a mixed transaction and

query environment, workload can not be characterized directly by the IRM model. Therefore, we separate the non-IRM and IRM access components. More specifically, our characterization method first distinguishes three types of access patterns from a trace: (1) *locality within a transaction,* (2) *random accesses by transactions,* and (3) *sequential accesses by long queries.* The presence of these types of access behavior has been shown (Rodriguez–Rosell, 1976; Hawthorn and Stonebraker, 1979; Effelsberg and Loomis, 1984; Verkamo, 1985; Kearns and Defazio, 1989). Each access component can be accounted for separately, both in terms of buffer management as well as buffer hit prediction. The overall buffer hit probability for transactions or queries is then given by the weighted average of the buffer hit probabilities of their access components.

Here we assume that the buffer manager will use *prefetching* for the sequentially accessed pages, and the LRU replacement policy for buffer management for the remaining pages. Prefetching provides very high buffer hit probability for sequential accesses. The prefetched pages are assumed to be discarded after they are used[1] and, hence, they do not affect the performance of the LRU policy for the randomly accessed pages. In DB2 (Teng and Gumaer, 1984), they were placed in a separate list from the normal LRU chain. As the rereferenced pages of the short transactions are also expected to be found in the buffer since transaction working set size generally is much smaller than most database buffer sizes, the synchronous database I/Os will come mainly from the random accesses. The buffer hit prediction for the random access component is a non-trivial task. Our main focus here is the characterization of random accesses for the prediction of buffer hit probability. The random access is not uniform over the entire database. For example, in a banking application (TPC-A workload; Gray, 1991) whenever an account record is updated, a branch total associated with this account is also updated. Therefore, each branch record is accessed more often than each account record. Furthermore, some accounts may be updated more often than others. Also, if the database pages are accessed through an index, the index pages are accessed more often than the data pages. The resulting non-uniform access pattern is referred to as *random access skew.*

In several earlier analytical models for transaction processing (Tay et al., 1985; Dan and Towsley, 1990; Dan et al., 1991*a*, 1994; Yu et al., 1993; Nicola et al., 1992), access skew was modeled by assuming that the database is divided into a small number of logical partitions (e.g., Hot-set Cold-set model) and the probability of accessing any page within a partition is the same. To make these models useful for

1. Chou and Dewitt (1985) and Sacco and Schkolnick (1986) proposed various query access models. Further knowledge of the query access pattern from the query optimizer may be used by the buffer manager to reduce prefetching I/O and/or buffer space requirement. However, this kind of top-down approach using the query plan and the optimizer information for buffer management (Ng et al., 1990; Faloutsos et al., 1991; Cornell and Yu, 1989; Yu and Cornell, 1991) is beyond the scope of this article, which uses a bottom-up approach based on the access trace information to predict buffer hit probability.

real environment, we follow the above skew characterization method (i.e., we assume that the large number of data pages can be logically grouped into a small number of partitions such that the frequency of accessing each page within a partition can be treated as equal). This requires a small number of parameters to express the access skew. More specifically, the access skew in a trace is characterized by the number of partitions, and the access frequency and size of each of the partitions. For example, 80-20 access rule can be represented by 2 partitions where 80% of the accesses go to smaller partition (i.e., 20% of the database). Based on this approach, we present a recursive binary partitioning algorithm that can infer the access skew characterization from the buffer hit probabilities for a subset of the buffer sizes. This avoids explicit estimation of individual access frequencies for the large number of database pages. The buffer hit prediction based on this skew characterization is also efficient.[2]

Therefore, our skew characterization method satisfies all three objectives that guide the characterization process: 1) *economy of expression* (very few parameters are needed to express the skew) 2) *ease of skew characterization* and 3) *efficient estimation of buffer hit probability based on the characterization.* Further discussions on alternative skew characterization methods can be found in (Dan et al., 1991).

In Section 2, we describe the composition of a database workload that is used for this study in terms of access patterns (e.g., random, sequential, rereference). Section 3 presents the details of the skew characterization algorithm that characterizes the skew in an access trace. Section 4 provides the validation of buffer hit prediction capability both for the overall database access characterization and for the random access skew characterization. We summarize the results in Section 5.

2. Database Workload Description

Two production database workloads are employed to validate our characterization approach. The first trace was taken on a DB2 system (Date and White, 1989; IBM, 1993) from a long distance communication company during the peak activity period for a duration of 2 hours and 30 minutes (2:30 pm to 5 pm). There are 10.7 million page accesses recorded on the trace. We describe this workload in detail since it will be the primary one for presentation in this article. A second DB2 trace from a commercial bank is also analyzed. It was taken at peak load for an hour with 13.3 million page accesses recorded, and it contains a substantially larger number of updates. It will be introduced for the validation of the predictive capability of the multi-node buffer invalidation effect. The buffer traces consist of a sequence of Getpage and Setwrite records, where a Getpage is recorded each time a page

2. If the database of size D is logically divided into K partitions, the computational complexity of estimating the buffer hit probability for a buffer size of B is $O(KB)$ under the LRU replacement policy (Dan and Towsley, 1990; Dan et al., 1994), where $K << D$.

Table 1. Size, access count, and access components of first database trace

	File size	*Access count*	*Sequential*	*Rereference*	*Random*
R1	170700 (9.50%)	1501951 (14.01%)	0.00%	24.74%	75.26%
R2	36300 (2.02%)	552874 (5.16%)	92.84%	3.64%	3.52%
R3	60560 (3.37%)	402460 (3.75%)	80.27%	6.89%	12.84%
R4	83420 (4.64%)	316706 (2.95%)	0.00%	22.56%	77.44%
R5	15300 (0.85%)	203586 (1.90%)	0.00%	43.90%	56.10%
R6	12900 (0.72%)	136189 (1.26%)	0.13%	37.20%	62.67%
R7	234600 (13.06%)	95332 (0.89%)	0.03%	6.79%	93.18%
	1065865 (59.34%)	6308434 (41.15%)	*Relation Summary*		
I1	39875 (2.22%)	807981 (7.54%)	0.00%	87.86%	12.14%
I2	7839 (0.44%)	461464 (4.30%)	0.00%	79.05%	20.95%
I3	22076 (1.23%)	206942 (1.93%)	51.19%	33.09%	15.72%
I4	26491 (1.47%)	91991 (0.86%)	14.22%	62.33%	23.45%
	730368 (40.66%)	4410865 (58.85%)	*Index Summary*		
	1796233 (100.00%)	10719299 (100.00%)	*Relation & Index Summary*		

reference request is made to the buffer manager from the data manager in DB2, and a Setwrite is recorded each time the buffer manager is informed that a page update is requested to the data manager. These records contain information on the database relation, page ID, process ID (also known as ACE or Agent Control Element in DB2), and timing information. The system maintains some fixed number of processes. An incoming transaction is executed by one of these processes. The same process executes an entire transaction. Therefore, the process ID can be used to track the access string of a particular transaction.

In the first trace, the database consists of 117 relations and 138 index files. Temporary work files are ignored in this study. Only 62 relations and 91 index files were accessed during the tracing period. These are referred to as the active files. Table 1 provides a summary of file size, access count, and fraction of various access components (random, rereference, and sequential) in each relation or index file for the most active ones among these files. We shortly define the access components and describe the algorithm that was used to identify each of the components in the trace driven simulations. Also indicated in parentheses in the file size column is the fraction relative to the total size of the active files. Similarly indicated in the access count column is the fraction relative to the total number of page accesses.

The access trace for each file (relation or index) consists of concurrent page accesses by multiple transactions or queries to the same file. For the identification of sequential accesses, the access string from each process (transaction or query) is tracked separately in the trace driven simulation based on the process ID. The query optimizer often provides explicit hints of sequential accesses to the buffer manager for the purpose of prefetching as well as separate handling of pages brought in by sequential accesses (Teng and Gumaer, 1984). Unfortunately, such hints were not recorded in the particular traces that we processed. Note that the primary objective for characterizing an access trace is to predict the buffer hit probability under the specific buffer management policy. Therefore, the characterization process should capture the behavior of the buffer management policy. In different environments, different methods may be used for identifying and, hence, prefetching sequential accesses. Here, we assume an environment where a simple dynamic prefetching algorithm is used by the buffer manager to identify the sequential accesses. The algorithm uses *run length* to identify sequential accesses and then triggers prefetch. An access is counted as a part of an ongoing run if it is to the same or the next page relative to the page accessed in the previous step (Smith, 1978). Once a sequential run up to some prespecified threshold (N_T) is detected, the subsequent references are considered to be part of a sequential access string. We assume that the dynamic prefetching will then be activated by the buffer manager. We further assume that N_P pages are brought into the buffer due to each prefetch. Prefetching for a process is deactivated once a break in the run is detected. We like to emphasize here that our goal is not to design an optimal prefetching policy or to determine the best way to allocate buffer space among sequential and random access components (i.e, query and transactions), but to characterize such access components for use in future buffer hit prediction.

The behavior of the buffer management policy depends on the values of control parameters N_T and N_P. Too small a value for N_T will identify most access strings as sequential, and will cause many false prefetches, while too large a value will miss most sequential accesses. The choice of N_P value for small values of N_T is also an important factor in determining minimum false prefetch I/O overhead (Smith, 1978). We will seek a large enough N_T to avoid substantial false prefetch overhead, while not affecting the buffer hit probability of the (what we identify) non-sequential component. Figure 2 shows the effect of recognizing and separating the sequential component on the buffer hit probability of the remaining component. Different curves correspond to different values of N_T, where the sequential access component is identified once an ongoing run of N_T is detected. Only references with run lengths less than N_T are kept in the main buffer and the rest of the pages (identified as sequential component) are put into a separate buffer. The pages needed by the sequential component are prefetched, and therefore will be found in the buffer as needed (i.e., there is no synchronous I/O requirement). However, if the prefetch pages are not buffered in a separate buffer space or through a separate LRU chain as in SLRU (Teng and Gumaer, 1984), a long sequential scan can wipe

out the frequently accessed pages by the transactions from the buffer.

The effect of the sequential access component on the buffer hit probability of the other access components is examined later in Figure 3. We assume that the small set of prefetched pages will be put into a separate buffer, called *prefetch buffer.* (This is similar to the separate logical chain in DB2 buffer; Teng and Gumaer, 1984). The size of the prefetch buffer only needs to be large enough to keep the number of prefetched pages (N_P) to get actually referenced during a sequential scan. (Here the prefetch buffer is not intended to capture rereferencing after the sequential scan. Optimizing buffer allocation for join queries to reduce the number of prefech I/Os is beyond the scope of this article (see Ng et al., 1990; Faloutsos et al., 1991; Yu and Cornell, 1991). This has a very small effect on the buffer hit probability of the non-sequentially accessed pages.[3] As shown in the figure, too large an N_T value (> 100) is not effective in identifying the sequential component. However, the buffer hit probability of the non-sequential component is not very sensitive to the smaller values of N_T. Our objective here is not how to select the best values of N_T and N_P, but to demonstrate the effectiveness of characterization process for any reasonable values of N_T and N_P. Therefore, we choose arbitrarily N_T and N_P to be 10.

Figure 3 shows the buffer hit probabilities of the sequential and non-sequential components after the sequential component is identified, and/or prefetched, and/or put into a separate prefetch buffer. The three solid curves are the buffer hit probabilities (random, sequential, and overall) for the case when the prefetching is not activated, and both the sequential and non-sequential components are put into the same buffer. The dashed curves are the corresponding buffer hit probabilities after the sequential component is prefetched but still put into the same buffer. The buffer hit probability of the sequential component becomes 1.0, but that of the non-sequential component changes very little. If the sequential component is put into a separate prefetch buffer, then the buffer hit probability of the non-sequential component improves substantially (dotted curve). Hereafter in this article, we assume that the buffer manager uses a separate prefetch buffer.

The non-sequential component is further divided into two components: (1) rereferenced pages within the locality set and (2) randomly accessed pages. Within a transaction a page may be rereferenced several times as a result of the executions of multiple SQL statements referencing the same page or same tuple. For example, in the TPC-A benchmark (Gray, 1991) an account is first updated (UPDATE statement), and then the result on that account is reported through a separate SELECT statement. Rereferencing can also occur in the small loops of the query

3. Note that in the simulation, a page is never replicated in two separate buffer spaces. If a sequentially accessed page is already present in the main buffer, the page is neither prefetched nor moved to the prefetch buffer. However, if a random access falls on a page present in the prefetch buffer, it will be moved to the main buffer.

access string that would not be identified as sequential accesses. The rereferenced pages within the small locality sets are highly likely to be found in the buffer even for very small buffer sizes. Therefore, for the estimation of buffer hit probability, this access component needs to be separately accounted for. Also, it will be shown later that rereferencing within a transaction has to be distinguished for accurate prediction of invalidation effect in a multi-system environment. Let N_W be the size of a transaction (i.e., the number of pages accessed by a transaction). The transaction boundaries were not recorded in this particular trace. Experimentation with the trace showed that most of the rereferences occurred within a window of size 10. Therefore, we arbitrarily chose the transaction size, (N_W), to be 10.

The remaining accessed pages are assumed to be random, particularly when they come from several concurrently executing transactions. However, the random access is not uniform over all database pages. Figure 4 shows the buffer hit probabilities of the rereference and random components for three relations and one index file in the first database workload. The rereference hit probability is 1.0 for most buffer sizes. However, the buffer hit probabilities of the random components are very different for the different relations and index files. The random accesses are clearly skewed, as buffer hit probability does not increase uniformly with the increase in buffer size. In the next section, we will provide an algorithm that can determine the access skew in the randomly accessed pages.

Given the fractions of various access components, and the characterization of the random access skew under the above described buffer management policy, the overall buffer hit probability can easily be computed. Let λ_S, λ_L, and λ_R be the access ratios for the sequentially accessed pages, rereferenced pages within locality sets, and the randomly accessed pages, respectively. Let h_S, h_L, and h_R be the corresponding buffer hit probabilities. Then the overall buffer hit probability is given by

$$H = \frac{\lambda_S h_S + \lambda_L h_L + \lambda_R h_R}{\lambda_S + \lambda_L + \lambda_R}. \quad (1)$$

Both h_S and h_L will be close to 1, if sequentially accessed pages are prefetched, and if the combined locality sets of all concurrently executing transactions are small compared to the database buffer size. h_R is estimated as a function of buffer size based on the skew characterization described in the next section. The characterization can also be used to predict the effect of multi-node invalidation. Buffer invalidation will have impact only on h_R. If the transactions are routed randomly to all nodes then, using the analysis of Dan et al. (1994), the buffer hit probability due to random access can be estimated in a multi-node environment.

3. Recursive Binary Partitioning Algorithm

In this section, we describe an algorithm that infers the access skew, given the random access component of a database access trace. As mentioned in the previous

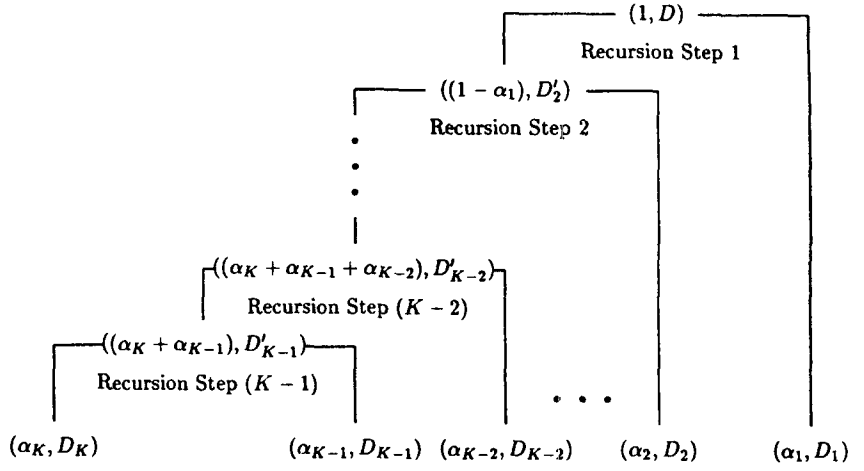
section, the access skew can be characterized by logically grouping the pages into a smaller number of disjoint partitions such that the access frequencies to pages within a partition can be treated as equal. Assume that a particular database needs to be divided into K partitions for a desired accuracy in the prediction of buffer hit probability. Let α_i be the frequency that an access will go to partition i, and let D_i be the number of pages in that partition. Then the access skew is characterized by $[(\alpha_i,D_i),\ i=1,...,K]$. Given an access trace, we have to to determine the value of K as well as $[(\alpha_i,D_i),\ i=1,...,K]$.

We first obtain the buffer hit probability vs buffer size curve under the LRU policy through trace driven simulation. Let B_j, $j=1,...,N$ be the set of buffer sizes for which buffer hit probabilities, h_j^{sim}, j=1,...,N, are evaluated. This can be done in a single pass of the trace (Mattson et al., 1970). The buffer sizes selected are somewhat arbitrary, but they should reflect the range of buffer sizes for which we are interested in predicting the buffer hit probability. Let h_j^{ana}, $j=1,...,N$, be the predicted buffer hit probabilities based on the skew characterization for the buffer sizes B_j, $j=1,...,N$ under the LRU policy. (See Appendix for a summary of the buffer hit analysis, which is based on Dan and Towsley, 1990.) We say that the skew is well characterized if the absolute difference ($|h_j^{sim} - h_j^{ana}|$) between the buffer hit probabilities for any buffer size under the given replacement policy obtained through a trace driven simulation and through an analytical prediction is within some desired accuracy (say, 1%). In this article, we use LRU replacement policy for the purpose of characterization. Similar approaches can be used for other replacement (e.g., clock) policies.

The above problem can be thought of as an optimization problem, where the optimal values of the parameters $[(\alpha_i, D_i),\ i=1,...,K]$ need to be determined such that some error function, $\mathcal{E}$ (h_j^{sim}, h_j^{ana}, $j=1,...,N$) is minimized. The number of partitions, K, is also an unknown parameter and, hence, needs to be determined by iterating over K (and carrying out optimization for each value of K) until some desired accuracy in the buffer hit probabilities is reached. For each search step in the optimization procedure, the computation of the predicted curve requires analytic solution of the LRU model for the buffer sizes $B_j, j=1,...,N$. Since this computational overhead may be significant, the efficiency of the optimization algorithm is of particular concern. The search space for the optimization algorithm is very large (e.g., range for partition size, D_i, is given by $1 \leq D_i \leq D_{max}$, where D_{max} is the maximum number of database pages), and starting anew for each iteration over K will make the algorithm inefficient.

We overcome these difficulties by taking advantage of one special property observed for the LRU replacement policy under the IRM access pattern considered. The observation is that in estimating the buffer hit probability for a given buffer size, the "relatively" hotter partitions (with total size much smaller than the buffer size) can be lumped together and treated as one partition with little effect on the accuracy of the estimate. This is due to the fact that the pages of the hotter partitions are mostly retained in the buffer for larger buffer sizes. This property

Figure 1. Recursive binary partitioning algorithm



is referred to as the property of *insensitivity* of the buffer hit probability for large buffer sizes to the differences in access frequency of the hotter partitions. We can thus devise a recursive algorithm that uses a two-partition model to match the simulated curve for the large buffer sizes first, and then increase the number of partitions to improve the accuracy for smaller buffer sizes. Each recursion step solves only a two-partition problem (details are described later in this section), by dividing the hotter partition (frequency and size) from the previous recursion step into two smaller partitions. The colder partitions derived in the earlier steps of the recursion need not be revised due to the insensitivity property (i.e., very few pages of the colder partitions are retained in the buffer for smaller buffer sizes).

3.1 Overview of the Algorithm

Figure 1 illustrates the algorithm through a schematic diagram. At the beginning, the undivided database consists of D pages and the relative access frequency to the whole database is assumed to be unity. At recursion step 1, the database is divided into two partitions such that the frequencies of the two new partitions add up to the frequency of the undivided database. α_1 and D_1 are the size and access frequency of the coldest partition. Here, the smaller partition represents the union of all other (hotter) partitions. The recursion step also determines the size of this equivalent partition, D'_2, that minimizes the error function at recursion step 1. At recursion step 2, the hotter partition is subdivided into two new partitions with frequency and size (α_2, D_2) and ($(1 - (\alpha_1 + \alpha_2)$, D'_3), respectively. The partitioning process is repeated and at each subsequent recursion step the error function is further minimized. In general, at recursion step l, the smallest partition (which is the partition to be split) represents the equivalent partition corresponding to the union of the $(K - l)$ hottest partitions and the frequencies and the corresponding sizes of

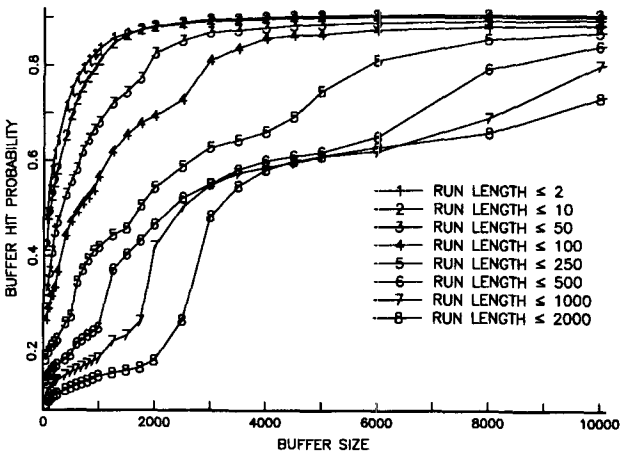
all colder partitions determined in earlier recursion steps, (α_i, D_i), $i < l$, are kept unchanged. The algorithm terminates either when the desired accuracy or some specified limit on the maximum number of partitions is reached. The choice of error function as well as the selection of the associated optimization algorithm are non-trivial problems and the issues are discussed below.

- *Choice of Error (Objective) Function:* We choose the area between the simulation and the prediction curves as the objective function to be minimized. Let s_1 and s_2 be the points on the simulated curve for buffer sizes b_1 and b_2, respectively. Let p_1 and p_2 be the corresponding points (for the same buffer sizes) on the predicted curve. Then the area enclosed by the 4 points between the two curves can be approximated as $(b_2 - b_1)((s_1 - p_1) + (s_2 - p_2))/2$. To put an ordering on the search space, we do not allow in any search step the predicted and simulated curves to intersect each other. Without any loss of generality, we assume that the predicted curve always lies below the simulation curve (i.e., $s_1 \geq p_1$ and $s_2 \geq p_2$. One could devise a similar scheme by assuming the predicted curve always lies above the simulated curve.) As the objective is to match the curves at the large buffer sizes first and then to improve the accuracy on smaller buffer sizes at subsequent recursive steps, we do not seek uniform discrepancies over all buffer sizes at all recursion steps. The better match at the higher end can be achieved by assigning higher weights (Section 3.2) to the areas at the higher end. Note also that the *least square error estimation* reduces the maximum error (since square of error translates to higher weight) rather than the errors we seek to minimize, and therefore is not appropriate for our objective.

- *Constraint on Search Space:* Under the IRM access pattern and the LRU replacement policy, the buffer hit probability vs buffer size curve is concave (i.e., with smaller marginal improvement of buffer hit probability for larger buffer size; Van den Berg, 1993). The buffer locations near the LRU stack top will have a higher probability of holding a hot page than the locations near the stack bottom. Therefore, the increase in buffer hit probability for an additional buffer allocation will be smaller for larger buffer sizes (i.e., will have diminishing return). For this type of buffer hit curve, we can use similarly shaped concave curves to match it from below. This restriction will greatly simplify the search space. Note that ordering in our matching process (higher-end of the buffer size first), as described earlier, further restricts the search space. A one step approach to match the curves would have a far more complex search space to go through.

- *Non-Concave Buffer Hit Curve:* As mentioned above, the buffer hit probability for an IRM access pattern is a concave function. However, in the access trace, the presence of sequential and looping sequential behavior will cause the buffer hit probability vs buffer size curve to violate this property. As

Figure 2. Effect of separation of sequential access on the buffer hit probability of remaining (R3)



discussed before, we use some simple rules to identify and filter out these non-random components. Still the process is not perfect and some of them may continue to be present in the trace and make the curve non-concave. Although in this study we rarely observe any non-IRM points, the following is included to make the methodology complete. Figure 5 shows a hypothetical buffer hit probability vs buffer size curve. As marked in the figure, point $p1$ violates the IRM property, as the slope of the curve on the right is larger than the slope of the curve on its left. The best we can hope for is to match the envelope of the buffer hit probability curve. In this case, our characterization algorithm ignores any points that violate the IRM property. We refer to the points on the curve that do not violate the IRM property, as IRM points, and the others as non-IRM points. In the above example, there are many such non-IRM points. Note that after the non-IRM point $p2$ is eliminated, point $p3$ also becomes non-IRM and should be eliminated. The elimination process continues, and at the end we are left only with the IRM points.[4]

- *Effective Characterization Range:* For some access skew a very accurate char-

4. Note that in Figure 5, there are no points on the left of point $p2$ that satisfy the IRM property. We therefore, have two choices: either to eliminate completely all the points to the left of point $p2$, or to include the leftmost point that will capture the upper envelope of the simulation curve as the curve to match. We have chosen the second alternative, since the first alternative results in characterization that gives larger error in buffer hit prediction for smaller buffer sizes.

Figure 3. Effect of prefetching on buffer hit of sequential and non-sequential accesses

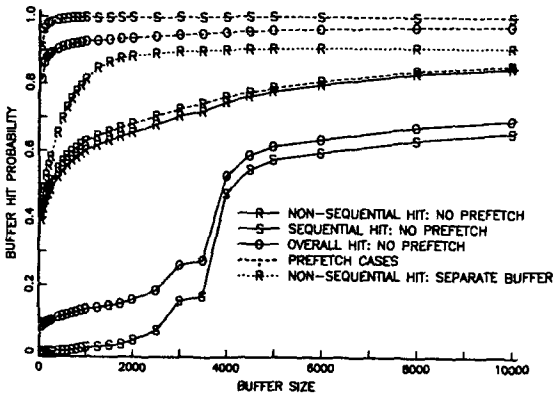
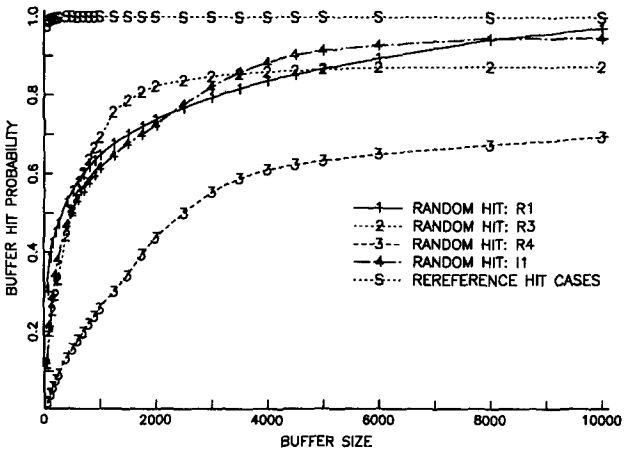


Figure 4. Random and reference buffer hit probabilities

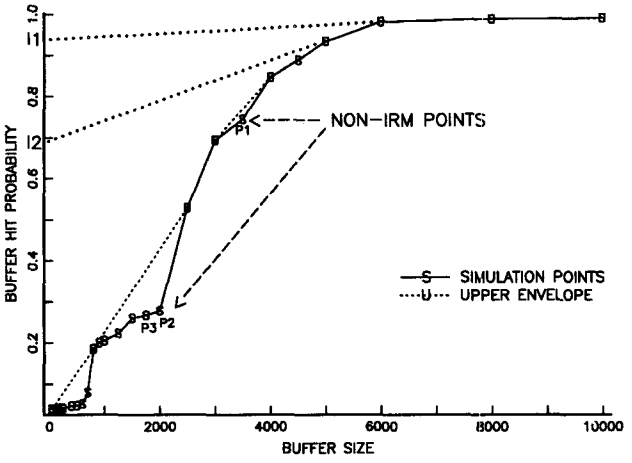


acterization may require a large number of partitions. Given a fixed number of partitions to choose, an effective characterization will choose only those points that result in minimum error in the prediction of buffer hit probability in some specified range. For example, if the minimum allocation of buffer space is large (say, 1,000 pages), small partitioning sizes that have significant effect only for small buffer sizes (say, < 500 pages) are not very important due to the insensitivity property.

3.2 Details of the Algorithm

We will now detail the algorithm. Assume that the buffer hit probabilities, h_j^{sim}, for the buffer sizes, B_j, $j=1,...,N$, are obtained from the trace driven simulation. In the case that there are points violating the IRM property, the algorithm removes these

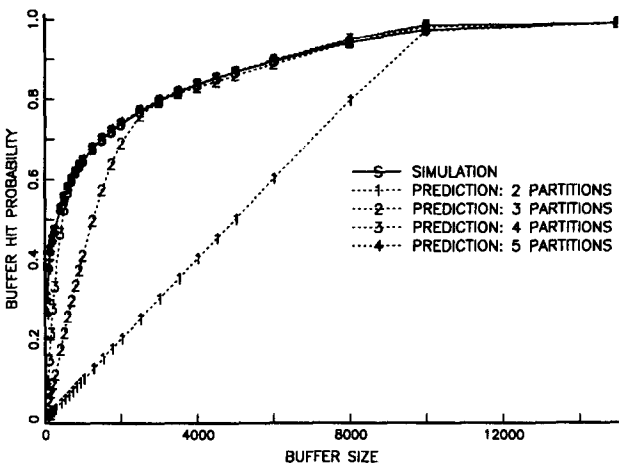
Figure 5. Illustration of recursive binary partitioning



points to construct an envelope of the simulated buffer hit curve for matching with the predicted curve. In the following, we still refer to this curve as the simulated buffer hit curve. The algorithm then tries to match the predicted curve to the simulated curve at the higher end through minimization of the weighted area. The weight assignment is as follows. Let M be the number of IRM points to be matched. The M points and the origin can be used to divide the buffer size axis and, hence, the area between the simulated and the predicted curves into M regions. We will refer to the region closest to the origin as region 1, and the region between the points $(j-1)$ and j as region j. As the algorithm tries to match the higher end first, higher weights should be given to the areas of the higher region. However, a very large weight for the higher region will make the algorithm insensitive for the lower buffer sizes. Since we merely want to express the preference of matching at the higher end, without making the algorithm insensitive for the lower end, a linear weight assignment (value of weight for each region is its index) seems to be a good compromise. We will show in the validation section that this weight assignment works well for a wide range of skewed access patterns.

Recall that at the l^{th} recursion step, the $(K-l)$ hottest partitions are grouped together into an equivalent partition of size D'_{l+1} and frequency $\sum_{i=l+1}^{K} \alpha_i$ (i.e., $1 - \sum_{i=1}^{l} \alpha_i$). The frequencies and their corresponding sizes, (α_i, D_i), $i < l$, of all colder partitions determined in the earlier recursion steps are kept unchanged. However, the smallest partition of the previous step (i.e., l^{th} partition) is divided into two new partitions, such that the frequencies of the two new partitions add up to the frequency of the undivided partition (Figure 1). The sizes of the two new partitions (D'_{l+1} and D_l) however, are allowed to vary independently (since, the size of the equivalent partition, D'_l, is always smaller than the total size of the new partitions, D'_{l+1} and D_l). Hence, at each recursion step l the optimization

Figure 6. Validation of random buffer hit probability (R1)



procedure determines only the three new parameters (D'_{l+1}, D_l and α_l) such that the weighted area is minimized further than the previous step. The algorithm is illustrated through an example. In Figure 6, the predicted buffer hit probability after each recursion step, as well as the simulated buffer hit probabilities, are shown for Relation 1. As can be seen from the figure, after the first step the predicted curve matches well with the simulated curve for larger buffer sizes ($>$ 10,000 pages). After the subsequent recursion steps, the predicted buffer hit probability matches for a longer and longer range of buffer sizes. Also, partitioning of the smallest partition does not affect the match at the higher end. Further validation will be shown in Section 4.1.

We will now describe a few ways to improve the efficiency of the optimization algorithm used at each recursion step.

1. *Efficient LRU Estimation:* The analytic estimation of the buffer hit probability under the IRM access pattern and LRU replacement policy for a buffer size of B and for a workload consisting of K partitions is given in the Appendix and is of order $O(KB)$ (Dan and Towsley, 1990). The computation overhead can be easily reduced for very large buffer sizes (say, $>>$1,000) by scaling down the database and the partition sizes (by the same scale down factor) such that the buffer size is of the order of 1,000 pages. We note from our experience that this introduces very little error.

2. *Starting Value for Parameters:* Good starting values of D'_{l+1}, D_l and α_l at each recursion step l are of extreme importance to reduce the number of search steps needed by the optimization procedure at each recursion step.

For recursion step 1, we note that the asymptote to the buffer hit probability curve will intersect the buffer hit probability axis at the point $(1 - \alpha_1)$ (i.e., the intersection point, I_1, represents the total frequency of accessing all but the coldest partition; see Figure 5).[5] Let *irm(M)* represent the M^{th} IRM point on the simulated curve. The asymptote is then approximated as the line passing through the highest two IRM points on the buffer hit curve (i.e., points *irm(M)* and *irm(M − 1)*). Therefore, (1 - I_1) is taken to be the starting value of α_1. At each subsequent recursion step l, the starting value of $\sum_{i=l+1}^{K} \alpha_i$ is taken to be the intersection point (I_l) on the buffer hit probability axis of the line passing through the highest two IRM points for which the predicted curve of the previous step differ from the simulated curve by an amount greater than the desired accuracy (Figure 5).

The starting values of D'_{l+1} and D_l are chosen as follows. Let B_s^l be the smallest buffer size for which the buffer hit probability $h_s^{sim} \geq I_l$. This implies that the total size of the smallest $(K - l)$ partitions, D'_{l+1}, is at most B_s^l.[6] Therefore, B_s^l is a good starting value for D'_{l+1}. The starting value for D_l is $(D'_l - D'_{l+1})$. Note that for $l = 1$, D'_1 is the size of the total active database (D), which is approximated as

$$D = B_{irm(M)} + \frac{(1 - h^{sim}_{irm(M)})(h^{sim}_{irm(M)} - h^{sim}_{irm(M-1)})}{(B_{irm(M)} - B_{irm(M-1)})}.$$

The above approximation linearly extrapolates the buffer hit probability curve to determine the buffer size for which the buffer hit probability becomes unity.

3. *Step Size in Parameter Search:* The search on any parameter can be made efficient by first taking progressively larger steps, and then once the predicted curve violates the constraint, taking progressively smaller steps (Bracketing and Bisection method; Press et al., 1986).

The resulting algorithm is very efficient. A summary of the algorithm is shown below, where $\mathcal{E}\ (h_j^{sim}, h_j^{ana}, j = 1, M)$ represents the weighted error estimate between the h_j^{sim} and h_j^{ana} over M selected points. We are unable to provide estimates of

---

5. If the coldest partition is much larger than the rest, the asymptote of the buffer hit probability curve would be $(1 - \alpha_1) + \alpha_1 B / D_1$, where B is the buffer size. That is to say for a large buffer size B, accessing the coldest partition has a buffer hit probability of B / D_1 while accessing the hotter partitions always results in a hit.

6. At buffer size B_s^l, not all pages in the buffer belong to the hottest $k - l$ partitions. If only pages from those partitions are buffered, buffer hit I_l would be reached at a smaller buffer size.

its computational overhead. However, for most of our examples presented here, it took less than a minute of CPU time (7.6 - 46.3 seconds) on the IBM mainframe. Note that once each relation is characterized prediction of buffer hit for various composition of workloads (due to buffer pool allocation and load change) and buffer sizes can be done very efficiently through analytical models. In contrast, trace driven simulation for each such composition will take on the order of minutes and hours.

l:=1; *remalpha* := 1.0; /* after l^{th} step *remalpha* = $\sum_{i=l+1}^{K} \alpha_i$ */
$D := B_{irm(M)} + (1 - h^{sim}_{irm(M)})\,(h^{sim}_{irm(M)} - h^{sim}_{irm(M-1)}) \,/\, (B_{irm(M)} - B_{irm(M-1)})$;
$D'_1 := D$; /* initial undivided partition */

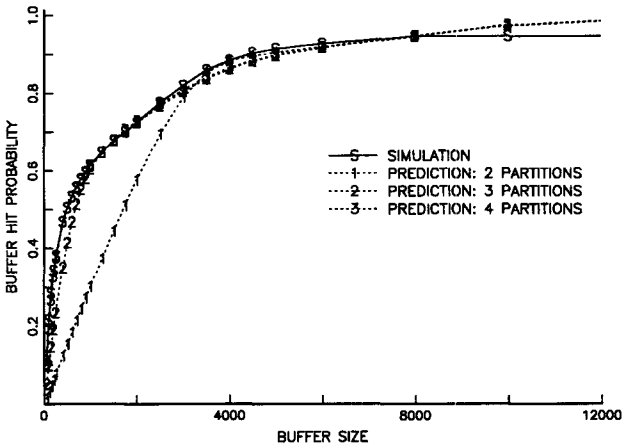
while$((l <= K) \& (\exists_i |h^{sim}_{irm(i)} - h^{ana}_{irm(i)}| > accuracy,\ i \in (1 \ldots M))$
{
    /* *starting values* */
    α_l := *remalpha* $- I_l$; /* I_l is the intersection point */
    $D'_{l+1} := B^l_s$;
    $D_l := D'_l - D'_{l+1}$;

    /* search for α_l, D_l, and D'_{l+1} */
    /* (α_i, D_i), $i < l$ are fixed at the l^{th} step */
Minimize $\mathcal{E}\left[h^{sim}_{irm(j)}, h^{ana}_{irm(j)}\left(D'_{l+1}, D_l, \alpha_l, (\alpha_i, D_i), i < l\right)\right.$,
    $\left. j=1,\ldots,M \mid (\alpha_i, D_i), i < l \right]$
    remalpha = *remalpha* $- \alpha_l$;
    $l := l + 1$;
}

3.3 Estimation of Update Probabilities

All database pages may not be updated with the same frequency. The knowledge of update probability for each of the logical data partitions is required for the estimation of the multi-node buffer invalidation effect (Section 4.3). As above, an analogous method can be devised to determine the update probabilities, γ_i, $i = 1,\ldots,K$, to each region. For this purpose, only the update (i.e., Setwrite) entries are used in the trace driven simulation to obtain the buffer hit probability vs buffer size curve only for the update operations. (Recall that in the previous subsection, we determine the access frequency and the size of each region from the buffer hit probability curve.) The partition sizes, D_i, $i = 1,\ldots,K$, are already known, and the corresponding access frequency of D_i is equal to $\gamma_i \alpha_i$, where α_i is also known. Hence, only γ_i, $i = 1,\ldots,K$, needs to be determined, such that the predicted curve using $(\gamma_i \alpha_i, D_i)$, $i = 1,\ldots,K$ is close to the simulated curve.

Figure 7. Validation of random buffer hit probability (I1)



4. Model Validation and Applications

In this section, we validate our algorithm extensively, based on the database workload described in Section 2. We then show how the characterization of individual files can be used to predict the buffer hit probability of the composite workload (i.e., more than one file sharing the same buffer space), as well as the effect of multi-node invalidation, with good accuracy. We will present results primarily based on database trace 1 except in Section 4.3 where database trace 2 is used.

4.1 Validation

The random buffer hit probabilities of Relation 1 (R1) and Index file 1 (I1), obtained through prediction and trace driven simulation, are plotted in Figures 6 and 7, respectively. The simulated curve presented in this section includes both IRM points and non-IRM points, if any. Multiple predicted curves refer to the predicted values after various recursion steps. (A small number of recursion steps is generally sufficient (e.g., four and three steps for R1 and I1, respectively.) The predicted curve after the last recursion step matches well with the trace driven simulation results for all buffer sizes, and the curves from the intermediate steps match only a portion of the simulated curve. Similar matches in buffer hit probability were found for many other relations and index files that we experimented with. The sizes and frequencies of all partitions (hottest to coldest) after all recursion steps for R1 are shown in Table 2. From now on, only the final predicted curves for various files are presented.

4.2 Prediction of Buffer Hit Probability of Composite Workloads

Next we consider how to predict the buffer hit probability of a composite workload

Table 2. Access frequencies and sizes of all partitions of Relation 1 after various recursion steps

Step #	*(α_i in per cent, D_i in number of pages)*				
1				(98.8, 9690)	(1.2, 36438)
2			(75.0, 1276)	(23.8, 7632)	(1.2, 36438)
3		(55.6, 204)	(19.4, 754)	(23.8, 7632)	(1.2, 36438)
4	(36.3, 20)	(19.3, 127)	(19.4, 754)	(23.8, 7632)	(1.2, 36438)

from those of its component workloads. Based on the skew characterization of each of the files and the access fraction to each of the files, the overall random buffer hit probability under the LRU replacement policy can be computed using the analytical buffer model (Dan and Towsley, 1990; see Appendix for a summary). The computation uses the total number of logical partitions as the sum of the partitions of each file. The corresponding access frequency is given by the product of the original access frequency to each partition (normalized for each file) and the fraction of the accesses to that file. The rereference component of each relation is assumed to be retained in the buffer. The overall non-sequential buffer hit probability is the weighted sum of the components. Figure 8 shows the simulated and predicted curves for random, rereference, and overall non-sequential buffer hit probabilities for the composite workload of R1 and R4.[7] As can be seen in the figure, all three buffer hit components are well predicted. The composite workload is created by filtering out the accesses to all other files except to R1 and R4 from the original database access trace. The fraction of the accesses of the composite workload that goes to each of the files is computed from the trace.

The validation process is repeated for various compositions of workloads (multiple files, and various types of files). Figure 9 shows the overall non-sequential buffer hit probabilities of the combinations (R1 & R4) and (R1 & I1). Also shown are those of R1, R4, and I1, respectively. The predicted values of the composite workloads (dotted curves) show excellent agreement with the simulation values. Note that the shapes of the composite buffer hit probabilities are very different for the two cases.

The prediction can be put to use to answer *what if* types of questions. As the fraction of the load to each of the files changes (due to change in application mix), the LRU analysis can also be used to predict the buffer hit probabilities for the new workload compositions, assuming the access skew characterization for each file remains unchanged (Kearns and Defazio, 1989). Figure 10 shows the changes in

7. R4 is chosen over R2 and R3 for the purpose of presentation as there are more random accesses to this relation compared to the others (see Table 1).

Figure 8. Validation of composite buffer hit probability (R1 & R4)

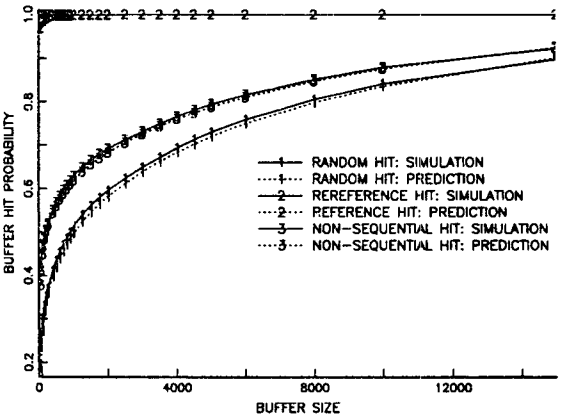
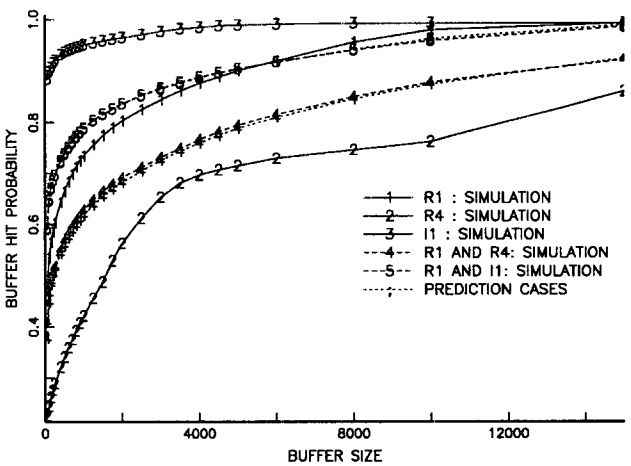


Figure 9. Validation of composite (non-sequential) buffet hit probabilities (R1 & R4 and R1 & I1)



buffer hit probability of the composite workload as the transaction load to R1 and R4 changes. Let $L1$ and $L2$ be the access rates to R1 and R4, respectively. The solid curve represents the original load mix and the dotted curves represent the predicted buffer hit probabilities after various load changes. Four different cases in terms of the ratios of $L1$ to $L2$ are presented. As the relative load to R1 decreases, the overall buffer hit probability decreases. The effect is significantly different for the combination of R1 and I1. In Figure 11, besides the original case, four additional cases in terms of the ratios on the load changes of R1 and I1 are presented. Note also that the composite buffer hit probability of the workload (R1 and I1) changes only for small buffer sizes in contrast to the case in Figure 10.

The changes in relative loads to the component files can have a significant effect on the buffer hit probability, if the access skews of the component files

are very different. This would affect the total buffer requirement or the optimal buffer allocations among the buffer pools in a multiple buffer pool environment (Effelsberg and Haerder, 1984). Dan and Towsley (1990) studied the performance gain of optimal file assignment and allocation of buffer pools, based on a given skewed access workload. Generally speaking, it would be too time consuming to try out various workload compositions or buffer pool assignments of files through trace driven simulations. To predict the effect of the load changes would be even more formidable. The proposed methodology provides an analytic approach to predict the buffer hit probability efficiently.

4.3 Prediction of Multiple-Node Invalidation Effect

Multi-node systems can provide the horizontal growth capability (Stickland et al., 1982; Kronenberg et al., 1986). One way to couple multiple systems is through data sharing (Dan et al., 1994*a*, 1994*b*) where the database resides on a set of shared disks and each processing node retains in its local (memory) buffer the recently accessed pages. Therefore, the same page may be retained in the buffers of multiple nodes. When a page is updated, the other copies of the page in remote nodes need to be invalidated (Dan et al., 1994*a*). Therefore, a buffer coherency protocol is required to maintain the consistency across nodes. The buffer invalidation reduces the local buffer hit probability. Prediction of the multi-node buffer invalidation effect is extremely useful from the point of view of system configuration and capacity planning when the number of nodes changes. Of course, the invalidation effect depends on the transaction routing (i.e., how the files are shared by the transactions executing in multiple nodes). We will assume random transaction routing and, therefore, all files are equally likely to be accessed by all nodes.

The invalidation effect depends not only on access skew characterization but also on the transaction update probability. The buffer hit probability is predicted using the analytical model (Dan et al., 1994*a*). Update probabilities may be different for hot and cold partitions of a relation. To obtain update probability for each partition, we first construct a trace consisting of only the Setwrite (update) operations. Since each Setwrite entry is preceded by one or more Getpage entries for the same page, a relationship exists between the number of Setwrite operations for each partition and its access frequency and update probability. Therefore, if we treat this new trace as any other access trace, a higher buffer hit probability in this trace implies a high update probability for the hottest partitions. We use the matching procedure outlined in Section 3.3 to obtain these update probabilities. Figure 12 shows the validation of Setwrite (update) buffer hit probability for Relation 2 of the second database trace, which is from a banking application with heavy updates. The access skew is characterized by three partitions and, therefore, two recursion steps are required to obtain the update probabilities. Step 1 shows the intermediate result where only the update probability of the largest partition is correctly determined. A large number of pages read from this relation are also updated. This will give a

Figure 10. Effect of load change on composite (non-sequential) buffer hit probability of R1 & R4

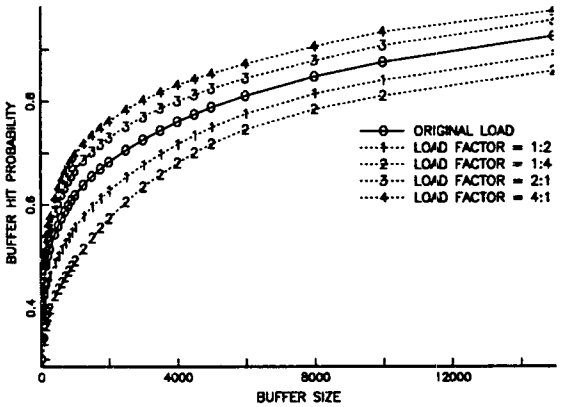
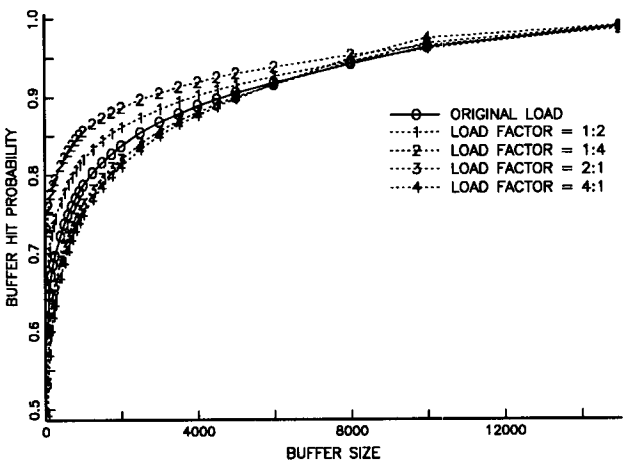


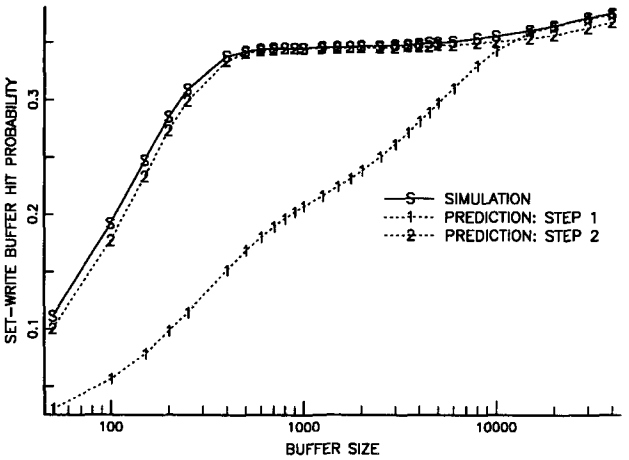
Figure 11. Effect of load change on composite (non-sequential) buffer hit probability of R1 & I1



very noticeable invalidation effect (i.e., a stress case to validate the methodology). The Setwrite buffer hit probability matches only if the update probabilities for each of the logical partitions are well predicted. The curves are plotted in logarithmic scale since the sizes of the hot partitions are small.

We now show by a simple example that to estimate the multi-node buffer invalidation effect, only the random access component needs to be considered, while the rereference component should be excluded. Consider the following example with a two-node system. Assume that each transaction only accesses one page of data. It rereferences the page nine times while making an update during the last rereference. We further assume that the first reference, which is a random access, has a buffer hit probability of 0.1. The overall buffer hit probability would be 0.91 since the other nine out of the ten references are hits. In a two-node system, if the

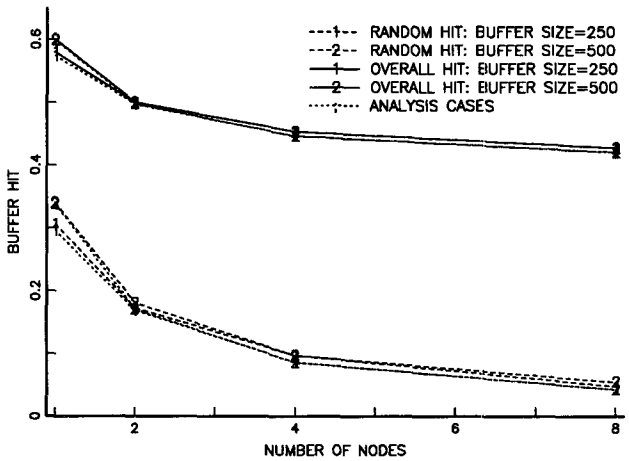
Figure 12. Validation of Setwrite buffer hit prediction



transactions are randomly routed between the two nodes, we expect the buffer hit probability of the first (random) reference to be the same at either node. A page update in a node causes an invalidation in the other node if a copy of that page resides in the buffer of the remote node. Therefore, the probability that the final update would cause a buffer invalidation at the other node would be the same as the buffer hit probability (0.1) of the initial random access, not the overall buffer hit probability (0.91).

Figure 13 shows the effect of cross-node buffer invalidation on the buffer hit probability for the above relation. This information would be useful in predicting the effect of migrating from a single node to multi-node environment when database load increases. There is a sharp drop in buffer hit probability due to the invalidation effect for the multi-node case. However, this is accurately captured by the proposed methodology as the match between simulation and predicted curves is excellent both for random and overall non-sequential buffer hit probabilities (for the buffer sizes of 250 and 500 pages/node). Note that the buffer hit probability curves for 250 and 500 pages are close to each other beyond a single node. This is due to the multi-node invalidation effect, since the effective number of hot pages that can be retained in the buffer becomes smaller than 250 pages (Dan et al., 1994). Also note that the invalidation effect has a stronger impact on the random buffer hit probability than on the overall non-sequential buffer hit probability. This is due to the fact that rereferenced pages are always found in the buffer since invalidation can only cause a buffer miss for the first reference to the page. If the two components (random and rereference) were not separately accounted, the invalidation effect will be over predicted as explained in Section 2.

Figure 13. Validation of buffer hit prediction for multi-node environment



Conclusions

In this article, we showed that, from a database reference trace, the database accesses can be effectively categorized into three types of access patterns: (1) random accesses by transactions, (2) locality within a transaction due to rereferencing, and (3) sequential accesses by queries. For buffer managers with prefetch capability for sequential accesses, the synchronous database I/Os mainly come from random accesses. Thus, the main issue is to characterize the random access pattern and predict its buffer hit probabilities under different buffer sizes. We proposed a skew characterization method that logically groups the pages into a small number of partitions such that the frequency of accessing a page within a partition can be treated as equal. We also developed an algorithm that can be used to infer the access skew from the buffer hit probabilities for a subset of the buffer sizes. This avoids explicit estimation of individual access frequencies for a large number of database pages. The skew characterization algorithm is recursive. At each recursion step, it divides the smallest partition into two new partitions and determines the sizes and access frequencies to the two new partitions. The algorithm terminates when the maximum number of partitions is reached or the difference between the predicted and the simulated buffer hit probabilities is within some desired accuracy. We provided extensive validation of our algorithm using production database traces. The characterization algorithm provides excellent buffer hit estimates for both the random access and the overall buffer hit probabilities.

The knowledge of database access skew is useful for both workload management (buffer pool allocation, transaction routing, etc.), as well as capacity planning for changing transaction mix and rate. We showed how the characterization of the individual database files can be used to estimate the buffer hit probability of a

composite workload where data from more than one file are buffered in the same buffer pool. Our estimate matched very well with the buffer hit probability obtained through a trace driven simulation. This approach can be used to predict the buffer hit probability under changing workload mix and rate. Also considered is the issue of predicting the buffer hit probability when migrating from a single node system to a configuration with multiple nodes. In a multiple node environment, cross-node buffer invalidation effect reduces the local buffer hit probability. The invalidation effect is sensitive to the skewness of the access pattern. We showed that the skew characterization can accurately predict this effect.

Acknowledgement

We would like to thank Joel Wolf and Steve Lavenberg for their comments and suggestions, and Ted Messinger for providing us with the trace.

References

Chou, H.T. and Dewitt, D.J. An evaluation of buffer management strategies for relational database systems. *Eleventh International Conference on Very Large Databases*, Stockholm, Sweden, 1985.

Cornell, D.W. and Yu, P.S. Integration of buffer management and query optimization in relational database environment. *Fifteenth International Conference on Very Large Databases*, Amsterdam, Netherlands, 1989.

Dan, A. and Towsley, D. An approximate analysis of the LRU and FIFO buffer replacement schemes. *ACM SIGMETRICS,* Denver, CO, 1990.

Dan, A., Dias, D.M., and Yu, P.S. Buffer analysis for a data sharing environment with skewed data access. *IEEE Transactions on Knowledge and Data Engineering,* 6(2):331-337, 1994*a*.

Dan, A., Yu, P.S., and Dias, D.M. Performance modelling and comparisons of global shared buffer management policies in a cluster environment. *IEEE Transactions on Computers,* 43(11):1281-1297, 1994*b*.

Dan, A., Yu, P.S., and Chung, J.Y. Characterization of database access skew in a transaction processing environment. *IBM Research Report RC 17436*, 1991.

Date, C.J. and White, C.J. *A Guide to DB2,* Third edition, Reading, MA: Addision-Wesley, 1989.

Effelsberg, W. and Loomis, M.E.S. Logical, internal, and physical reference behavior in CODASYL database systems. *ACM Transactions on Database Systems,* 9(2):187-213, 1984.

Effelsberg, W. and Haerder, T. Principles of database buffer management. *ACM Transactions on Database Systems,* 9(4):560-595, 1984.

Faloutsos, C., Ng, R., and Sellis, T. Predictive load control for flexible buffer allocation. *Seventeenth International Conference on Very Large Databases*, Barcelona, Spain, 1991.

Gray, J., ed. *The Benchmark Handbook for Database and Transaction Processing Systems.* San Mateo, CA: Morgan Kaufmann, 1991.

Hawthorn, P. and Stonebraker, M. Performance analysis of a relational data base management system. *ACM SIGMOD*, Boston, MA, 1979.

IBM Database 2 Administration Guide, Vol. III, Section 7, *Performance Monitoring and Tuning,* SC26-4888-00, 1993.

Kearns, J.P. and Defazio, S. Diversity in database reference behavior. *Performance Evaluation Review,* 17(1):11-19, 1989.

Kronenberg, N., Levy, H., and Strecker, W.D. VAXcluster: A closely-coupled distributed system. *ACM Transactions on Computer Systems*, 4:130-146, 1986.

Mattson, R.L., Gecsei, J., Slutz, D.R., and Traiger, I.L. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78-117, 1970.

Ng, R., Faloutsos, C., and Sellis, T. Flexible buffer allocation based on marginal gains. *ACM SIGMOD*, Atlantic City, NJ, 1990.

Nicola, V.F., Dan, A., and Dias, D.M. Analysis of the generalized clock buffer replacement scheme for database transaction processing. *ACM SIGMETRICS*, Newport, RI, 1992.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. *Numerical Recipes.* New York, NY: Cambridge University Press, 1986.

Rodriguez-Rosell, J. Empirical data reference behavior in data base systems. *Computer*, 9(11):3-13, 1976.

Sacco, G.M. and Schkolnick, M. Buffer management in relational database systems. *ACM Transactions on Database Systems,* 11(4):473-498, 1986.

Smith, A.J. Sequentiality and prefetching in database systems. *ACM Transactions on Database Systems,* 3(3):223-247, 1978.

Strickland, J.P., Uhrowczik, P.P., and Watts, V.L. IMS/VS: An evolving system. *IBM Systems Journal*, 21:490–510, 1982.

Tay, Y.C., Suri, R., and Goodman, N. A mean value performance model for locking in databases: The no-waiting case. *Journal of the ACM,* 32(3):618-651, 1985.

Teng, J.Z., and Gumaer, R.A. Managing IBM Database 2 Buffers to Maximize Performance. *IBM Systems Journal*, 23(2):211-218, 1984.

Van den Berg, J. and Towsley, D. Properties of the miss ratio for a 2-level storage model with LRU or FIFO replacement strategy and Independent References. *IEEE Transactions on Computers,* 42(4):508-512, 1993.

Verkamo, A.I. Empirical results on locality in database referencing. *ACM SIGMETRICS*, Austin, TX, 1985.

Yu, P.S., Dias, D.M., and Lavenberg, S.S. On the analytical modeling of database concurrency control. *Journal of the ACM,* 40(4):831-872, 1993.

Yu, P.S. and Cornell, D.W. Optimal buffer allocation in a multi-query environment. *Seventh International Conference on Data Engineering*, Kobe, Japan, 1991.

Appendix

In this appendix, we outline the analysis of the LRU replacement policy (Dan and Towsley, 1990), which is used to derive the buffer hit probability for a database consisting of multiple partitions. Let the database consists of K partitions, and the access frequency and size of the i^{th} partition be α_i and D_i, respectively. Let B be the size of the buffer. To estimate the steady state buffer hit probability, we first estimate the average number of pages of each partition in the buffer. Let $X_i(j)$ denote the average number of pages of partition i in the top j locations of the LRU stack. Therefore, the buffer hit probability of the i^{th} partition is estimated as $H_i = X_i(B)/D_i$, and the overall buffer hit probability for a page as $H = \sum_{i=1}^{K} \alpha_i X_i(B)/D_i$. Let $p_i(j)$ be the probability that the j^{th} buffer location from the top of the LRU stack contains a page of partition i. Then,

$$X_i(j) = \sum_{l=1}^{j} p_i(l). \tag{2}$$

A recursive formulation is used to determine $p_i(j+1)$ given $p_i(j)$ for $j \geq 1$. Consider a smaller buffer consisting of the top j locations only. The buffer location $(j+1)$ receives the page that is pushed down from location j. Let $r_i(j)$ be the rate at which pages of partition i are pushed down from location j. The estimation of $p_i(j)$ is given as follows.

$$r_i(j) = \alpha_i(1 - \frac{X_i(j)}{D_i}). \tag{3}$$

(This is referred to as the *conservation of flow* as the rate of type i pages brought into the top j buffer locations must be equal to that taken out of them.)

$$p_i(j+1) \approx \frac{r_i(j)}{\sum_{i=1}^{K} r_i(j)}, \quad j = 1 \ldots B-1. \tag{4}$$

Equations 2, 3, and 4 can be solved iteratively, with the base condition of $p_i(1) = \alpha_i$. At the point when $X_i(j)$ is very close to its limit (D_i), $X_i(j)$ may exceed D_i because of the approximation in the above equations. This is corrected by resetting $X_i(j)$ to D_i whenever $X_i(j)$ exceeds D_i and $r_i(j)$ is taken to be zero for all subsequent steps for that partition.

The buffer hit probability of a composite workload can also be predicted using the above analysis. Assume the workload is composed of M files. Each file may have a different skewed access pattern. Let K_m be the number of partitions in the m^{th} file. Also, let $\alpha_{i,m}$ and $D_{i,m}$ be the frequency and sizes of the i^{th} partition of the m^{th} file. For the LRU analysis, the total number of partitions is the sum of all the partitions of the component files, that is, $K = \sum_{m=1}^{M} K_m$. Let λ_m be

the access rate (load) to the m^{th} file. In the composite workload, a new index, $I(i,m)$, can be defined to reference the i^{th} partition of the the m^{th} file. Then the normalized frequency to each of the K partitions are given by

$$\alpha_{I(i,m)} = \frac{\alpha_{i,m}\lambda_m}{\sum_{m=1}^{M}\lambda_m}. \tag{5}$$

The above LRU analysis methodology can again be applied to estimate the buffer hit probability of the composite workload.