Optimal Design of Multiple Hash Tables for Concurrency Control

Ming-Syan Chen, *Senior Member*, *IEEE*, and Philip S. Yu, *Fellow*, *IEEE*

Abstract—In this paper, we propose the approach of using multiple hash tables for lock requests with different data access patterns to minimize the number of false contentions in a data sharing environment. We first derive some theoretical results on using multiple hash tables. Then, in light of these derivations, a two-step procedure to design multiple hash tables is developed. In the first step, data items are partitioned into a given number of groups. Each group of data items is associated with the use of a hash table in such a way that lock requests to data items in the same group will be hashed into the same hash table. In the second step, given an aggregate hash table size, the hash table size for each individual data group is optimally determined so as to minimize the number of false contentions. Some design examples and remarks on the proposed method are given. It is observed from real database systems that different data sets usually have their distinct data access patterns, thus resulting in an environment where this approach can offer significant performance improvement.

Index Terms—Concurrency control, hash algorithms, lock contentions, multiple hash tables.

1 Introduction

Due mainly to rapid technology advances, it has recently become very attractive to couple multiple systems for database transaction processing in order to answer the increasing demand for better capacity, availability and cost-performance. The advantages that such coupled systems can offer include better price performance, improved capacity and responsiveness due to parallelism, scalability, enhanced fault tolerance, and better dynamic load balancing [2], [9]. An example configuration of the coupled system is given in Fig. 1. A significant amount of research effort has investigated various issues for such data sharing environments, or related client/server paradigms, including the impact of data skew, global buffer management [3], [7], and concurrency control [5], [6], [8].

In a data sharing environment, different transactions running on different computing nodes may access the same data item. Each access to a shared data item has to be granted a required lock first to ensure correct concurrency control. To handle the lock granting process, it is common that a hash scheme is applied to each incoming lock request to see if that lock request has any contention with existing locks in the system [6]. It is noted that to deal with a large amount of incoming lock requests, hashing is, in general, more efficient than search methods such as B-tree search. A hash scheme utilizes a hash table which has a number of hash entries. The number of hash entries in a hash table is referred to as the size of the hash table. Which hash entry a data item to be locked will be mapped into is determined through the use of a hash function. As an example of a hash function, consider a hash table with 100 hash entries. A hash function for a data item, say page number k, could be $h(k) = k \bmod 100$. Thus, page number 732 would correspond to hash entry 32 in the hash table.

Depending on the nature of the operation intended, there are basically two lock types of a lock request, namely S (share) lock and X (exclusive) lock. S locks are used for data items to be read only, and X locks are needed for updating data items. Two locks to the same data item are called compatible only if both are S locks, and called incompatible, otherwise. Two lock requests are said to have a real contention if they are incompatibly locking the same data item, and said to have a false contention if they are for different data items but are incompatibly hashed into the same hash entry.[1] Although existing hash schemes effectively provide concurrency control, they may result in a high probability of false contention. In the above example, a false contention would occur if pages 732 and 1,432 were requested to be modified (i.e., two X locks) or one read from and one modified (i.e., one S and one X locks), since both would be incompatibly hashed into hash entry 32. Because they are two different pages, concurrent access to them would be allowed eventually. However, extra overhead is incurred in resolving the false contention; i.e., determining that the contention is false rather than real.

Note that in some existing database systems, the resource names of existing locks are not kept in the hash table. Instead, the hash table only records the occupancy of each hash entry by any existing lock and gives a pointer to the location (maybe located in some computing node) where more information can be found. Specifically, if an incoming lock is hashed into a hash entry which has been occupied by an existing lock, then the system with the incoming lock is guided

- *M.-S. Chen is with the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan, Republic of China. E-mail: mschen@cc.ee.ntu.edu.tw.*
- *P.S. Yu is with the IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. E-mail: psyu@watson.ibm.com.*

Manuscript received 22 Sept. 1994.
For information on obtaining reprints of this article, please send e-mail to: transkde@computer.org, and reference IEEECS Log Number 104394.0.

1. Note that a hash collision caused by two S lock requests for different data items usually incurs little overhead to resolve, and is not deemed as a false contention in this paper.

1041-4347/97/$10.00 © 1997 IEEE

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

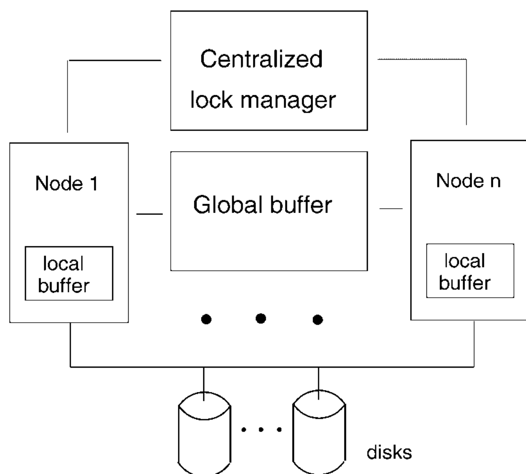


Fig. 1. A coupled system with the data sharing approach.

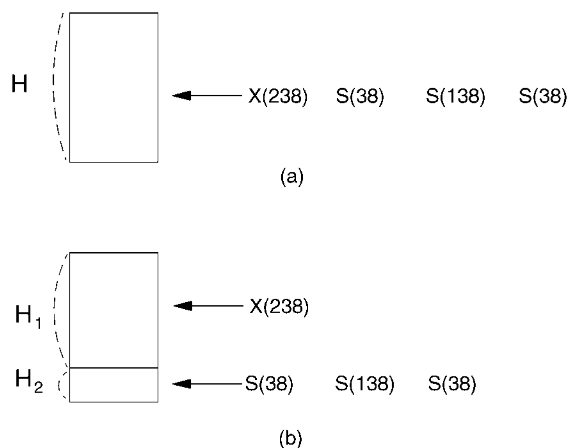


Fig. 2. Example of using multiple hash tables.

into a location where more information is available to decide whether there is a real contention or not. By avoiding storing the resource names in the hash table, not only is the design of a hash table simplified, but also the system possesses the flexibility of deploying the hash table together with other components in a pre-allocated memory space. The ratio between memory spaces required by the hash table and other components can then be adapted for specific customer needs.[2] In such a system, however, the overhead required to deal with false contention is usually very high, and could in fact be one of the major sources for causing the performance degradation for coupling multiple systems. It would greatly improve coupling efficiency if a hash scheme is so developed that the likelihood of false contention is minimized.[3] It is noted that one way to accomplish the goal of decreasing the likelihood of false contentions is to increase the hash table size. However, this solution may run counter to the purpose of a hash scheme to minimize the resource required to deal with concurrency control, since an increase in the hash table size would require a larger amount of memory space. Consequently, it is preferable to employ a hash scheme that minimizes the number of false contentions without overburdening resources.

Since S locks do not cause contentions among themselves, a write intensive workload will naturally need a larger hash table than a read intensive workload to maintain the same false contention level. Note that various incoming lock requests usually have their own distinct data access patterns. For example, in some workloads, one may have very intensive updates to data pages but have very few updates to index pages. As a result, locks to index pages are mostly S locks and those to data pages are mostly X locks. In view of this, we would like to partition the hash table for index and data pages separately, while allocating a larger portion of the hash table to data pages and a smaller one to index pages. As such, the feature that S locks to the same hash entry are compatible can be fully exploited and false contention caused by two S and X locks are reduced. The number of false contentions can therefore be minimized. For an illustrative example, consider the locking scenario in Fig. 2a where the number in each parenthesis means the data item, say page number, the hash table size is 100 and the hash function for page k is $h(k) = k \bmod 100$. It can be seen that three false contentions occur for the lock sequence X(238), S(38), S(138), and S(38) in the hash entry 38, which is first occupied by the exclusive lock X(238). Suppose that pages with page number less than or equal to 200 are index pages which are known to be mostly for read purposes.[4] Then, one could partition the hash table into two regions, as shown in Fig. 2b, in such a way that those false contentions involving these index pages can be avoided (by hashing them to H_2). An example hash function to achieve the effect demonstrated in Fig. 2b is $h(k) = (k \bmod 20) + 80$ if $k \leq 200$, and $h(k) = (k \bmod 80)$, otherwise.

It is noted that such an approach of partitioning a hash table for different lock requests to minimize the occurrence of false contention is not limited to the one based on index and data pages, and is, in fact, applicable to an environment where data items can be divided into a proper number of groups each of which has its own data access pattern. It is observed from some real traces that various data sets usually have their distinct data access patterns, which are obtainable from proper work load characterization [1]. As will be shown later, not only how to partition data items into groups, but also the way to assign the hash region size for each data group, as in Fig. 2b, has an important effect on the overall contention probability, and their solutions could lead to a significant performance improvement. For ease of presentation, the problem of partitioning a given hash table into separate hash regions for different data groups will be described as the one of using multiple hash tables in this paper. Consequently, we explore in this paper the approach of using multiple hash tables for lock requests with different data access patterns to minimize the number of false contentions. Specifically, we shall first derive some theoretical results

2. Such an advantage is made possible only if the space required by the hash table can be determined off-line. It is relatively difficult to achieve this feature if the resource names of existing locks have to be kept in the hash table.

3. For an obvious reason, the use of a hash table does not affect the occurrence of real contention.

4. In practice, it is usually feasible to identify index pages in the lock manager level (such as from the corresponding table ids).

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

on using multiple hash tables. Then, in light of these derivations, a two-step procedure to design multiple hash tables is developed. In the first step, data items are partitioned into a given number of groups. Each group of data items is associated with the use of a hash table in such a way that lock requests to data items in the same group will be hashed into the same hash table. In the second step, given an aggregate hash table size, the hash table size for each individual data group is determined so as to minimize the number of false contentions. Some design examples are given to illustrate our approach.

This paper is organized as follows. Some theoretical results on using multiple hash tables are derived in Section 2. A two-step design procedure is developed in Section 3. A complete design example and remarks are given in Section 4. This paper concludes with Section 5.

2 Mathematical Model for Using Multiple Hash Tables

We shall derive in this section some theoretical results on using multiple hash tables. Based on these results, a two-step design procedure will be developed in Section 3.

2.1 Problem Formulation

To use multiple hash tables, data items are partitioned into a given number of groups, say m groups, each of which corresponds to the use of a hash table. A lock request accessing a data item in group i is called a class i lock request. Then, m is the number of different classes of lock requests. Denote the arrival rate of a class i lock request as λ_i, and we have $\sum_{i=1}^{m} \lambda_i = \lambda$. Let p_i^X and p_i^S be the probabilities for a class i lock request to ask for an X lock and an S lock, respectively, and $p_i^X + p_i^S = 1$. Denote the number of total hash entries available as H. H_i is the number of hash entries to be employed by class i lock requests. Let T be the average lock holding time. According to Little's rule [4], the number of existing class i locks in the system can be approximated as $\lambda_i T$.

Two S locks to the same hash entry are compatible. Then, for a class i lock request with X lock type, assuming using a perfect hash function, the probability of having a contention can be approximated as $\frac{\lambda_i T}{H_i}$. On the other hand, for a class i lock request with S lock type, the probability of having a contention is approximated as $\frac{\lambda_i T p_i^X}{H_i}$. It is noted that, since an ideal hash function is used here, the formulated contention probability is a lower bound. However, such an assumption is made for ease of discussion, and is not essential for the usefulness of the proposed method. Hence, $f(H_1, H_2, \cdots, H_m)$, the overall contention probability can be expressed by,

$$f(H_1, H_2, \ldots, H_m) = \sum_{i=1}^{m} \left(P_i^X \frac{\lambda_i T}{H_i} + P_i^S \frac{\lambda_i P_i^X T}{H_i} \right) \frac{\lambda_i}{\lambda}$$

$$= \frac{T}{\lambda} \sum_{i=1}^{m} \frac{\lambda_i^2 p_i^X \left(1 + p_i^S\right)}{H_i}. \tag{1}$$

It is noted that λ_i, p_i^X, and p_i^S, $1 \le i \le m$, can be estimated from a given workload. Consequently, we want to determine the vector $<H_1, H_2, \cdots, H_m>$ that minimizes f in (1) in such a way that $\sum_{i=1}^{m} H_i = H$. It is worth mentioning that the value T could be affected by the locking scheme used. Specifically, better locking schemes will reduce the average lock holding time. Note that our derivation for the optimal partition of a hash table in the following subsection is only dependent upon the values of λ_i, p_i^X, and p_i^S, $1 \le i \le m$, and does not require that T be a constant. Explicitly, the average lock holding time T can be thought of as a monotonically increasing function of the contention probability f (i.e., a smaller contention probability leads to a smaller T). It then follows from (1) that T is a monotonically increasing function of

$$f_A = \frac{1}{\lambda} \sum_{i=1}^{m} \frac{\lambda_i^2 p_i^X \left(1 + p_i^S\right)}{H_i}.$$

Thus, the vector $<H_1, H_2, \cdots, H_m>$ to be derived, which minimizes f_A, will also minimize f in (1).

2.2 Deriving Optimal Solutions

To determine the vector $<H_1, H_2, \cdots, H_m>$ that minimizes f in (1), we equivalently want to solve the vector $<k_1, k_2, \cdots, k_m>$ that minimizes g in (2) below such that $\sum_{i=1}^{m} k_i = K$.

$$g(k_1, k_2, \ldots, k_m) = \sum_{i=1}^{m} \frac{a_i}{k_i}, \tag{2}$$

where $a_1, a_2, \cdots, a_m$ are given. By relating $g(k_1, k_2, \cdots, k_m)$ to $f(H_1, H_2, \cdots, H_m)$, it can be seen that K corresponds to H, k_i corresponds to H_i and a_i corresponds to $\lambda_i^2 p_i^X (1 + p_i^S)$. Note that with variables $k_1, k_2, \cdots, k_m$, the minimum of $g(k_1, k_2, \cdots, k_m)$ can be obtained by taking the first order derivatives against $k_1, k_2, \cdots, k_m$, and having $g'(\,) = 0$. As a consequence, it can be shown that the vector $<k_1, k_2, \cdots, k_m>$, in which

$$k_i = \frac{\sqrt{a_i} K}{\sum_{j=1}^{m} \sqrt{a_j}},$$

$1 \le i \le m$, will yield a minimum g for (2) with given $a_1, a_2, \cdots, a_m$. The minimal value of g is

$$g = \frac{\left(\sum_{j=1}^{m} \sqrt{a_j}\right)^2}{K}.$$

From the solution to (2), it then follows that

$$H_i = \frac{\lambda_i H \sqrt{p_i^X \left(1 + p_i^S\right)}}{\sum_{j=1}^{m} \lambda_j \sqrt{p_j^X \left(1 + p_j^S\right)}}, \tag{3}$$

$1 \le i \le m$, will minimize $f(H_1, H_2, \cdots, H_m)$, i.e., the contention probability, in (1). We then have

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

$$f = \frac{T\left(\sum_{j=1}^{m} \lambda_j \sqrt{p_j^X\left(1 + p_j^S\right)}\right)^2}{\lambda H}. \quad (4)$$

It will be shown in Section 4 that using (3) to allocate multiple hash tables for different lock requests, as opposed to using one hash table with size H to handle all lock requests, can significantly reduce the overall contention probability and thus improve the coupled system performance.

3 A Two-Step Procedure to Design Multiple Hash Tables

Based on the results derived in Section 3, a two-step design procedure for using multiple hash tables is developed in this section. Data items are partitioned into groups in this first step. In the second step, the hash table size for each data group is optimally determined so as to minimize the contention probability. Recall that m is the number of partitioned data groups. λ_i is the arrival rate of a class i lock request. p_i^X and p_i^S are the probabilities for a class i lock request to ask for an X lock and an S lock, respectively. A two-step design procedure is described by Procedure P below.

Procedure P: A two-step design procedure.

Step 1: Partition all data items (in a form of tables, etc.) into m groups so as to minimize $\sum_{j=1}^{m} \lambda_j \sqrt{p_j^X(1 + p_j^S)}$.

Each group, consisting of many data items, is associated with the use of a hash table (i.e., corresponding to one class of lock requests).

Step 2: Given an aggregate hash table size, determine the hash table size for each data group (i.e., H_i, $1 \le i \le m$).

Note that to use m hash tables, there are many ways conceivable to partition all data items into m groups and to associate each of them with a hash table. Step 1 of Procedure P is devised in light of (4) to minimize the false contention probability. Specifically, Step 1 can be performed by algorithm M_1 below.

Algorithm M_1: To derive m classes of lock requests.

/* Suppose access patterns for n data groups are available in the beginning. $n > m$. */

1) Sort these n groups in descending order of their update probabilities. Label these groups as G_i, $1 \le i \le n$, such that $p_j^X \ge p_k^X$ if $j \le k$, where p_i^X is the average update probability for locks accessing data items in group G_i.
2) Compute $\Delta_i = f(G_i \cup G_{i+1}) - f(G_i) - f(G_{i+1})$, $1 \le i \le n$, where $f(G_i) = \lambda_i \sqrt{p_i^X(1 + p_i^S)}$.
3) Determine the value i such that $\Delta_i = min_{1 \le j \le n}\{\Delta_j\}$.
4) Merge groups G_i and G_{i+1} into G_i.
   Let $P_i^X = \frac{\lambda_i}{\lambda_i + \lambda_{i+1}} P_i^X + \frac{\lambda_{i+1}}{\lambda_i + \lambda_{i+1}} P_{i+1}^X$,
   $P_i^S = 1 - P_i^X$, and $\lambda_i = \lambda_i + \lambda_{i+1}$.
5) Relabel group G_{j+1} as G_j, for $j = i + 1, n - 1$.
   Let $n = n - 1$.
6) If $n = m$ Stop else Goto (2).

Basically, algorithm M_1 is greedy in nature and merges groups based on their similarity of update probabilities. After the sorting in Step 1 of M_1, the two groups whose merger would result in the minimal increase in the corresponding false contention probability will be merged (Step 2 to Step 5 of M_1). This process continues until m final groups are reached. It can be verified that the value of Δ_i in Step 2 of M_1 is always nonnegative. That is, merging groups will not decrease contention probabilities. The targeted number of lock request classes, m, could be decided in view of the allowable complexity of the global hash function. The scenario of merging n original data groups into m groups is illustrated in Fig. 3. In addition, Step 2 of Procedure P can be performed by algorithm M_2, which is devised based on (3).

Algorithm M_2: To determine the size of each hash table H_i, for $i = 1, m$.

1) The hash table size employed by class i lock request is determined by, for $i = 1, m$,

$$\lfloor H_i \rfloor = \frac{\lambda_i H \sqrt{p_i^X\left(1 + p_i^S\right)}}{\sum_{j=1}^{m} \lambda_j \sqrt{p_j^X\left(1 + p_j^S\right)}},$$

   where $\lfloor H_i \rfloor$ (i.e., flooring function) means the largest integer less than or equal to H_i.
2) Allocate the remaining $q = H - \sum_{i=1}^{m} H_i$ hash entries to the first q hash tables, i.e., $H_j = H_j + 1$ for $j = 1, q$.

It is worth mentioning that since the hash table size is usually very large (more than 10,000 in general) [1], [6], the application of flooring functions in Step 1 of M_2 has little impact on the optimality of the solution. To illustrate the two-step procedure devised in this section, a complete design example is given in Section 4.

4 Design Example and Remarks

We shall present a design example in Section 4.1. Some remarks on the application and performance of this approach are given in Section 4.2.

4.1 Design Example

Consider a database system where lock requests are made against six tables. Suppose that three hash tables are to be

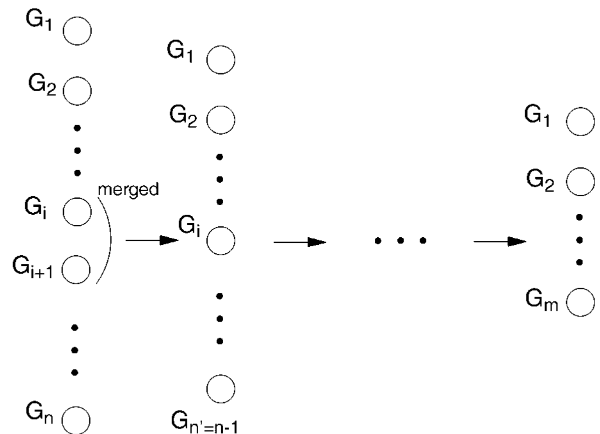


Fig. 3. Merging n original data groups into m groups.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

employed to handle the lock requests. Also, for illustrative purposes, assume that the data access patterns, in terms of lock arrival rates and update probabilities, for data in these tables are those as given in Table 1.[5]

TABLE 1
THE ORIGINAL SIX GROUPS OF DATA ITEMS

Tables	arrival rate	update prob.	read-only prob.
table 1	240	.91	.09
table 2	200	.03	.97
table 3	100	.30	.70
table 4	140	.95	.05
table 5	200	.01	.99
table 6	120	.88	.12

After the first step of algorithm M_1, these six groups (tables) are sorted according to their update probabilities. We then obtain the profile in Table 2. From Table 2 and Step 2 of M_1, it can be obtained that $\Delta_1 = f(G_1 \cup G_2) - f(G_1) - f(G_2) = 378.92 - 139.82 - 237.93 = 1.17$. Similarly, $\Delta_2 = f(G_2 \cup G_3) - f(G_2) - f(G_3) = 1.13$, $\Delta_3 = 12.62$, $\Delta_4 = 22.46$, and $\Delta_5 = 2.76$. Since Δ_2 is the minimum among them, we merge G_2 and G_3 (i.e., tables 1 and 6), and obtain Table 3 after the calculation in Step 4 of M_1. Following the above procedure, we can obtain the profile given in Table 4 after further merging.

After data items are partitioned into three groups according to M_1, the hash table size for each group can be determined by M_2, with a given aggregate hash table size. Performance improvement achieved by this approach can be seen by considering the following two cases.

Case 1: Use one hash table (i.e., the conventional approach).

The average update probability, P^X, can be obtained by combining those from these three groups. Thus, $p^X = (500 * 0.914 + 100 * 0.3 + 400 * 0.02)/1{,}000 = 0.495$, and $p^S = 1 - 0.495 = 0.505$. $\lambda = 500 + 100 + 400 = 1{,}000$. From the same reasoning as in Section 2.1, we get the expected false contention probability $f_1 = \frac{1000T(0.505 * (1+0.495))}{H} = \frac{745T}{H}$. For example, with $H = 65{,}536$ and $T = 0.5$sec, we have $f_1 = 0.54$ percent.

Case 2: Use three hash tables (i.e., the proposed approach).

From the profile in Table 4 and (3) in Section 2.2, we get $H_1 = .7673H$, $H_2 = .1101H$, and $H_3 = .1226H$, as shown in Fig. 4. For example, with $H = 65{,}536$ we get $H_1 = 50{,}286$, $H_2 = 7{,}216$, and $H_3 = 8{,}034$ by algorithm M_2. Then, it can be obtained from (4) that the corresponding false contention probability

$$f_2 = \frac{T\left(500\sqrt{0.914(1+0.86)} + 100\sqrt{0.3(1+.7)} + 400\sqrt{0.20(1+.98)}\right)^2}{1000H} = \frac{421T}{H}.$$

With $H = 65{,}536$ and $T = 0.5$sec, we have $f_2 = 0.32$ percent. It can be verified that $f_2 \simeq 56$ percent of f_1, showing a significant performance improvement by the proposed method. Based on the original profile in Table 1, the effect of using three hash tables for different aggregate hash table sizes is analytically shown in Fig. 5. It can be seen from Fig. 5 that in addition to increasing the hash table size, using multiple hash tables is an effective method to reduce the false contention probability.

For comparison purposes, consider a naive method of using three hash tables where data items are partitioned into three groups based on the order of their table ids and the size of each hash table is determined by equally partitioning the aggregate hash table size (i.e., $H_1 = H_2 = H_3 = \frac{H}{3}$). From Table 1, we can derive the corresponding profile for this method, which is given in Table 5. It can be obtained from Table 5 that without using the proposed two-step procedure, this naive method of using three hash tables leads to a contention probability

5. In many database applications, such access statistics can be made available from the access history in the table level.

TABLE 2
THE SIX DATA GROUPS AFTER SORTING

Groups	λ_i	P_i^X	P_i^S	$f(G_i)$
G_1 (tb 4)	140	.95	.05	139.82
G_2 (tb 1)	240	.91	.09	237.93
G_3 (tb 6)	120	.88	.12	119.13
G_4 (tb 3)	100	.30	.70	71.41
G_5 (tb 2)	200	.03	.97	48.62
G_6 (tb 5)	200	.01	.99	28.21

TABLE 3
THE FIVE DATA GROUPS AFTER MERGING TABLES 1 AND 6

Groups	λ_i	P_i^X	P_i^S	$f(G_i)$
G_1 (tb 4)	140	.95	.05	139.82
G_2 (tb's 1, 6)	360	.90	.10	358.19
G_3 (tb 3)	100	.30	.70	71.41
G_4 (tb 2)	200	.03	.97	48.62
G_5 (tb 5)	200	.01	.99	28.21

TABLE 4
THE THREE DATA GROUPS FOR USING THREE HASH TABLES

Groups	λ_i	P_i^X	P_i^S	$f(G_i)$	Hash table size
G_1 (*tb's* 1, 4, 6)	500	.914	.086	498.14	$0.7673H$
G_2 (*tb* 3)	100	.30	.70	71.41	$0.1101H$
G_3 (*tb's* 2, 5)	400	.02	.98	79.59	$0.1226H$

TABLE 5
A NAÏVE METHOD TO FORM THREE DATA GROUPS

Groups	λ_i	P_i^X	P_i^S	$f(G_i)$	Hash table size
G_1 (tb's 1, 2)	440	.51	.49	383.55	$0.44H$
G_2 (tb's 3, 4)	240	.68	.32	227.31	$0.24H$
G_3 (tb's 5, 6)	320	.33	.67	239.34	$0.32H$

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

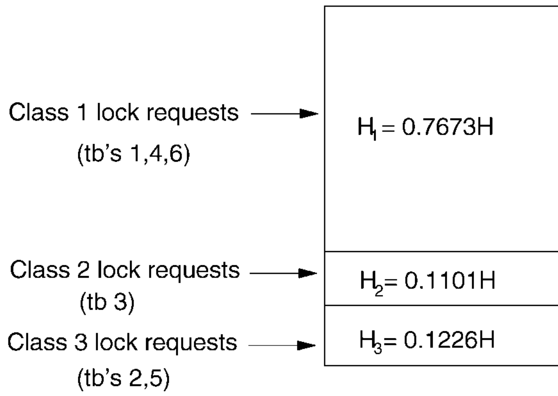


Fig. 4. The three hash tables resulting from the profile in Table 4.

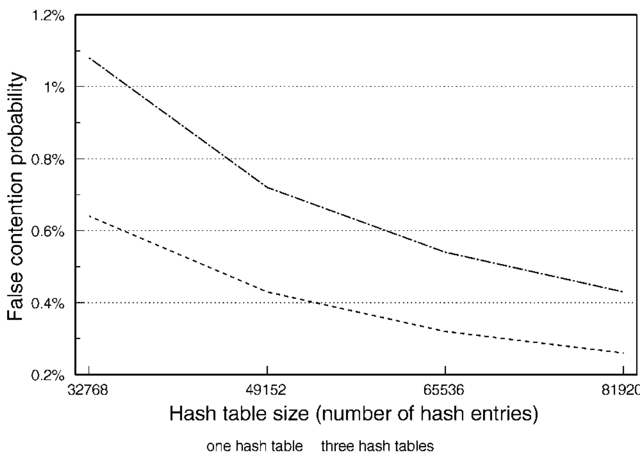


Fig. 5. Performance improvement by using three hash tables.

$$f_3 = \frac{3T}{\lambda H}\sum_{i=1}^{3}\lambda_i^2 p_i^X\left(1+p_i^S\right) = \frac{768T}{H},$$

even larger than f_1. This example indicates the necessity of employing the proposed method to design multiple hash tables.

4.2 Applications and Remarks

Note that the concept of using multiple hash tables can be practically used without much overhead. In fact, we have implemented this method in a data sharing system simulator [1] and obtained results which are very consistent with what are projected by our analytical models. Moreover, a simple variation of this approach has been used by one database product, and is shown to result in a significant performance improvement. The benefit from reducing false contention (and thus reducing the corresponding pathlengths required) by this approach clearly outweighed the corresponding overhead required to employ multiple hash tables. It is noted that in real database systems, it is common to have some data groups that have distinct data access patterns, thus resulting in an environment where this approach can offer prominent performance improvement.

For a straightforward application, let λ_I and λ_D be, respectively, the arrival rates of lock requests to index and data pages. p_I^X and p_I^S are the probabilities for an index page lock request to ask for an X lock and an S lock, respectively. p_D^X and p_D^S are defined similarly. Then, the optimal partition of a hash table size H for H_I and H_D is,

$$H_I = \frac{\lambda_I H\sqrt{p_I^X\left(1+p_I^S\right)}}{\lambda_I\sqrt{p_I^X\left(1+p_I^S\right)}+\lambda_D\sqrt{p_D^X\left(1+p_D^S\right)}}$$

and

$$H_D = \frac{\lambda_D H\sqrt{p_D^X\left(1+p_D^S\right)}}{\lambda_I\sqrt{p_I^X\left(1+p_I^S\right)}+\lambda_D\sqrt{p_D^X\left(1+p_D^S\right)}}$$

It is worth mentioning that the advantage of the approach of using multiple hash tables mainly stems from the feature that lock requests to different data groups have distinct data access patterns. The more different the access patterns to different groups are, the more improvement achievable by this approach. To provide more insights into the performance of this approach, consider the case of partitioning data items into two groups which have the same arrival rates (i.e., $\lambda_1 = \lambda_2 = \lambda/2$). Thus, the lock update probabilities of these two groups are P_1^X and $P_2^X = 2P^X - P_1^X$, where P^X is the average update probability of all data items. Let ρ be the ratio of "the false contention probability resulting from using two hash tables" to "the original false contention probability resulting from using one hash table." After some algebraic manipulations based on (4), ρ can be formulated as

$$\rho = \frac{\left(\sqrt{p_1^X\left(2-p_1^X\right)}+\sqrt{\left(2p^X-p_1^X\right)\left(2-2P^X+p_1^X\right)}\right)^2}{4P^X\left(2-P^X\right)}. \quad (5)$$

Thus, for a workload with a given lock update probability P^X, the performance improvement by using two hash tables can be assessed by (5). For example, with $P^X = 0.5$ we get,

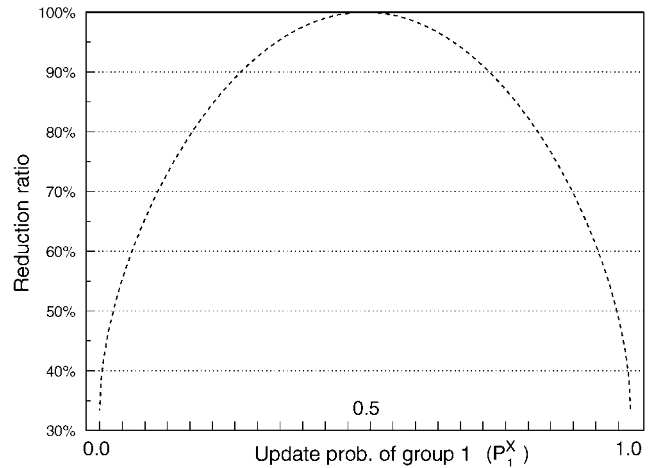


Fig. 6. Performance improvement by using two hash tables based on (6) (i.e., $P^X = 0.5$).

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.

$$\rho = \frac{\left(\sqrt{p_1^X\left(2 - p_1^X\right)} + \sqrt{\left(1 - p_1^X\right)\left(1 + p_1^X\right)}\right)^2}{3}, \qquad (6)$$

whose values are shown in Fig. 6, where the abscissa is the value of P_1^X and the ordinate is the reduction ratio ρ. It can be seen from Fig. 6 that there is no performance improvement when $P_1^X = 0.5$, since in that case P_2^X is also 0.5 and there is no difference between the update probabilities of these two groups. However, when the values of P_1^X and $P_2^X = 1 - P_1^X$ are further apart, meaning that data access patterns of these two groups are more different from each other, the better (i.e., smaller) reduction ratio is achievable by this approach.

5 Conclusion

In this paper, we proposed the approach of using multiple hash tables for lock requests with different data access patterns to minimize the number of false contentions in a data sharing environment. We have derived some theoretical results on using multiple hash tables. Then, based on these results, a two-step procedure to design multiple hash tables was developed. In the first step, data items were partitioned into a given number of groups, each of which corresponds to the use of a hash table. In the second step, given an aggregate hash table size, the hash table size for each data group was optimally determined so as to minimize the false contention probability. Some design examples and remarks on the proposed method were given. It has been observed from real database systems that different data sets usually have their distinct data access patterns, thus resulting in an environment where this approach can offer significant performance improvement.

Acknowledgments

Ming-Syan Chen is supported in part by the National Science Council, Project No. NSC 86-2621-E-002-023-T, Taiwan, Republic of China.

References

[1] M.-S. Chen, P.S. Yu, and T.-H. Yang, “On coupling Multiple Systems with a Global Buffer,” *IEEE Trans. Knowledge and Data Eng.*, vol. 8, no. 2, pp. 339-344, Apr. 1996.

[2] IBM Corporation, “Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex,” Technical Report GC28-1208-00, Apr. 1994.

[3] M.J. Franklin, M.J. Cary, and M. Livny, “Global Memory Management in Client Server DBMS Architecture,” *Proc. 18th Int'l Conf. Very Large Data Bases*, pp. 596-609, Aub. 1992.

[4] L. Kleinrock, *Queueing Systems*. Wiley, 1975.

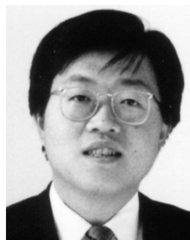
[5] E. Rahm, “Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems,” *ACM Trans. Database Systems*, vol. 18, no. 2, pp. 333-377, June 1993.

[6] D.R. Ries and M. Stonebraker, “Locking Granularity Revisited,” *ACM Trans. Database Systems*, vol. 4, no. 2, pp. 210-227, June 1979.

[7] Y. Wang and L.A. Rowe, “Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture,” *Proc. ACM SIGMOD*, pp. 367-376, May 1991.

[8] P.S. Yu, D.M. Dias, and S.S. Lavenberg, “On the Analytical Modeling of Database Concurrency Control,” *J. ACM*, vol. 40, no. 4, pp. 831-872, Sept. 1993.

[9] P.S. Yu, D.M. Dias, J.T. Robinson, B.R. Iyer, and D.W. Cornell, “On Coupling Multi-Systems through Data Sharing,” *Proc. IEEE*, vol. 75, no. 5, pp. 573-587, May 1987.

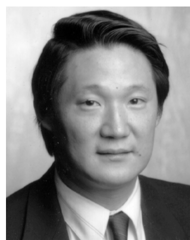


Ming-Syan Chen (S'87-M'88-SM'93) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1982, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1985 and 1988, respectively.

Dr. Chen was a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, New York, from 1988 to 1996, primarily involved in projects related to parallel databases, multimedia systems, and Internet applications. He is currently a faculty member of the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan. He has published more than 75 refereed international journal/conference papers in the areas of parallel and distributed database systems, query processing, data mining, and multimedia systems.

Dr. Chen is currently serving on the editorial board of *IEEE Transactions on Knowledge and Data Engineering*. In addition to serving as a program committee member for many conferences, Dr. Chen was a tutorial speaker on parallel databases at the 11th IEEE International Conference on Data Engineering in 1995 and was a guest co-editor for *IEEE Transactions on Knowledge and Data Engineering* for a special issue on data mining in December 1996. He holds, or has applied for, 17 U.S. patents in the areas of interactive video playout, video server design, and concurrency and coherency control protocols. He received the Outstanding Innovation award from IBM Corporation in 1994 for his contribution to parallel transaction design and implementation for a major database product, and has received numerous awards for his inventions and patent applications.

Dr. Chen is a senior member of the IEEE and a member of the ACM.



Philip S. Yu (S'76-M'78-SM'87-F'93) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China, in 1972; the MS and PhD degrees in electrical engineering from Stanford University in 1976 and 1978, respectively; and the MBA degree from New York University in 1982.

Since 1978, he has been with the IBM T.J. Watson Research Center, Yorktown Heights, New York. He is currently the manager of the Software Tools and Techniques Group in the Internet Technology Department. One of the focuses of the project is development of algorithms and tools for Internet applications, such as Web usage mining tool. His current research interests include database systems, data mining, Internet applications, multimedia systems, parallel and distributed processing, disk arrays, computer architecture, performance modeling, and workload analysis. Dr. Yu has published more than 200 papers in refereed journals and conferences, and more than 140 research reports and 90 invention disclosures. He holds or has applied for 42 U.S. patents.

Dr. Yu is a fellow of the ACM and the IEEE. He was an editor of *IEEE Transactions on Knowledge and Data Engineering* and also a guest coeditor of a special issue on mining databases. In addition to serving as a program committee member on various conferences, he was the program cochair of the Second International Workshop on Research Interests on Data Engineering: Transaction and Query Processing. He will be serving as the general chair of the 14th International Conference on Data Engineering. He has received several IBM and external honors, including Best Paper awards, two IBM Outstanding Innovation awards, an Outstanding Technical Achievement award, a research Division award, and 18 Invention Achievement awards.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:24 from IEEE Xplore. Restrictions apply.