Integration of Buffer Management and Query Optimization In Relational Database Environment

Douglas W. Cornell Philip S. Yu

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In today's relational database systems, query optimization and buffer management are generally treated independently. However, the query access plan contains information on data access patterns which can be useful hints to the buffer manager. Furthermore, the optimal access plans under different buffer sizes can be quite different. In this paper, an integrated approach to buffer management and query optimization is proposed and analyzed. The query strategy of all transaction types is simultaneously considered together with the buffer allocation strategy so as to optimize overall system performance. As the buffer allocation depends upon the buffer holding time or transaction response time which is determined by the buffer allocation and query optimization strategies, an optimization method combining a integer programming model with a queueing model applied iteratively is developed to capture this effect. To reduce the size of the optimization problem, a methodology based on the concept of buffer consumption is proposed to pre-analyze the query and substantially cut down the access plans to be considered. A detailed simulation is used to demonstrate the superiority of the integrated strategy as compared to other buffer management strategies based on working set and hot set.

1. Introduction

Database systems have generally relied on memory buffers to reduce disk accesses. Even with the trend of ever increasing memory size, the memory buffer usually can not accommodate all the databases in the system, and some buffer management strategy is needed to make the best use of the buffer space. The traditional approach to memory management in a virtual memory environment uses a working set model where the popular replacement policy is LRU, which replaces the least recently used page by a new page [Denn68].

For a network or hierarchical database system, reference strings tend to be unpredictable except for batch processing. A study on network databases can be found in [Effe84]. These types of systems seem to fit reasonably well with the working set model. However, queries to relational databases [Codd70] imply a lot of information on data references. The discussion in this paper is based on a database system similar to system R [Astr76]. The query optimizer analyzes each query and generates an access plan which contains detailed information on how each relation is accessed. (Information related to access plans can be found in [Seli79].) These access plans assume that none of the referenced pages will be found already in the buffer. The buffer manager does not take hints on reference patterns from the query optimizer except prefetch for sequential scan. Although some variant of the LRU policy is often used for buffer management, it is not considered to be well suited for the reference patterns of relational databases [Ston81]. Previous research directed toward utilizing information available about reference patterns has been described in [Sacca82], [Chou85], and [Sacca86]. In [Sacca82], [Sacca86] a hot set model is proposed. The basic idea there is to determine a hot set for every query and allocate sufficient buffer space to cover the hot set before executing a query. Certainly, this is a local optimization for each query to provide it with sufficient buffer space to minimize disk IO accesses. The potential buffer contention among queries is not addressed in the buffer allocation strategy. As pointed out in [Saaca86], straightforward implementation of this idea can lead to problems such as infinite waits, long queries blocking short queries, etc.. Some ad hoc techniques to relieve these problems are also suggested. In [Chou85], active instances (due to references from different queries) of a file are given different buffer pools and are managed by different replacement disciplines. A DBMIN algorithm is proposed for estimating the size and discipline for each file instance of a query is described. To compare performance with the hot set strategy, a simulation is developed in which a query is described by the CPU usage, number of IO's, and hot set size. All these works investigate the "right" buffer allocation for a given query plan or access path selection without considering the effect of other queries.

In a transaction processing environment, the characteristics of all transactions/queries are known a priori. Thus, if the transaction rate for each type is also known, a buffer management strategy may be devised to optimize buffer allocation for all queries. Furthermore, the query optimizer may be integrated with buffer management to take into consideration the buffer availability in selecting an appropriate access plan for each query to optimize overall performance. In this paper, we examine the integration of buffer management and query optimization. A detailed simulation is developed to compare the integrated approach with buffer management schemes based on the LRU working set model and the hot set model, respectively. We discuss the issues of buffer management and query optimization that motivate the integrated approach in Section 2. The mathematical formulation of the optimization problem is given in Section 3. Section 4 shows the performance improvement through the integrated approach using a hypothetical example. To reduce the size of the optimization problem, Section 5 develops a methodology based on the concept of buffer consumption to substantially trim down the number of access plans to be considered in the

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

optimization procedure. Buffer consumption is defined to be the product of query response time and buffer allocation. An access plan with a small amount of buffer allocation is not desirable unless the buffer consumption is also small. Concluding remarks appear in Section 6.

2. Buffer Management and Query Optimization

The conventional approach to query optimization is to examine each query in isolation and select the execution plan with the minimal cost based on some predefined cost function of IO and CPU requirements to execute the query [Seli79]. The impact of buffer management generally is not reflected in the cost function. The number of IO's estimated in the cost analysis does not reflect the potential effect of other transactions concurrently under execution. On the other hand, it is also quite difficult to assess this effect without detailed knowledge on how other concurrent transactions are progressing.

For each query type, different execution plans show different sensitivities to the size of memory allocated for data and thus have different memory requirements. In this paper, we use the expression data buffer to refer to memory allocated for storing data from the database. The best access plan for a given buffer size is not necessarily the best plan for another buffer size. Furthermore, it is not even clear whether every query type needs to be given the same constraint on buffer allocation. Different query types may have different diminishing points of return on buffer size and face different penalties on suboptimal buffer size. An integrated buffer management and query optimization strategy is thus needed so that the access plans for all query types are considered together based on buffer availability.

Let's look at an example. Consider a query joining three relations, R_1, R_2, and R_3. The relations consist of 10, 20, and 40 pages with cardinalities (C_i, $i = 1, 2, 3$) of 50, 100, and 500, respectively. The join selectivities used to calculate the number of matching tuples from the join operations are taken to be .01 for all joins. Assume that no index is available. To simplify the discussion, nested-loop join is assumed to be the method of choice. The query optimizer still needs to decide the join order. Ignoring buffer availability, a conventional query optimizer would choose R_1, R_2, R_3 as the join order with R_3 the inner-most relation. The number of page fetches would be 3000 ($= 20 \times C_1 + 40 \times 0.01 \times C_1 \times C_2$) if none of the relations are kept in the buffer. The number of IO would be minimized if relations R_2 and R_3 can be kept in the buffer. A total of 61 ($=20+40+1$) buffer pages would be required. This strategy would be optimum if that amount of buffer space can be made available. If the join order is taken to be R_3, R_2, R_1 with R_1 the inner-most relation, the number of page fetches would be 15000 ($= 20 \times C_3 + 10 \times 0.01 \times C_3 \times C_2$) if none of the relations are kept in the buffer. However, with 31 ($=10+20+1$) buffer pages, the number of IO can be minimized to read in each relation exactly once. Of course, this access plan does need to perform more tuple comparisons. This illustrates that the optimum access plan depends upon the buffer allocation and that there are trade-offs between different hardware resource (CPU and IO) requirements. So far we have been looking at one query alone. In the presence of multiple queries, buffer space needs to be allocated among the different queries so as to optimize the overall performance. One thus needs to consider all query types together.

3. Methodology

The methodology to obtain a global optimization on buffer management and query optimization is now described in this section. We focus on join queries here, as select type queries have little buffer requirement. An integer programming formulation is used to select the best access plan and associated buffer allocation. The objective function is similar to the cost function used in a conventional query optimizer, but is the sum over all query types weighted by query frequencies. The constraints are that not only the buffer allocation for each query, but also the time-averaged buffer requirement over all queries must be less than the total buffer size. Note that the time-averaged buffer requirement for each query type is proportional to its response time and arrival frequencies in addition to the buffer allocation for each instance of its execution. The response time certainly depends upon access plan selection and buffer allocation.

To get around this inter-dependency problem, we can decompose the problem into two parts and take an iterative approach. The first part is the optimization just described which uses an assumed response time for each query type, and the second part is a queueing model to solve for the response time based on the access plan selections and buffer allocation from the first part (the optimization problem). The optimization problem then uses the response time from the queueing model to solve for an improved solution. Alternatively, we can use the average response time over all queries (derived in Section 3.2) as the objective function and employ a heuristic technique like simulated annealing [Kirk83] to solve the non-linear programming problem. Simulated annealing can be helpful to address very large size problems or optimize response times directly [Wolf88].

3.1 Optimization Problem Under the Integrated Strategy

Consider a set of join queries Q_i, $i = 1, ..., N_Q$, and relations R_k, $k = 1, ..., N_R$. For each query type Q_i, there is a set of different join plans, QS_{ij}, $j = 1, ..., n_i$. Under the integrated strategy, each join plan specifically indicates whether a joining relation is to be kept in the buffer. For example, in a two-way join of R_k and R_j, nested-loop join with R_k (or R_j) in the buffer as the inner relation and nested-loop join with R_k (or R_j) as the inner relation but not in the buffer are four potential join plans. Although there may be a large number of potential access plans for each query, we can eliminate most of them from consideration of the optimization procedure discussed below through some simple pre-analysis of the access plans. This is addressed in Section 5. Let X_{ij} be 1 if plan QS_{ij} is adopted for Q_i and 0 otherwise. That is to say, each join plan is identified by a (0,1) integer programming variable to indicate whether it is adopted or not. Let D_{ij} be the number of pages to be read from disk if plan QS_{ij} is adopted. D_{ij} can be calculated based on the join algorithms and buffer allocation strategies. Define λ_i to be the arrival frequency for Q_i.

Now we formulate the optimization problem. The objective function can be any cost function on IO and CPU over all query types. Here we use the aggregate IO rate as the cost function assuming IO is the performance bottleneck. Thus,

$$\textit{objective function} = \sum_i \sum_j D_{ij} X_{ij} \lambda_i.$$

Constraints are included to guarantee that exactly one strategy is adopted for each query,

$$\sum_j X_{ij} = 1, j = 1,..., n_i.$$

Additional constraints are added to prevent the buffer from being overcommitted. First of all, no query can get a buffer allocation more than the total buffer size. Let B be the number of buffer pages available and F_{ij} be the number of buffer pages allocated for query Q_i under plan QS_{ij}. Thus F_{ij} is an input parameter derived from the access plan.

$$\sum_j X_{ij} F_{ij} \le B, \qquad \text{for each } i,$$

Furthermore, the average buffer usage of each query type is estimated as the product of buffer allocation and response time times the arrival frequency. In order to accommodate the fluctuations in query workload, the average buffer usage needs to be less than some fraction, α, of the total buffer. The appropriate value for α is explored in the next section.

$$\sum_i \lambda_i \sum_j X_{ij} F_{ij} RT_{ij} \le \alpha B$$

where RT_{ij} is the average response time of query Q_i under plan QS_{ij}. A starting value needs to be selected for each RT_{ij}, for $i = 1,..., N_Q$ and all plans $j = 1,..., n_i$ for query Q_i. The join plans selected by the solution of the resulting integer program are not yet optimum, since the initial response times are not chosen to be optimum. In Section 3.2, a queueing model is presented to estimate a new set of response times based upon the "optimal" strategy $\{QS_{i\beta_i}, i = 1,..., N_Q\}$ chosen by the optimization procedure. We iteratively solve the execution plan selection and buffer allocation from the integer programming problem and response time calculation from the queueing model until no further change in plan is observed or until the calculated average response time remains the same on successive iterations. If we choose a large response time for each query type to start the optimization problem, a very conservative buffer allocation will be made, as exaggerated response time leads to overestimates on the time average buffer requirement. The queueing analysis then provides an improved response time based on the strategy recommended from the optimization problem. With the improved response time or shortened buffer hold time, more buffer space can be allocated and lead to further improvement in the response time. It appears to converge after a few iterations.

3.2 The Queueing Model

So far we have been concentrated on buffer allocation. The concurrently executing queries compete not only for buffer but also CPU and IO. The contention on CPU and IO would increase the response time, thus the buffer hold time. The response time of the transaction processing system is determined using an open queueing network model. The system consists of a single CPU with speed *MIPS* and multiple (N_D) disks. Each database is assumed to be partitioned uniformly across the N_D disks based on its primary key. Under a given strategy, $\{QS_{i\beta_i}, i = 1,..., N_Q\}$, the overall system performance can be analyzed by assuming there is no reuse of the buffer between queries. The response time of each query can be calculated as its resource requirement on the CPU and disks can be predicted given the buffer allocation strategy. Define $U_{i\beta_i}$ to be the query-processing pathlength at the CPU for Q_i under $QS_{i\beta_i}$. Let I_1 be the CPU overhead for scanning/comparing a tuple, I_2 the overhead for returning a tuple to the application program, I_3 the overhead for an IO operation and I_{A_i} the application processing pathlength for type i query. $U_{i\beta_i}$ can be expressed in terms of these parameters as given in Appendix A. The total CPU processing cost is then

$$U_{CPU} = \sum_i \lambda_i U_{i\beta_i}$$

Let T_{IO} be the disk service time to perform an IO operation. The utilization of each disk assuming the load is spread uniformly across all disks for each relation can be shown to be

$$\rho_{IO} = \frac{T_{IO}}{N_D} \sum_i \lambda_i D_{i\beta_i}$$

The average response time of Q_i is

$$RT_{i\beta_i} = \frac{U_{i\beta_i}}{MIPS - U_{CPU}} + \frac{D_{i\beta_i} T_{IO}}{1 - \rho_{IO}},$$

where the first component is the sum of the service time and waiting time at the CPU and the second component is the sum of those times at the disks. The average response time over all queries, under strategy, $\{QS_{i\beta_i}, i = 1,..., N_Q\}$, would then be $RT = \sum_i \lambda_i RT_{i\beta_i} / \sum_j \lambda_j$.

The following model parameters are used in the performance comparisons. The pathlength parameters are set at $I_1 = 70$, $I_2 = 5K$, $I_3 = 5K$ and $I_{A_i} = 20K$, for all i. The system consists of a 14 MIPS processor and 4 disks, each with a 20 millisec access time, i.e. $MIPS = 14$, $N_D = 4$, and $T_{IO} = 20$. During table scans, it is assumed that multiple pages are fetched together in each read IO, as in IBM DB2 [Teng84]. To estimate $D_{i\beta_i}$, a prefetch blocking factor of 10 is used here for a table scan.

4. Performance Comparison

4.1 Alternative Strategies

The proposed integrated buffer management and query optimization strategy is compared with two alternative buffer management strategies. One strategy, referred to as strategy W, is based on the working set model, where a global LRU replacement algorithm is used to manage the buffer. For strategy W, the access plans for all queries are decided without consideration of available buffering, i.e. by assuming that referenced pages are prefetched from disks on successive scans. The other strategy, referred to as strategy H, is based on an enhanced hot set model. The hot set in a multi-join query is defined here to be the subset of the joining relations (if table scan is used) or the non-leaf parts of the indexes (if index scan is used) that can fit into a given fraction y of the total buffer and result in the most I/O reduction. In choosing the hot set, we consider different access path selections, index vs segment, and select the one that requires the least IO. That

is to say instead of selecting the hot set for an arbitrarily chosen access path, we choose the best access path and the corresponding hot set under a given buffer constraint. Still for strategy H, the join order is taken to be the same as that in strategy W to reduce the number of enumerations to be considered. Strategy H ensures that a buffer space of size equal to the query's hot set size can be allocated before a query gets executed. Otherwise the query has to be put into wait state. When additional buffer space becomes available, the longest waiting query whose hot set can be accommodated is scheduled. (An alternative is to schedule the query in FCFS order. For the example considered in this section, both scheduling schemes lead to very similar result.) For both strategies H and the integrated strategy, pages which are not in the hot set compete for buffer space, which are not pre-allocated to accommodate the hot sets, based on LRU. Note that for the nested-loop join example considered later in this section, the desired buffer allocation under DBMIN in [Chou85] for each relation based on the access path selected by Strategy H is similar to the buffer allocation Strategy H considered.

4.2 Simulation

A simulation program is developed to compare the three strategies under multi-way join queries. Although the integrated strategy can be applied to different join methods, to keep the comparison simple, nested-loop join is used as the join algorithm. Each join attribute is assumed to have an index. Even in this case, substantial differences in performance can be observed among the different strategies. The simulation attempts to capture the sequence in which tuples from different relations are scanned and compared, and in which pages are fetched from disks in a multi-way join. Two types of scan are simulated, table and index. In a table scan, under the uniform distribution assumption on attribute value, the number of tuples between matches has a geometrical distribution. (See [Chris84] for an analysis of the effects of assuming uniform distribution and attribute independence on performance projections.) In the simulation, instead of generating a random value for each tuple to carry out the join, we treat the number of tuples between matches as a random variable with geometric distribution and simulate the actions related to scanning and matching tuples. For example, scanning into a new page triggers an IO operation if the page is not in the buffer, finding a match in an outer relation initiates a scan into the inner relation, and reaching the end of scanning an inner relation returns the scan back to the outer relation. For index scans, the data page to be fetched is randomly selected as any one of the pages of the relation. Accessing of three index pages is required (assuming a three level index) as well as 1 data page. The simulation program generates the index page addresses assuming a B-tree index structure and captures the hit to re-referenced leave pages.

Transactions are placed on a wait list upon arrival. Each transaction has associated with it a value which is the least number of pages required on the free buffer list that will allow it to move to the CPU queue for execution. For the integrated strategy, this is the number of pages required to accommodate the relations to be kept in buffer, as determined by the optimization procedure. For strategy H, this is the number of pages in the hot set. There is no buffer requirement under strategy W. A transaction moves from the wait list to the CPU queue if it is the first in the wait list which has a buffer requirement less than the free page list and the multiprogramming level is not exceeded.

4.3 Example Problem

We illustrate the methodology using the following transaction processing environment with 14 transactions and 22 relations. Table 1 presents the cardinality and size of each relation. Table 2 describes the transaction characteristics. These include the number of joins and, for each join, the two relations involved and the selectivities on each join attribute, respectively. Also presented is the arrival frequency of each transaction. For example, transaction 1 has an arrival frequency of .07 Tx/sec and requires three joins. The join between relations R_2 and R_4, has a join selectivity on the join columns of both relations of .001.

Relation	Cardinality	No. of pages
1	500	90
2	1000	200
3	300	100
4	1200	50
5	500	70
6	1000	250
7	1000	60
8	100	100
9	500	400
10	200	100
11	4000	260
12	500	90
13	1000	200
14	300	100
15	1200	50
16	500	70
17	1000	250
18	1000	60
19	100	100
20	500	400
21	200	100
22	4000	260

Table 1. Relations for the example problem

The alternative access plans for a given join order are restricted for simplicity to using nested-loop join with the inner relations either kept in memory using table scan or not kept in memory using either table scan or index scan. If index scan is used, the non-leaf portion of the index will be kept in memory. The iterations required for the two step optimization is about six cycles to converge and consumes about 10 sec of CPU time on an IBM 3090 processor.

TX	Run Freq.	No. of Joins	Relations		Selectivities	
1	.07	3	2	4	.001	.001
			4	6	.001	.001
			6	8	.001	.01
2	.071	3	3	5	.004	.002
			5	7	.002	.001
			7	9	.001	.002
3	.11	3	1	4	.002	.001
			4	5	.001	.002
			5	7	.002	.001
4	.200	3	1	2	.002	.001
			2	8	.001	.010
			8	10	.010	.005
5	.06	4	3	5	.004	.002
			5	7	.002	.001
			7	9	.004	.002
			9	11	.002	.001
6	.210	1	1	4	.002	.001
7	.300	1	3	5	.004	.002
8	.072	3	13	15	.001	.001
			15	17	.001	.001
			17	19	.001	.01
9	.071	3	14	16	.004	.002
			16	18	.002	.001
			18	20	.001	.002
10	.11	3	12	15	.002	.001
			15	16	.001	.002
			16	18	.002	.001
11	.201	3	12	13	.002	.001
			13	19	.001	.010
			19	21	.010	.005
12	.062	4	14	16	.004	.002
			16	18	.002	.001
			18	20	.004	.002
			20	22	.002	.001
13	.210	1	12	15	.002	.001
14	.311	1	14	16	.004	.002

Table 2. Transactions for the example problem

In Figure 1, we plot the average response time from the simulation vs buffer size for the three strategies. We set $\alpha = 0.9$ for the integrated strategy and $\gamma = 0.65$ for strategy H, respectively. Note that these parameters, α, γ, are set at the optimum values for a 400 page buffer size. Sensitivities to these parameters are discussed later. The integrated strategy shows uniformly better performance over strategies W and H. This is especially the case when the buffer size is smaller (say, a 400 page buffer). Strategy H based on the hot set concept also shows significant improvement over strategy W using LRU, in agreement with [Sacca86]. This shows the importance of recognizing the semantics of the query and keeping the critical relations in the buffer. A simple LRU type strategy like strategy W, ignoring the query semantics, performs very badly. Still, strategy H is only a local optimization on each query. Imposing a uniform limit on hot set size over all queries can be suboptimal. The integrated strategy considers the behavior of all queries together to derive a decision on the amount of buffer to be allocated for the execution of each query.

Table 3 summarizes the access plan selection and buffer allocations for each of the three strategies when the buffer size is 400 pages. The optimization procedure to determine the access plan and buffer allocation requires about 6 iterations for this example. In the "Join Order" column, the first or leftmost relation is the outer most-relation in the join, and the last or the rightmost relation is the inner-most relation. In the "Scan Type" column, S and I represent table scan and index scan, respectively, for access of each of the relations presented according to the join order. In the "Buffer" column, y means the corresponding relation is kept in the buffer while n means that no effort is made to keep the relation in the buffer. For example, under the integrated strategy, the join order of transaction 1 is R_6, R_4, R_8, R_2, with R_2 the inner-most relation. All relations are scanned sequentially through table scan. All relations except R_6 are kept in the buffer. Note that under strategy W, no relation is pre-allocated any buffer space. A different join order may be chosen under the different strategies. Let us look specifically at transaction 2. Under the integrated strategy, R_7, R_5, and R_3 are kept in the buffer requiring a total of 230 pages, while R_9 with 400 pages is chosen to be the outer-most relation and is not kept in the buffer. Under strategy W, as no buffering of the relation is assumed, R_9 is instead chosen as the inner-most relation due to its low selectivity and the availability of an index. Under strategy H, the same join order is chosen as strategy W, but R_5 and R_7 are kept in the buffer. In Table 4, we compare some run time statistics between the integrated strategy and strategy H. Both the average IO's per transaction and the fraction of transactions delayed due to insufficient buffer size are substantially lower for the integrated strategy.

We observe that in the integrated strategy, the average response times determined from the queueing model in Section 3.2 and from the simulation are comfortably close to each other, differing by amounts within 20%. The difference is mainly due to the fact that the analysis does not capture some of the inter-transaction effect considered in the simulation. One is the IO reduction from buffer hit due to transactions referencing relations that were brought in by another transaction referencing the same relations. For the conditions of Figure 1, none of the strategies causes the transactions to read as many pages as it would if it ran alone on the system. The other is the delay due to temporary over-allocation of the

buffer. In the integrated strategy, if a relation to be kept in the buffer for the execution of a transaction cannot be accommodated, the transaction execution is delayed. As shown in Table 4, this delay occurs quite infrequently. As the response time estimate in the integrated strategy is only used in the optimization constraint to measure the buffer requirement, the inaccuracy can be more or less absorbed in the factor α. We have examined the sensitivity of the integrated strategy to α and found that the response time remains the same over a wide range of α from 0.6 to 0.9 for the example shown.

Strategy W			
Tx Type	Join Order	Scan Type(s,i)	Buffer (y,n)
1	8,6,4,2	s,i,i,i	n,n,n,n
2	3,5,7,9	s,i,i,i	n,n,n,n
3	5,4,1,7	s,i,i,i	n,n,n,n
4	8,2,1,10	s,i,i,i	n,n,n,n
5	3,5,7,9,11	s,i,i,i,i	n,n,n,n,n
6	1,4	s,i	n,n
7	3,5	s,i	n,n
8	19,17,15,13	s,i,i,i	n,n,n,n
9	14,16,18,20	s,i,i,i	n,n,n,n
10	16,15,12,18	s,i,i,i	n,n,n,n
11	19,13,12,21	s,i,i,i	n,n,n,n
12	14,16,18,20,22	s,i,i,i,i	n,n,n,n,n
13	12,15	s,i	n,n
14	14,16	s,i	n,n
Integrated Strategy			
Tx Type	Join Order	Scan Type(s,i)	Buffer (y,n)
1	6,4,8,2	s,s,s,s	n,y,y,y
2	9,7,5,3	s,s,s,s	n,y,y,y
3	1,4,5,7	s,s,s,s	n,y,y,y
4	2,1,8,10	s,s,s,s	n,y,y,y
5	9,7,5,3,11	s,s,s,s,i	n,y,y,y,n
6	1,4	s,s	n,y
7	3,5	s,s	n,y
8	17,15,19,13	s,s,s,s	n,y,y,y
9	20,18,16,14	s,s,s,s	n,y,y,y
10	12,15,16,18	s,s,s,s	n,y,y,y
11	13,12,19,21	s,s,s,s	n,y,y,y
12	20,22,18,16,14	s,i,s,s,s	n,n,y,y,y
13	12,15	s,s	n,y
14	14,16	s,s	n,y
Strategy H (γ = .65)			
Tx Type	Join Order	Scan Type(s,i)	Buffer (y,n)
1	8,6,4,2	s,i,s,s	n,n,y,y
2	3,5,7,9	s,s,s,i	n,y,y,n
3	5,4,1,7	s,s,s,s	n,y,y,y
4	8,2,1,10	s,i,s,s	n,n,y,y
5	3,5,7,9,11	s,s,s,i,i	n,y,y,n,n
6	1,4	s,s	n,y
7	3,5	s,s	n,y
8	19,17,15,13	s,i,s,s	n,n,y,y
9	14,16,18,20	s,s,s,i	n,y,y,n
10	16,15,12,18	s,s,s,s	n,y,y,y
11	19,13,12,21	s,i,s,s	n,n,y,y
12	14,16,18,20,22	s,s,s,i,i	n,y,y,n,n
13	12,15	s,s	n,y
14	14,16	s,s	n,y

Table 3. Access plan selection and buffer allocation

(400 page buffer)

Strategy	I/O per transaction	Fraction of Tx with delayed starts
Integrated	50	.02
Strategy H (γ = .65)	438	.42

Table 4. Run statistics for 400 page buffer

Although the integrated strategy is found to be quite insensitive to the parameter α, strategy H is very sensitive to the parameter γ. Figure 2 shows the sensitivity of response time to γ under strategy H for a 400 page buffer. The variation in response time with the choice of γ is quite substantial. The variability in the response time curves points out the weakness of a local optimization strategy. Keeping a relation in memory can reduce the response time of the transaction referencing it, but may also create a detrimental effect for other transactions. This can occur if the reduction in response time is not large enough so that buffer consumption, which is the product of response time and buffer allocation, increases. Figure 3 shows the case for a 600 page buffer. A big zigzag in response time is observed. The peak at $\gamma = .5$ comes about because in going from $\gamma = .4$ to $\gamma = .5$, query Q_{12} has been allowed to keep relation R_{22} of 260 pages in the buffer. The simulation shows that although the number of page reads for query Q_{12} has been reduced from 5000 to 4020, the fraction of queries with delayed starts has increased from .03 to .32. At $\gamma = .6$, Q_{12} can keep both relations R_{18} and R_{22} with a reduction in IO to 1521. Because of the substantial reduction in its response time and hence buffer consumption for Q_{12}, the fraction of delayed starts is reduced to .25 even with larger buffer allocation. Note that the optimum γ depends upon not only the workload but also the buffer size. This would make selecting γ a nontrivial job for a real environment.

5. Reducing the Size of the Optimization Problem

In an environment with a large number of query types which require multiple joins over several relations, the number of variables, X_{ij}, can become very large. We can eliminate many of the access plans by pre-analyzing each query type separately before applying the global optimization. An important observation is that for a given query, some access plan is essentially inferior in the sense that there exists another plan that consumes less resources and offers better or similar performance. Inferior plans should be eliminated from consideration in the optimization procedure described in Section 4. For a given query, let us define buffer consumption, P_j^B, under access plan j to be $B_j T_j$ where T_j and B_j are the execution time and buffer requirement of the query under plan j, respectively. Note that here we use execution time instead of response time. The additional queueing delay due to the impact of concurrently executing queries is ignored so we can examine each query separately. An access plan is feasible if the total buffer requirement is less than the total buffer size or some predefined maximum. We can state two simple rules to determine whether an access plan is inferior:

1. For any access plan i, if there exists another feasible

access plan j, such that $B_i = B_j$ and $T_i > T_j$, or $B_i > B_j$ and $T_i \geq T_j$, access plan i is regarded as an inferior plan.

2. For any access plan i, if there exists another feasible access plan j, such that $P_i^B > P_j^B$ and $T_i \geq T_j$, access plan i is regarded as an inferior plan.

The first rule indicates that a plan with a larger buffer allocation should achieve a smaller execution time. The second rule implies that a plan with a smaller buffer allocation and longer execution time is not necessarily desirable unless the buffer consumption is also smaller. Let access plan j_C using B_{j_C} buffer be the one with the minimum buffer consumption. Any plan with a larger buffer consumption than plan j_C should be rejected unless the extra buffer consumption leads to reduction in the execution time. Thus any plan using less than B_{j_C} buffer can be eliminated because it takes too little buffer so the long execution time results in large buffer consumption. Query strategies using more than B_{j_C} may achieve smaller response time at the expense of the buffer consumption. These strategies may be acceptable if there is sufficient buffer. This is to be determined by the global optimization procedure in Section 4. We need only to eliminate those where the response time improvement is small compared to the additional buffer used.

In Figure 4, we plot the execution time verses buffer allocation for transaction 1 of the Example problem described in Section 4. Several different access plan strategies correspond to each buffer size due to join-order permutations and the sum of the sizes of relations R_2 and R_8 being the same as the sum of the sizes of relations R_4 and R_6. Only the strategy with the smallest execution time for a given buffer allocation is plotted. Also, a buffer allocation is not considered unless a decrease in execution time can be realized compared with a smaller buffer allocation. That is to say, we only consider strategies which do not violate rule 1 stated above. Transaction 1 has 216 potential access plans, as there are 8 possible join orders (notice that not all permutations make sense, e.g. R_2, R_6, R_4, R_8) and each relation except the outer most can either be kept in buffer for table scan or not be kept in buffer while both table scan and index scan are possible alternative scanning strategies. The 216 candidate access plans of transaction 1 only result in five meaningful buffer allocations and access plans as follows:

1. 4 pages: join order R_8, R_6, R_4, R_2 with index scans on all except the outer most relation.
2. 50 pages: join order R_8, R_6, R_4, R_2 with R_4 kept in the buffer and index scans on R_6 and R_2
3. 150 pages: join order R_6, R_4, R_8, R_2 with R_4 and R_8 kept in the buffer and index scan on R_2
4. 250 pages: join order R_8, R_6, R_4, R_2 with R_2 and R_4 kept in the buffer and index scan on R_6.
5. 350 pages: join order R_6, R_4, R_8, R_2 with all pages of relations except the outer relation be kept in the buffer and accessed with table scans.

Figure 5 shows the plot of buffer consumption vs buffer size allocation for transaction 1. From rule 2, by comparing any two feasible strategies, the one with the higher buffer consumption can be eliminated unless it has the lower execution time. For a given buffer size of 400 pages, all five strategies are feasible as they all require less than 400 pages. Strategy 1 having 4 pages is the one with the minimum buffer consumption. By comparison with strategy 5, strategies 2, 3, and 4 are eliminated as they not only take longer to execute (Figure 4) but also have larger buffer consumption (Figure 5). Strategy 5, having 350 pages of allocated buffer, is still acceptable as its execution time is a lot smaller than strategy 1. Thus only strategies 1 and 5 need to be considered in the global optimization. The global optimization in Section 4 which considers all strategies has in fact picked out strategy 5. The number of strategies to be retained for global optimization from the other transactions of the illustrative example can be similarly reduced.

6. Summary

Existing query optimizers for relational databases attempt to minimize a linear combination of CPU and page IO costs for each query separately. The impact of buffer management is not reflected in the cost function. Although relational queries contain a lot of information on data references, it is generally not conveyed to the buffer manager by the query optimizer. In this paper, we propose and analyze a mathematical algorithm to integrate buffer management and query optimization. When making access plan selections, the integrated strategy explicitly considers whether a relation is kept in the buffer. The query strategies for all transactions in the workload are simultaneously determined to optimize an objective function on overall system performance with constraints on total buffer requirement. The mathematical formulation uses a two step approach combining an integer programming problem with a queueing network model to be solved iteratively. A detailed simulation is developed to study the performance improvement of the integrated strategy as compared with two other buffer management strategies based on the working set and hot set, respectively. The study shows that by ignoring all the reference information in the query, the LRU type strategy does very poorly. Explicitly taking into consideration the reference behavior from the query in buffer management, as in the hot set strategy, can improve the query performance. However, picking the right hot set size can be difficult. Integration of query optimization and buffer management to consider the requirement of all queries together gives substantial further improvement to the performance. This showed the importance of integrating of buffer management and query optimization strategies so that reference information from the queries can be captured to manage the buffer, and buffer availability can be reflected in the access path selection. Furthermore, each query can be pre-analyzed separately to reduce the number of access plans to be considered in the global optimization. We proposed an approach which eliminates an access plan if there exists another one which has less buffer consumption but similar or better performance. This pre-analysis was found to be able to greatly reduce the size of the optimization program.

Acknowledgement:

The authors would like to thank Guy Lohman for his suggestions on revising the paper.

References

[Astr76] M.M. Astrahan, M.W. Blasgen, D.D. Chamberlin, K.P. Eswaren, J.N. Gray, P.P. Griffith, W.F. King, R.A. Lorie, P.R. McJones, J.W. Mehl, G.F. Putzolu, I.L. Traiger, B.W. Wade, and V. Watson, "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976, pp. 97-137.

[Chou85] H.-T. Chou and D. J. DeWitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems", Proc. of the 11th International Conference on Very Large Data Bases, Stockholm, 1985, pp. 127-141.

[Chris84] S. Christodulakis, "Implications of certain assumptions in database performance evaluation", ACM Trans. Database Syst., Vol. 9, No. 2, June 1984, pp. 163-186.

[Codd70] E.F. Codd, "Relational Model of Data for Large Shared Data Banks", Comm. of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[Denn68] P. J. Denning "The Working Set Model for Program Behavior.", Comm. of the ACM, Vol. 11, No. 5, May 1968, pp. 323-333.

[Effe84] W. Effelberg and M. E. S. Loomis, "Logical, internal and physical reference behavior in CODASYL database systems", ACM Trans. Database Syst., Vol. 9, No.2, June 1984, pp. 187-213.

[Kirk83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", Science, Vol. 220, No. 4598, May 1983, pp. 671-680.

[Sacca82] G. Sacca and M. Schkolnick, "A Mechanism for Managing the Buffer Pool in a Relational Database System using the Hotset Model", Proc. of the 8th International Conference on Very Large Data Bases, Mexico City, Sept. 1982, pp. 257-262.

[Sacca86] G. Sacco and M. Schkolnick, " Buffer Management in Relational Database Systems", ACM Trans. on Database Syst., Vol. 11, No. 4, Dec. 1986, pp. 474-498.

[Seli79] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price, "Access Path Selection in a Relational Database Management System", Proc. of ACM SIGMOD 20, 1979, pp. 23-34.

[Ston81] M. Stonebraker, " Operating System Support for Database Management", Comm. of the ACM, Vol. 24, No. 7, July 1981, pp. 412-418.

[Teng84] T.Z. Teng and R.A. Gumaer, "Managing IBM Database 2 Buffers To Maximize Performance", IBM Systems Journal, Vol. 23, No. 2, 1984, pp. 211-218.

[Wolf88] J.L. Wolf, D.M. Dias, B.R. Iyer, and P.S. Yu, "A Hybrid Data Sharing-Data Partitioning Architecture for Transaction Processing", Proc. 4th Intl. Conference on Data Engineering, Los Angeles, CA, Feb. 1988, pp. 520-527.

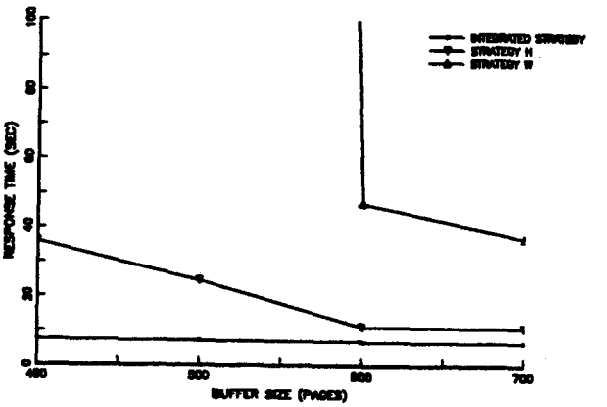

Fig. 1 Transaction response time vs buffer Size

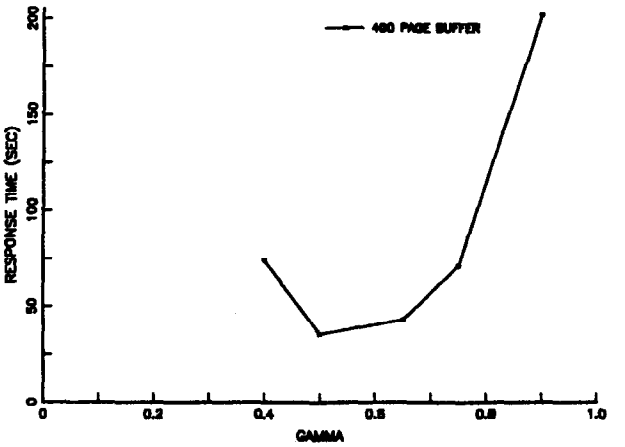

Fig. 2 Sensitivity of strategy II to γ (400 page buffer)

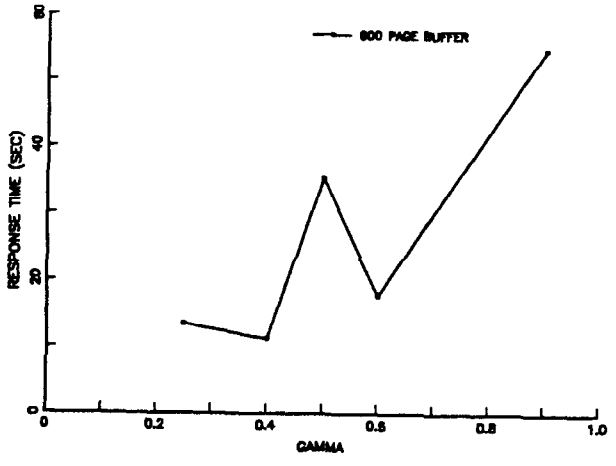

Fig. 3 Sensitivity of strategy II to γ (600 page buffer)

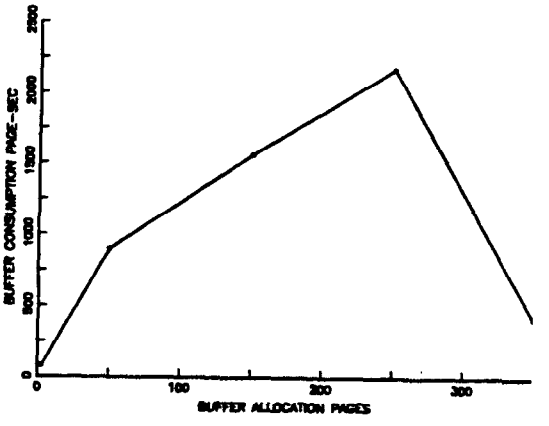

Fig. 5 Buffer consumption of transaction 1

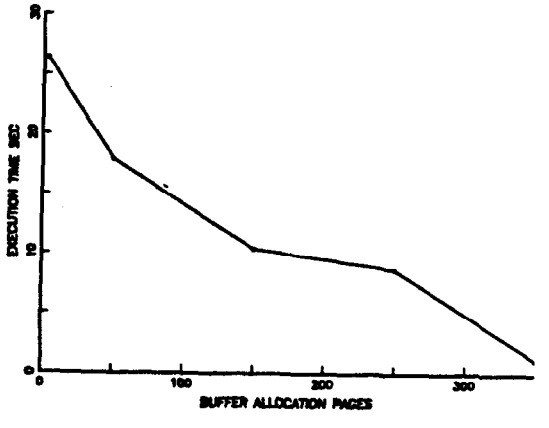

Fig. 4 Execution time of transaction 1